요구사항명세(SRS)

Project : Nox

작성자 : 조 수 운

검수자 서명

1. 모든 이해 당사자들은 아래 명세의 내용을 충분히 이해하고, 면밀히 검토하였다.
2. 구현 결과물 및 테스트 결과물은 이 문서에 적힌 명세의 내용을 올바르게 반영하여야 한다.
3. 명세 내용을 변경할 경우, 변경한 내용이 프로젝트의 수행 비용에 미칠 영향력에 대해 평가할 것을 강력히 권고한다.
4. 명세에 대한 변경사항이 있는 경우, 그 변경 내용을 반드시 문서에 반영해야 한다.

관계	부서	직위	이름	서명

일러두기

◆ 내용에 있어 각 문장의 글씨 색깔이 의미하는 바는 대략 이렇다.

- **검은색**
: 일반적인 내용의 서술에 사용한다. 강조하는 의미로 **굵은 글씨**를 사용한 부분도 있다.

- **빨강색**
: 내용을 읽을 때, 특별한 주의점을 나타내거나 강한 강조를 위해, 주목을 끌 목적으로 사용한다. 주로 부정적인 강조일 때 사용한다.

- **파란색**
: 내용을 읽을 때, 특별한 주의점을 나타내거나 강한 강조를 위해, 주목을 끌 목적으로 사용한다. 주로 긍정적인 강조일 때 사용한다.

- **연한 파란색**
: 약한 강조를 위해 사용한다. 긍정 / 부정적인 의미는 크게 구분하지 않고, 알아두면 좋을 만한 사항에 주로 사용한다.

- **진한 빨강**
: 현재 명세에는 해당하는 사항이 없거나, 혹은 아직 명확하게 정의되지 않은 내용이 있음을 나타낸다.

- **보라색**
: 외부 문서에 대한 참조는 이 색깔로 표기한다.

- **분홍색**
: 이 문서의 어떤 명세 항목에서 문서 내 다른 명세 항목을 참조해야 하는 경우, 참조할 명세 번호를 이 색깔로 표기한다.

- **황갈색**
: 작성자, 작성 시점, 추가 / 삭제 / 재정의 관련 처리 내용을 표기한다.

- **파란 음영 + 짙은 파란색**
: 명세 항목의 세부적인 내용을 나타낸다.
명세 항목의 개요 부분은 음영을 넣지 않고, 세부 내용에만 파란 음영을 넣는다.

- **녹색 음영 + 녹색**
 : **기술 노트.**
 (프로그래밍의)기술적인 힌트나, 왜 이렇게 구현을 했는가에 대한 의도 등은 이런 형식으로 표시한다. 이 색깔로 표시된 문장은, 읽기 싫은 사람이나 기술적인 용어가 짜증나는 사람들은 안 읽어도 상관없다.

- **주황색 음영 + 갈색**
 : **테스트 노트.**
 명세 및 구현 내용에 대해 테스트해야 할 경우, 테스트와 관련된 상세한 내용을 나열한다.

- **보라색 음영 + 보라색**
 : **마케팅 노트.**
 운영 / 과금 / 라이선스 등의 정책과 관련하여 알아두면 좋을 상세 내용을 나열한다.

- 회색 음영
 ~~삭제된 기능은 이렇게 삭제 표시가 된 채, 회색 음영으로 내용을 감싼다. (음영은 경우에 따라 넣지 않을 수 있다.)~~
 ~~만일 내용 중에 이미 음영 처리가 된 부분이 있다면, 그 부분만 회색 음영으로 감싸서 제외되었음을 표시한다.~~
 ~~그러므로, 명세 기능을 삭제하는 경우가 아니라면 이 색깔의 음영을 사용하지 말 것.~~

※ 근데 막상 작성하다 보면, 이런 색상 규칙 같은 거 잘 안 지키게 된다.-_-
그냥 이런 게 있다 생각하고, 글자 색상과 의미가 반드시 일치하는지 따지지는 말 것...

◆ **항상 일러두기를 최신으로 유지하려고는 하는데, 간혹 내용이 틀릴 수도 있다. 그러니 일러두기의 형식을 지나치게 의식하는 것은 좋지 않다.**

그냥! 편하게 읽으면 된다.(...) 편하게...

목　차

A. 개 요

A-1. Concept

◆ A-1-1. 게임 특징

- **A-1-1-1.** 플레이어가 직접 조종하거나, 혹은 자동화 된 전투로도 진행할 수 있는, **모바일 플랫폼의 액션 RPG**

※ 좀 더 직설적으로 표현하자면, "모바일 플랫폼의 Diablo Ⅲ"와 같은 게임이다.

- **A-1-1-2.** 다양한 게임 모드를 지원한다.

1. 스토리 모드
: 여러 개의 독립적인 스테이지로 구성된 던전 돌파 형 게임 구성을 가지고 있다.

2. 무작위한 던전 모드
: 무작위로 만들어지는 지형 맵 + 무작위로 배치되는 몬스터들로 구성하는 특수한 스테이지 모드이다. 최고 레벨 플레이어들이 더 좋은 아이템과 스킬, 재화들을 얻기 위해 도전하는 모드. 로 구성한다.

3. 결투장(P vs P 컨텐츠)
: 다른 플레이어들의 캐릭터와 대결하는 모드

4. 월드 보스 몬스터 공략전(P vs E 컨텐츠)
: 게임에 접속해 있는 모든 플레이어들과 협동해서 게임 세계에서 하나만 존재하는 보스 몬스터를 공략하는 모드이다.
참가하는 플레이어들은 월드 보스 몬스터를 공략한 데 대한 기여도를 바탕으로 보상을 받는다.

5. 길드 전쟁(R vs R 컨텐츠)
: 플레이어들은 연합 세력으로 길드(Guild)를 창설할 수 있다.

- **A-1-1-3.** 플레이어는 언제든지 전투를 비롯한 플레이 과정을 자동적으로 진행할 수 있다.
: 사실, 이러한 기능은 최근 모바일 RPG들은 거의 '기본사양'에 가깝게 지원하는 기능이긴 하다.

- **A-1-1-4.** 비록 **자동으로 플레이를 진행하더라도, 플레이어가 중간중간 수동 조작을 통해 플레이에 개입할 수 있다.**

: 플레이어의 수동 조작에 의한 입력 이벤트가 다 끝나면, 인공지능이 현재 입력이 없음을 판단하고, 다시 자동 진행 플레이 방식으로 되돌아간다.

- **A-1-1-5.** 진행 방법을 알기 쉬운 게임 플레이 방식
: 게임 진행은 단방향성을 가지고 있다.
복잡한 상호작용이나 선택 과정을 주기보다는, **그냥 열심히 몬스터 때려잡으면 다 진행되는 구조**로 제작하고자 한다.

- **A-1-1-6.** 같은 종류와 외형을 가진 아이템이라도, **그 아이템마다 개별적인 능력 속성들은 다양하고 무작위적으로 조합**되어 있다.

※ 아이템을 획득할 때마다, 중복되어 있는 능력을 가진 아이템을 만날 확률이 거의 없으며, 플레이어로 하여금 항상 더 좋은 능력들의 조합으로 구성된 아이템을 찾도록 유도한다.

- **A-1-1-7.** 모든 스킬들은 스킬 고유의 능력에 대해 어느 정도의 기본 골격을 공유하지만, 세부적인 능력 속성의 수치들은 조금씩 달라질 수 있다.
: 이를테면, 캐릭터의 공격 스킬에도 부가적인 능력이 다양하게 합성될 수 있기 때문에, **같은 종류의 스킬이라도, 캐릭터마다 세부적인 능력과 사용할 때의 체감이 달라질 수 있다.**

※ 이를 통해 '같은 종류의 스킬'임에도 불구하고 더 좋은 능력의 조합을 가지고 있는 스킬들을 찾고자 하는 플레이 방식을 유도한다.

- **A-1-1-8.** 각 아이템과 스킬은 강화, 무작위 뽑기, 합성이 가능하다.
: 이 부분은 게임의 주된 과금 모델, 비즈니스 모델로 판단하고 있다.

◆ A-1-2. 핵심 컨텐츠

- **A-1-2-1.** **스테이지 클리어를 통한 보상**
: 아이템, 재화를 무작위 뽑기를 통한 획득 방식(가챠, Gacha)으로 플레이어에게 준다.

- **A-1-2-2.** **아이템 / 스킬의 등급과 능력에 대한 강화**
: 아이템과 스킬은 능력 속성의 강함과 희귀도에 따라 등급이 나뉘어져 있고, 비용을 지불해서 등급과 능력 속성에 대해 강화를 시도할 수 있다.
일반적으로 더 많은 비용을 지불할수록 더 희귀하고 강력한 아이템과 스킬을 획득할 수 있다.

- **A-1-2-3.** **아이템 / 스킬 능력 속성의 전환**
: 아이템과 스킬의 능력 속성들은 비용을 지불하고 다른 속성으로 전환할 수 있다.
전환 과정 그 자체는 완전히 무작위적이어서, 원하는 옵션이 등장할 때까지 여러 번 반복해야

할 수 있다.

- **A-1-2-4.** **혼돈 던전 모드**
: 스토리 모드에서 사용했던 스테이지 중에서 무작위적인 스테이지가 등장하고, 몬스터들도 무작위하게 등장한다.
유일한 규칙은, 스테이지의 마지막 지점에는 반드시 무작위하게 생성된 보스 몬스터가 존재하며, 이 몬스터를 쓰러뜨리면, 그 혼돈 던전 스테이지에서 승리하게 된다.
[향후 업데이트 기능임]

- **A-1-2-5.** **월드 보스 레이드**
: 현실 시간으로 일정한 기간마다 생성되는 강력한 월드 보스 몬스터를, 게임에 접속한 전체 플레이어들과 함께 사냥한다.
월드 보스 몬스터를 처치하는데 기여한 바에 따라 각 플레이어들은 보상을 획득한다.
[향후 업데이트 기능임]

- **A-1-2-6.** **결투장**
: 플레이어 캐릭터끼리만 편을 갈라서 서로 대결하는 컨텐츠이다.
결투를 위해 특별히 고안된 스테이지를 사용하며, 이러한 스테이지들은 향후 업데이트 상황에 따라서, 여러 개일 수도 있다.
대결 성적은 랭킹을 매겨서 관리한다.
[향후 업데이트 기능임]

- **A-1-2-7.** **길드 전쟁**
: 플레이어가 길드에 참여하면, 그 길드들끼리 전쟁을 할 수 있다.
길드 전쟁을 위해 특별히 고안된 스테이지를 사용하며, 이러한 스테이지는 향후 업데이트 상황에 따라서, 여러 개가 될 수도 있다.
이것 역시 대결 성적을 랭킹을 매겨서 관리한다.
[향후 업데이트 기능임]

◆ A-1-3. 배경 / 분위기

- **A-1-3-1.** 기본적으로 중세 판타지를 컨셉으로 하는 배경을 가진다

- **A-1-3-2.** 등장하는 캐릭터들은 7 ~ 8등신의 인체비율을 가지는, 실사적인 느낌으로 제작한다.

- **A-1-3-3.** 숲, 사막, 얼음, 성채 등 지역에 따라 자연 배경이 극적으로 바뀐다.
: 이러한 자연 배경들은 플레이를 진행할 때, 막(Act)에 따라 구분한다.

◆ A-1-4. 컨텐츠 제한 사항

- **A-1-4-1.** 목표 심의 등급
  : 결정해서 넣을 것. 현재는 대한민국 기준으로 '18세 이상 가'가 목표...

- **A-1-4-2.** ~~어떠한 색깔로든 피를 직접적으로 표현하면 안 된다~~.
  피와 혈흔을 표현하는 데 있어 특별한 제약은 없다.

- **A-1-4-3.** ~~골절 및 신체 절단에 대한 표현이 들어가서는 안 된다~~.
  골절 및 신체 절단을 표현하는 데 대한 특별한 제한사항은 없다.

- **A-1-4-4.** 과도하게 선정적인 장면이나, 동물의 성기(사람을 포함해서)를 직접 표현하는 리소스가 들어가서는 안 된다.

- **A-1-4-5.** 광고는 로그인 직후와 게임을 종료할 때 확인하는 GUI 창의 배경에서 보여준다. 그 외 다른 장면에서는 광고를 노출하지 않는다.

※ 광고는 가장 직접적으로 사용자의 시야를 방해한다. 그러므로 노출 시점과 위치를 신중하게 선택해야 한다.
이에 대한 정확한 사항은 출시 마켓과 발행사의 정책에 따라 달라질 가능성이 있다.

- **A-1-4-6.** 처음 설치한 사용자가 무작정 튜토리얼부터 강제로 플레이 해야만 하는 상황을 만들지 말 것.
  : 튜토리얼을 플레이 할 것인지에 대해 선택지를 준다.

A-2. 재미 요소

◆ A-2-1. 액션 / 타격을 통한 재미

- **A-2-1-1.** 강력한 몬스터와의 전술적인 [1 vs 1] 상황을 여러 차례 가지기보다는, [1 vs 다수] 의 구도에서 화끈하게 쓸어버리는 듯한 전투의 느낌을 줘야 한다.

- **A-2-1-2.** 사용자가 '느끼기에' 모든 플레이 요소는 실시간으로 상호작용하고 있다고 생각하게 만들어야 한다.
  : 즉, 턴 방식, 또는 세미 – 턴 방식의 플레이 진행 요소는 넣지 않는다.

- **A-2-1-3.** 동시에 여러 대상을 타격할 수 있는 기능이 있어야 한다. (Non-Targeting, Multi-Targeting)
  : 공격 궤적, 또는 범위 대상 공격에 의해 물리적인 거리 계산으로, (단일 혹은 다수)목표 대상들이 명중했는지 여부를 결정하는 기능이 존재해야 한다.

◆ A-2-2. 아이템 수집과 강화

- **A-2-2-1.** 캐릭터가 장비할 수 있는 여러 종류의 아이템들이 존재한다.
  : 현재 8종의 장착 아이템으로 디자인하고 있다.

- **A-2-2-2.** 장비 아이템을 착용하면, 캐릭터의 (전투와 관련된) 능력이 상승한다.
  : 그래서 더 강력한 적을 더 쉽게 상대할 수 있게 된다.

- **A-2-2-3.** 각 아이템들의 능력은 완전히 조합적인 방식이다.
  : 외형(모델링이나 아이콘 텍스처.)이 같아도, 세부적인 능력 속성은 제각기 다를 수 있다.
  플레이어들은 좀 더 좋은 조합의 능력을 가진 아이템을 찾아서 계속해서 플레이를 하게 된다.

- **A-2-2-4.** 남들이 손쉽게 얻지 못하는 독특한 외형을 아이템에 입힐 수 있다.
  : 아이템의 능력을 전혀 건드리지 않고, 외형만 변경한다.
  또한, 아무나 획득하지 못하는 형상을 획득할 수 있게 해서, 플레이어 개인의 만족감 및 다른 플레이어에 대한 우월감을 자극한다.

◆ A-2-3. 스킬 강화

- **A-2-3-1.** 스킬은 기본적인 스킬 고유의 능력 외에도, 부가적인 능력 속성이 더 붙을 수 있다.

- **A-2-3-2.** 주 능력이 같은 스킬 일지라도, 추가적인 부가적인 능력 속성의 종류와 개수는 스킬 마다 다를 수 있다.

- **A-2-3-3.** 스킬도 아이템처럼 같은 종류 내에서도 등급 차가 있다.
  : 더 높은 등급의 스킬은 더 강력한 능력과 더 많은 부가적인 능력 속성들을 가진다.

◆ A-2-4. 사용자 경쟁 요소

- **A-2-4-1.** 사용자 캐릭터들끼리 대결을 하고, 그 순위를 매긴다.
  : 사용자들의 경쟁심을 유발한다.

- **A-2-4-2.** 높은 순위를 획득한 사용자 캐릭터는 그만큼의 보상을 받게 되고, 그러한 이득을 얻기 위해 경쟁적으로 순위를 올리기 위한 노력을 한다.
  : 아이템 강화, 더 좋은 아이템 뽑기, 더 상위의 컨텐츠를 플레이 할 것을 추구하는 등...

- **A-2-4-3.** 순위 요소의 종류는 컨텐츠의 성격마다 구분해서 산정하고, 그 보상 역시 서로 호환되지 않는 종류로 설정한다.
  : 모든 종류의 경쟁 요소들에 플레이어가 적극적으로 참여하게 해야 한다.

◆ A-2-5. 커뮤니티 협동 요소

- **A-2-5-1.** 게임 도중에 만난 다른 플레이어들과 친구 관계를 맺고, 상호간 지속적으로 활동을 모니터링할 수 있다.
  : 언제든 원하는 때에, 친구가 현재 레벨이 몇인지, 어떤 아이템과 스킬을 가지고 있는지 등을 볼 수 있다.

- **A-2-5-2.** 친구 혹은 무작위하게 만난 다른 플레이어들과 파티를 결성해서 더 어려운 스테이지를 더 쉽게 공략할 수 있다.

- **A-2-5-3.** 다른 플레이어와 협동해서 강력한 보스 몬스터를 공략하고, 그에 따른 희귀한 보상을 얻을 수 있다.

- **A-2-5-4.** 플레이어들은 자신이 성장시킨 캐릭터를 이용해서 다른 플레이어의 캐릭터와 대결을 할 수 있다. (PvP)
[향후 업데이트 기능임]

- **A-2-5-5.** 플레이어는 다른 플레이어들과 길드를 결성할 수 있다.

- **A-2-5-6.** 길드를 결성한 플레이어는 길드와 길드 간의 전쟁에 참여할 수 있다.
[향후 업데이트 기능임]

◆ A-2-6. 목적 달성

- **A-2-6-1.** 일정한 주기로 달성할 수 있는 작은 목표들을 계속해서 제시한다.
: 플레이어는 손쉽게 달성할 수 있는 주어진 작은 목표들을 연속해서 달성해나가다 보면, 어느새 플레이에 익숙해지고, 게임을 진행하는 패턴을 익히게 된다.

- **A-2-6-2.** 목적을 달성할 때마다 주어지는 보상을 모아서 앞으로 플레이를 더 쉽게 진행할 수 있다.
: 더 강력한 아이템을 획득하거나, 더 많은 소모품을 살 수 있거나, 과금 기능을 추가적으로 이용할 수 있게 되는 등...

◆ A-2-7. 희귀 요소 정복

- **A-2-7-1.** 보물 던전
: 무작위한 확률로 등장하는 보물 던전에 진입해서, 일반적인 스테이지에서보다 더 많은 골드를 획득한다.

- **A-2-7-2.** 월드 보스 공략전
: 모든 플레이어가 협동해서 하루에 한 번만 등장하는 가장 강력한 보스 몬스터를 공격한다.
보스 몬스터를 물리치면 공헌한 바에 따라 많은 보상을 획득할 수 있다.

◆ A-2-8. 플레이어가 연구할만한 요소 제공

- **A-2-8-1.** 아이템의 옵션 조합에 따라 플레이어 캐릭터가 더 강력한 능력을 소유할 수 있다.

- **A-2-8-2.** 액티브 스킬과 패시브 스킬의 조합에 따라, 같은 스테이지에 대해 플레이 체감 난이도가 달라질 수 있다.

- **A-2-8-3.** 스킬과 잘 연계될 수 있는 아이템과 스킬 간의 조합을 찾음으로써, 같은 스테이지에 대해 체감 난이도가 달라질 수 있다.

- **A-2-8-4.** 스킬과 스킬 룬의 조합에 따라 스킬의 사용 용도와 위력이 격변할 수 있다.
  : 그다지 위력적이지 않던 스킬이 스킬 룬과의 조합으로 주력 스킬로 변모하거나...

- **A-2-8-5.** 아이템과 보석의 조합에 따라, 아이템의 위력이 격변할 수 있다.
  : 무기에만 장착할 수 있는 발동 보석의 경우, 일반 공격 스킬에 추가적인 효과를 더하기 때문에, 더욱 그러할 수 있다.

A-3. 제품 사용 시나리오

◆ A-3-1. 처음 시작

- **A-3-1-1.** 처음 게임을 실행하면, 개발사(스노우프렌즈(주))와 발행사의 로고가 먼저 뜬다.
  : 로고는 애니메이션으로 동작할 수도 있다.

※ 로고 화면을 1단계로만 둘지, 여러 단계로 둘지, 어떤 순서대로 재생할 것인지 여부는 발행사의 연출 요구사항이나, 팀 내부의 합의를 통해 결정한다.

- **A-3-1-2.** 로고를 노출하는 단계에서 사용하는 리소스들은 반드시 응용 프로그램 패키지 내부에 내장되어 있어야 한다.
  : 실행파일의 용량 제한에 민감해야 한다면, 로고 노출 단계에서 사용할 리소스들의 용량에 대해 반드시 고려해야 한다.

※ 이 단계에서는 어떠한 형태의 외부 리소스 사용도 가정하고 있지 않다.
네트워크가 연결이 되지 않은 상태일지라도, 그 때문에 로고를 띄우는 데 필요한 리소스를 확보할 수 없어서 로고를 재생하지 못하는 현상이 벌어지면 안 된다는 말이다.

- **A-3-1-3.** 게임의 시작 동영상이 존재한다면, 로고가 재생된 직후에 자동으로 동영상을 재생할 수 있다. 없으면 곧바로 시작 장면으로 넘어가면 된다.
  : 동영상은 매번 재생하지 않아도 된다. 가장 처음 게임을 실행했을 때만 자동으로 재생하고, 그 다음부터는 곧장 시작 장면으로 넘어가게 해도 된다.

- **A-3-1-4.** 게임의 시작 전에 재생하는 동영상은 게임 실행 파일의 패키지 내부에 반드시 포함할 의무가 없다.
  : 따라서, 게임 소개 동영상을 재생하는 단계를 둘 생각이라면, 이 동영상이 없더라도 다음 단계로 진행하는 데 문제가 없게 만들어야 한다.

◆ A-3-2. 회원 가입 및 접속

- **A-3-2-1.** 플레이어가 시작 화면에 들어가면 ID와 패스워드를 입력하라는 GUI를 마주한다.
  : GUI 뒤의 배경에는 게임 제목과 함께 화려한 원화 그림, 혹은 고품질의 시작 화면 모델이 애

니메이션 되고 있다.

<로그인 화면의 예>

- **A-3-2-2.** 만약 처음 시작하는 플레이어라면, 회원 가입을 먼저 진행해야 한다.
 : ID와 패스워드를 입력하는 GUI 근처에 회원 가입으로 유도하는 버튼이 존재한다.

- **A-3-2-3.** 회원 가입 버튼을 터치하면, 약관에 동의하라는 화면이 등장한다.

- **A-3-2-4.** 약관 내용은 확인 버튼을 표시할 때 배경 GUI 내에 직접 보여줄 수도 있고, '상세보기' 링크로 대체할 수도 있다.
 : 이는, 약관 내용에 대한 데이터를 게임 패키지 내부에 반드시 내장할 필요는 없다는 의미이다.

※ 외부 URL 로 약관의 상세한 내용을 대체한다면, 약관이 변경되었을지라도 응용 프로그램 패키지를 다시 배포하지 않을 수 있다.

- **A-3-2-5.** 약관이 갱신되었을 때, 그 갱신 여부를 감지해서 다시 사용자에게 자동으로 약관을 확인하게 하는 기능이 있어야 한다.

- **A-3-2-6.** 사용자가 약관에 동의하면 회원 가입 화면으로 넘어간다.
 : 사용할 ID와 패스워드를 등록하면 회원 가입 절차가 끝난다.

※ 사용자의 회원 가입 및 정보 저장은 자체적으로 서비스를 구축할 수도 있고, 발행사 또는 외부 서비스 업체의 솔루션을 이용할 수도 있다.
 예를 들면, Kakao Talk, Line 등의 모바일 메신저와 연동하는 경우, NAVER, Facebook 등의 웹 서비스의 계정과 연동하는 것을 말한다.
 구체적으로 어떤 방식을 사용하기 될지는 사업적인 이슈에 의존한다.

- **A-3-2-7.** 사용자는 등록한(또는 이미 가지고 있는) ID와 패스워드를 이용해 로그인을 시도한다.

: 로그인에 성공해야만 다음 단계로 넘어가며, 로그인에 어떤 이유로든 실패하면, 실패한 이유를 대략적으로 알려주고 다시 시도하게 한다.

◆ A-3-3. 버전 확인 및 패치

- **A-3-3-1.** 로그인에 성공한 사용자들은 서버로부터, 현재 기기에 설치한 앱과 데이터에 대해 버전을 확인 받는다.
: 애플리케이션이 최신 버전이 아닌 경우, 최신 애플리케이션으로 새로 다운로드 받아서 설치하도록 링크를 제공한다.

- **A-3-3-2.** 추가 데이터의 버전이 최신이 아니거나, 혹은 추가 데이터 자체가 설치되지 않은 경우에는, 추가 데이터를 내려 받고 설치하는 장면으로 넘어간다.

- **A-3-3-3.** 추가 데이터의 설치 장면에서는 추가 데이터의 다운로드 및 설치 진행 상황이 어느 정도 되었는지를 대략적으로 짐작할 수 있는 GUI를 제공해야 한다.
: 대개, 진행 상황에 따라 늘어나는 프로그레스 바와 백분율 표시를 제공한다.

- **A-3-3-4.** 추가 데이터를 패치 하는 동안, 배경 화면에서는 게임을 진행하는 데 도움을 주는 도움말이 포함된 배경 그림들을 무작위로 몇 초씩 보여준다.

※ 플레이어가 덜 심심하도록 하는 효과도 있고, 실제로 게임 플레이를 더 잘하도록 유도할 수 있을 것이다.

- **A-3-3-5.** 게임 애플리케이션과 추가 데이터의 버전이 모두 맞으면, 사용자에게 최종 확인을 받는다.
: 'Game Start' 버튼이 깜박거리면서 사용자의 터치를 기다리는 GUI가 대표적이다.

◆ A-3-4. 계정 확인

- **A-3-4-1.** 사용자가 계정을 처음 만들고 최초로 접속하면, 사용자의 별명을 정해야 한다.
: **사용자의 별명을 정하지 않으면, 게임의 다음 진행 단계로 넘어갈 수 없다.**

- **A-3-4-2.** 사용자의 별명은 **특수 문자가 아닌 문자열과 아라비아 숫자**로 구성해야 하고, 띄어쓰기를 허용하지 않는다. 그리고 **첫 글자로는 숫자가 올 수 없다.**

- **A-3-4-3.** 사용자의 별명에 사용할 문자는 영어가 아닌 문자열도 허용한다.
: 따라서 반드시 유니코드로 입력 / 저장할 수 있어야 한다. (사실, GUI에 포함하는 문자열들이

다 그래야 한다.)

- **A-3-4-4.** 사용자의 별명은 **최대 16자리의 문자**로 구성할 수 있다.

※ 한글의 경우에는 이 정도로도 거의 만족하지만, 독일어처럼 단어의 길이가 몹시 긴 경우가 많을 때에도 괜찮은지 여부를 고려해야 한다.

- **A-3-4-5.** 사용자 계정의 별명은 서버군 내에서는 유일한 이름이어야 한다.

- **A-3-4-6.** 사용자 계정의 별명으로 게임 운영 원칙에 의해 제한되는 단어를 이름으로 사용할 수 없다.
: 욕설, 저속한 단어라든가, 게임 내 자산과 겹치는 이름, 기타 정책에 의한 제한 사항들을 포함한다.

◆ A-3-5. 캐릭터 생성

- **A-3-5-1.** 사용자가 계정을 처음으로 만들었다면, 게임을 플레이 할 캐릭터를 생성해야 한다.

<어떤 직업 클래스의 캐릭터를 선택할지 골라야 한다.>

- **A-3-5-2.** 사용자는 원하는 직업 클래스의 캐릭터를 선택하고, 확인 버튼을 누르면 사용자 자신이 플레이 할 캐릭터를 생성할 수 있다.

- **A-3-5-3.** 사용자가 생성한 각 캐릭터의 이름은 캐릭터마다 별도로 짓지 않고, **사용자의 별명으로 공유해서 사용**한다.

- **A-3-5-4.** 사용자가 플레이어 캐릭터를 선택할 때, 그 캐릭터의 초보 레벨 장비를 착용한 모습과 상위 레벨 장비를 착용한 모습을 모두 참고해볼 수 있다.

- **A-3-5-5.** 플레이어 캐릭터를 생성할 때, 기본 신체의 일부를 변경하는 기능, 즉 신체 커스터마

이징 기능은 제공하지 않는다.

※ 향후 업데이트 방향에 따라, 기본 신체의 일부분을 커스터마이징 할 수 있는 기능이 제공될 수는 있다. (가능성은 낮다.)
그러나, 이 때에도 신체의 길이를 바꾸거나 아예 다른 애니메이션을 필요로 하는 모양의 신체로 바꿀 수 있는 기능은 고려하지 않는다.
그렇다면 이런 제약 하에서 변경 가능한 요소는 얼굴이나 표정 정도 뿐인데, 모바일 기기에서 실행하는 이 게임의 특성에 비추어, 그다지 변화가 눈에 띌 만한 요소라고 생각하지 않는다.

- **A-3-5-6.** 캐릭터를 추가적으로 생성하기 위해서 게임 재화를 이용해 비용을 지불해야 할 수 있다.
: 추가적인 캐릭터 슬롯이거나, 혹은 최초 선택한 직업 외의 다른 직업 캐릭터를 생성할 때 비용을 적용할 수 있다.

◆ A-3-6. 캐릭터 선택

- **A-3-6-1.** 사용자의 계정은 최대 X명의 캐릭터를 소유할 수 있다.
: 몇 개까지의 캐릭터를 소유할 수 있을지 결정할 것

- **A-3-6-2.** 만들어진 캐릭터가 존재하면, 플레이어는 만들어진 캐릭터 중에서, 자신이 플레이 할 캐릭터를 골라야 한다.

- **A-3-6-3.** 하나의 사용자 계정은 동시에 여러 캐릭터를 게임 플레이에 데리고 갈 수 없다.
: 사용자 계정 당 한 번에 하나의 캐릭터만으로 게임 플레이를 할 수 있다.

- **A-3-6-4.** 캐릭터를 선택한 뒤, 시작 버튼을 터치하면 로비 장면으로 들어간다.

◆ A-3-7. 로비

- **A-3-7-1.** 로비 화면에서는 플레이어가 선택한 캐릭터의 3D 모델이 애니메이션 되고 있으며, 그 주변으로 로비 장면에서 이용할 수 있는 각종 기능들은 메뉴 형식으로 제공하는 GUI가 보인다.

- **A-3-7-2.** 로비 화면에서 보이는 캐릭터의 3D 모델 외형은, 그 캐릭터가 착용한 장비 아이템들과, 현재 부여된 상태효과들의 이펙트를 게임 플레이 장면에서 실제로 게임을 진행할 때의 모델과 전혀 다름이 없이 표현해야 한다.

※ 위 내용은 각 직업 클래스다가 별도로 제작하는 '대표 모델 데이터'를 사용하는 방식을 쓰지 않겠다는 뜻이기도 하다.
로비 화면에서 별도의 고품질 리소스 데이터를 이용한 3D 모델을 사용하려면, 모든 장비 아이템의 파츠들마다 고품질 용 모델을 별도로 두는 건 현실적으로 어렵기 때문이다.
아무래도 데이터 사용량에 따른 네트워크 요금이 붙는 모바일 기기의 특성상, 용량 문제에 민감할 수 밖에 없다.

- **A-3-7-3.** 로비에서 현재 게임 플레이를 위해 준비한 캐릭터의 관리 기능을 수행한다.
: 아이템을 교체하거나, 새로 구입하거나, 미션 / 퀘스트를 수락하거나 하는 행위 등...

◆ A-3-8. 전투 입장

- **A-3-8-1.** 로비에서 전투 모드를 선택한 뒤에, 해당하는 전투 모드의 게임 플레이 장면으로 진입할 수 있다.

※ 사용자가 이용할 수 있는 기능들을 유사한 집합끼리 묶어서 별도의 장면으로 제공하고, 로비 장면에서는 그러한 기능 집합 장면으로 넘어갈 수 있는 버튼과 아이콘을 제공한다.
: 싱글 플레이, PvP, 보스전, 상점, 친구 관리, 길드 관리 등의 메뉴가 전부 로비 장면에 널려 있으면 화면도 복잡하고, 알아보거나 이용하기도 쉽지 않을 것이다.

- **A-3-8-2.** 사용자 계정 혹은 플레이어 캐릭터의 조건에 따라 이용할 수 없는 모드들이 존재할 수 있다.
: 플레이어 캐릭터의 레벨이 부족하다든가, 기타 조건에 의해서 그러할 수 있다.

- **A-3-8-3.** 현재 이용할 수 없는 모드들은 흑백 색상으로 보이거나 혹은 아주 어두운 음영으로 가려진다. 그 상태에서 터치하는 경우, 왜 이용할 수 없는지에 대한 이유를 팝업 윈도우 또는 텍스트 레이블로 알린다.

◆ A-3-9. 스토리 모드

- **A-3-9-1.** 전투 입장 화면에서 스토리 모드를 선택하면 플레이 할 수 있는 모드이다.

- **A-3-9-2.** 정해진 순서대로 차례차례, 현재 스테이지를 돌파하면 다음 스테이지에 진입할 수 있는 방식이 반복된다.

- **A-3-9-3.** 스테이지는 최소한 수십 개 이상이며, 여러 개의 스테이지가 모여서 하나의 막(Act)를 구성한다.

- **A-3-9-4.** 막(Act)이 바뀌면, 전반적인 지형의 분위기, 등장하는 몬스터의 종류도 달라진다.
 : 초원지대, 사막, 얼음지대, 용암, 지하수로 등의 전체적인 분위기를 말한다.

- **A-3-9-5.** 스테이지 진행 동기는 미션 / 퀘스트에 의한 것이지만, 이는 단순한 알림 메시지 정도의 역할일 뿐, **내용은 대개 해당 스테이지의 몬스터들을 전멸시키는 형식**이다.

- **A-3-9-6.** 스테이지 중 일부는 강력한 보스 몬스터가 등장하는 특별한 스테이지이다.
 : 일반적으로는 막의 가장 마지막 스테이지가 그러하다.

- **A-3-9-7.** 일반 스테이지에서도 막바지에는 일반 몬스터보다 훨씬 강력한 보스 몬스터가 존재할 수 있다.
 : 타입에 대한 명칭 분류는 명확히 해야 할 것이다. '중간 보스 몬스터'로 하든지, 아니면 '정예 몬스터'로 하든지...

- **A-3-9-8.** 스테이지에 진입하기 위해서는 **입장 자원이 필요**하다.
 : 보통 이런 입장 자원은 시간이 지나면 하나씩 채워진다.
 물론, 지금 당장 플레이 하기를 원한다면 과금 화폐를 지불하고 사는 방법도 있다.

- **A-3-9-9.** 스테이지에 진입하기 전에, 플레이에 도움을 주는 부여효과(Enchanted Effect)를 구매할 수 있다.
 : 이 부여효과들은 해당 스테이지를 플레이 할 때만 유효하며, 그 스테이지를 빠져나오면(공략 포완료, 혹은 포기) 효과가 사라진다.

※ 대부분 플레이어 캐릭터의 전투 능력을 향상시켜주는 부여효과들로 구성되어 있다.
구체적인 내용은 관련 기획서, 혹은 명세의 부여효과 관련 항목들을 참고한다.

- **A-3-9-10.** 스토리 모드에서는 자신의 친구로 등록한 플레이어 캐릭터를 소환해서 잠시 동안 같이 플레이 할 수 있다.
 : 친구 캐릭터가 등장해서 인공지능에 의해 플레이어 자신의 캐릭터와 같이 돌아다니면서 몬스터와 전투를 할 수 있다.

- **A-3-9-11.** 스토리 모드에서 친구 소환을 하기 위해서는, 게임에 진입하기 전에 어떤 친구 캐릭터를 소환할 것인지 먼저 결정하고 들어가야 한다.
 : 친구 소환을 하지 않고 플레이 하기로 결정할 수도 있다. 이런 경우, 친구 소환 버튼은 사용불가 상태가 된다.

- **A-3-9-12.** 소환된 친구 캐릭터는 일반적으로 제한된 시간만큼 내 캐릭터를 도와주고 사라진다.

하지만, 원한다면 일정 비용을 지불하고(아마도 과금 화폐), 친구 캐릭터를 다시 소환해서 전투를 진행하는 것도 가능하다.

- **A-3-9-13.** **스테이지에 입장할 때마다 입장 자원을 소모**한다.
: 기본적으로 스테이지의 성패와 관계없이 입장 자원을 소모한다.

- **A-3-9-14.** 스테이지를 진행하다가 실패하더라도 입장 자원은 일부만 '환불'받을 수 있다.
: 스토리 모드에서는 입장할 때 입장 자원을 5 소모하고, 스테이지 돌파에 실패하면 1을 반환해준다.

- **A-3-9-15.** 일반 난이도의 스토리 모드를 진행할 때는, 입장 자원 제한만 아니라면, 자신이 지금까지 완료한 스테이지들 중에서 어떤 스테이지를 몇 회 반복해서 플레이하든 제한이 없다.

◆ A-3-10. 스토리 모드(P vs E) – 정예 난이도

- **A-3-10-1.** 스토리 모드의 일반 난이도와 같은 방식의 스테이지 진행을 가지지만, 내부의 몬스터 배치와 수량 등은 일반 난이도의 같은 스테이지 구간에 비해 훨씬 어렵게 디자인 한다.

- **A-3-10-2.** 정예 난이도의 경우, **하루에 같은 스테이지를 3회까지만 완료할 수 있다.**
: 입장 자원이 충분하다고 하더라도, 같은 스테이지를 3회 완료했으면, 다음 날이 되기 전까지는 더 이상 그 스테이지(정예 난이도)는 플레이 할 수 없다.

- **A-3-10-3.** 스토리 모드의 정예 난이도는 일반 난이도보다 입장 자원을 2배 소모한다.

- **A-3-10-4.** VIP 레벨이 향상되면 수행한 횟수를 초기화할 수 있는 권한을 얻을 수 있다.

- **A-3-10-5.** 같은 스테이지에 대해 수행 횟수를 초기화를 하는 횟수에 대해 특별히 제한사항은 없다.
: 하나의 정예 난이도 스테이지에 대해서 계속해서 수행 횟수 초기화를 해도 된다. (수행 횟수를 초기화할 수 있는 권한의 수량이 가능한 한도 내에서 말이다.)

- **A-3-10-6.** 스토리 모드의 보통 난이도와 정예 난이도는 진행 과정을 독립적으로 구성한다.
: 즉, 보통 난이도와 정예 난이도는 별도로 구분되어 있는 UI의 페이지에서 각각 진행한다.
: 이는 후술할 몇 가지 사양들이 성립하기 위한 중요한 기본 전제이다.

- **A-3-10-7.** 스토리 모드의 정예 난이도의 각 스테이지는 등장 몬스터와 지형 배경의 컨셉은 보통 난이도와 공유한다.

- **A-3-10-8.** 보통 난이도의 같은 스테이지 번호와 비교해서, 몬스터의 체감 난이도, 배치, 하나의 스테이지를 구성하는 부분 영역의 개수, 지형 모양 등은 다를 수 있다.

※ 기술적인 내부 구현 사양에서는, 사실 어떤 스테이지의 보통 난이도와 정예 난이도 구분은 별다른 의미가 없다.
그러니까, 사실상 두 가지 난이도 종류의 스테이지는 기술적으로는 아예 별개의 스테이지라고 보면 되고, 보통 난이도다, 정예 난이도다 하는 구분은 그냥 사용자에게 컨텐츠를 제공하는 입장에서의 구분점으로 보면 된다.
즉, 스테이지의 난이도 별 구분은 등장할 컨텐츠의 외형적인 컨셉을 제외하고는 스테이지 난이도 간에는 직접적인 연관성이 없다.
이런 방식이 성립하기 위해서는, 각 스테이지를 선택한 뒤, 그 스테이지의 난이도를 고르는 게 아니라, 난이도를 선택한 뒤, 스테이지를 선택하는 방식이어야 한다. 그렇지 않다면 스테이지와 난이도 분류가 연관이 있어야 하기 때문에 위와 같은 방식의 구현에도 제약이 생긴다.

- **A-3-10-9.** 특정 정예 난이도의 막(Act)를 진행하려면, 먼저 그 막의 보통 난이도를 마지막 스테이지까지 완료해야 한다.
: 1막의 스테이지가 10개까지 있다면, 1막 스테이지 10까지 완료해야 1막의 정예 난이도를 시작할 수 있다.

◆ A-3-11. 일일 던전(P vs E)

- A-3-11-1. 보석과 재료를 획득할 수 있는 스테이지들이 종류별로 나열되어 있고, 그 중에 플레이어가 원하는 보석 혹은 재료를 얻을 수 있는 스테이지를 골라서 도전하는 방식이다.

- A-3-11-2. 하루에 총 5회만 수행할 수 있다.
: 즉, 자신이 원하는 보석과 재료를 주는 스테이지를 골라서 가는 방식이다.

- A-3-11-3. 수행할 때마다 입장 자원을 소모한다.
: 만약 입장료가 부족하다면, 입장료를 추가로 구매할 것을 권유한다.

- A-3-11-4. 플레이어는 일일 던전을 수행할 때, 난이도를 고를 수 있다.
: 더 높은 난이도의 일일 던전은 더 많은 보상을 주지만, 완료하기가 더 어렵다.

- A-3-11-5. VIP 레벨이 향상되면 도전 횟수를 늘릴 수 있다.

- A-3-11-6. 여러 개의 고정적인 난이도를 두고, 플레이어가 원하는 난이도의 던전 스테이지를 골라서 도전하는 방식이다.
: 현재 총 6단계의 난이도로 구성한다.

~~- A-3-11-7. 각 난이도 별 스테이지는 처음부터 모든 난이도를 자유롭게 드나들 수는 없고, 수행하는 캐릭터의 레벨에 따라 순차적으로 더 높은 난이도에 도전할 수 있는 자격을 준다~~

~~- A-3-11-8. 보상~~
~~: 도전한 스테이지에서 주도록 되어 있는 재료와 보석을 받는다.~~

※ 기획에서 일일 던전과 요일 던전을 주간 던전으로 통합하기로 변경하였다.

◆ A-3-12. 주간 던전(P vs E)

- **A-3-12-1.** 주간의 특정한 요일에만 열리는 던전이다.

- **A-3-12-2.** 던전의 종류
: 홀수 요일에는 회복약을 얻을 수 있는 던전 스테이지, 짝수 요일에는 골드로 변환할 수 있는 아이템(골드 벌이), 일요일에는 둘 다 열린다.

요일	보상	설명
월, 수, 금	회복약	보상으로 회복약을 얻을 수 있는 스테이지
화, 목, 토	골드 변환 아이템	보상으로, 판매하면 많은 골드를 주는 특수한 아이템을 얻을 수 있는 스테이지
일요일	회복약, 골드 변환 아이템	두 가지 스테이지가 모두 열린다.

- **A-3-12-3.** 각 던전의 종류마다 수행할 수 있는 횟수가 3회로 제한되어 있다.
: 단, 그렇기 때문에 두 가지 던전이 다 열리는 일요일에는 각기 3회씩 총 6회를 플레이할 수 있다. (물론, 던전의 종류마다 제한하는 횟수는 동일하므로 각 던전마다 3회씩 플레이해야 한다.)

- **A-3-12-4.** 수행할 때마다 입장 자원을 소모한다.
: 만약 입장료가 부족하다면, 입장료를 추가로 구매할 것을 권유한다.

- **A-3-12-5.** VIP 레벨이 향상되면 하루 도전 횟수를 늘릴 수 있다.

- **A-3-12-6.** 여러 개의 고정적인 난이도를 두고, 플레이어가 원하는 난이도의 던전 스테이지를 골라서 도전하는 방식이다.
: 현재 총 6단계의 난이도로 구성한다.

- **A-3-12-7.** 각 난이도 별 스테이지는 처음부터 모든 난이도를 자유롭게 드나들 수는 없고, 수행하는 캐릭터의 레벨에 따라 순차적으로 더 높은 난이도에 도전할 수 있는 자격을 준다.

◆ A-3-13. 혼돈 던전(P vs E)

- **A-3-13-1.** 혼돈 던전은, **스토리 모드에서 사용했던 모든 스테이지의 지형 맵과 몬스터의 종류들을 뒤섞어서 무작위로 구성**하는 던전이다.

- **A-3-13-2.** 혼돈 던전 모드의 구성은 다음과 같다.

1. 혼돈 던전 입장을 하면, 전체 스테이지의 지형 맵 중에서 무작위하게 하나의 지형 맵이 선택된다.
2. 등장하는 몬스터도 무작위하게 선택된다. 지형 맵이 어떤 분위기인지는 고려하지 않는다.
: 지형 맵은 용암 던전인데, 등장하는 몬스터는 얼음거인일 수도 있다.
3. 몬스터들의 배치 방식 역시 무작위하다.
4. 혼돈 던전의 가장 안쪽에는 반드시 보스 몬스터가 존재한다.
5. 특별히 혼돈 던전 스테이지를 위한 퀘스트는 없다. 그저 스테이지의 보스 몬스터를 처치하면 그 스테이지는 승리했다고 판단한다.

※ Diablo Ⅲ의 '균열 모드'와 매우 비슷한 개념이다.

- **A-3-13-3.** 스테이지를 구성하는 지형 맵과 몬스터의 종류에 따른 난이도 격차는 존재하지 않는다.
: 즉, 스토리 모드에서 저 레벨 구간이었던 지형 맵이나 몬스터가 등장한다고 해서 난이도가 낮아지거나, 고 레벨 구간이었던 지형 맵이나 몬스터가 등장한다고 해서 난이도가 더 높아지지 않는다.

- **A-3-13-4.** 스테이지의 마지막 위치에는 보스 몬스터가 존재하고, 이 보스 몬스터를 물리치면 그 스테이지를 완료한 것으로 판정한다.
: 혼돈 던전 스테이지에서 무작위 하지 않은 요소 중 하나는, **어떤 종류이든 항상 마지막 지점에 보스 몬스터가 있다**는 점이다.

※ 몬스터를 구성할 때, 몬스터의 외형과 몬스터의 능력을 주는 부분을 별도로 구성할 수 있도록 설계해야 이 요구사항을 만족할 수 있다.
같은 외형을 가진 몬스터 일지라도, 능력은 1레벨 캐릭터가 상대할 수 있는 수준부터 최고 레벨의 캐릭터도 쉽게 상대할 수 없는 수준까지 다양하게 설정할 수 있어야 하기 때문이다.

- **A-3-13-5.** 핵심적인 재미 요소는, 들어갈 때마다 변하는 지형 맵, 몬스터들의 공격 방식, 몬스터 종류 별 조합과 그에 따른 전술적인 시너지들로 인해 체감 난이도와 플레이 방식이 다양하게 느껴진다는 점이다.

※ 같은 난이도의 혼돈 던전일지라도, 내 캐릭터의 직업과, 게임 플레이에 가지고 들어간 액티브 스킬의 종류, 설정한 패시브 스킬의 종류가 등장한 몬스터들의 공격 / 방어 / 스킬 / 행동 유형과 '궁합이 잘 맞으면' 체감 난이도가 더 쉽게 느껴질 수 있을 것이다. 만일 그렇지 않다면 같은 난이도에서도 체감 난이도가 더 높을 것이다.
근접하느라 뭉치는 몬스터가 많이 등장하는 스테이지에서는 사거리가 짧아도 광역 공격을 하는 스킬을 가지고 있는 캐릭터가 유리할 것이고, 원거리 공격을 많이 하며 흩어지는 성향이 강한 몬스터가 많이 등장하는 경우에는 사정거리가 길고 한 발 한 발의 위력이 강한 스킬을 보유한 캐릭터가 더 손쉽게 진행할 수 있을 것이다.

- **A-3-13-6.** 수행할 때마다 입장 자원을 소모한다.
: 만약 입장료가 부족하다면, 입장료를 추가로 구매할 것을 권유한다.

- **A-3-13-7.** VIP 레벨이 향상되면 하루 도전 횟수를 늘릴 수 있다.

- **A-3-13-8.** 난이도는 몇 가지 고정적인 종류의 난이도가 존재하고, 플레이어는 이 난이도 중에서 선택해서 도전하는 방식이다.
: 최고 레벨을 달성하지 못한 플레이어도 즐길 수 있는 낮은 난이도부터, 캐릭터의 최고 레벨은 기본이고, 최고의 아이템을 보유하고 사냥 기술을 연마한 플레이어들이 도전해야 하는 난이도까지 존재한다.

- **A-3-13-9.** 각 난이도 별 스테이지는 처음부터 모든 난이도를 자유롭게 드나들 수는 없고, 수행하는 캐릭터의 레벨에 따라 순차적으로 더 높은 난이도에 도전할 수 있는 자격을 준다.

- **A-3-13-10.** 주요한 보상

보상	설명
악의 정수	전설 아이템 제작 재료
골드	언제나 골드는 필요하다.
아이템 강화석	아이템의 강화 점수를 높인다.
아이템 초월석	아이템의 강화 최대 한계를 늘린다.

◆ A-3-14. 초월 던전(P vs E)

- **A-3-14-1.** **첫 출시에는 반영하지 않으며, 업데이트로 추가한다.**

- **A-3-14-2.** 일종의 디펜스 모드이다.

- **A-3-14-3.** 여러 단계의 제파(Wave)식으로 밀려오는 몬스터들을 막아내는 방식으로 진행한다.

- **A-3-14-4.** 등장하는 몬스터들의 종류와 조합 방식은 완전히 무작위하다.
: 혼돈 던전과 마찬가지로, 등장하는 몬스터의 종류와 조합, 그리고 플레이하는 캐릭터의 직업 클래스 및 스킬 셋에 따라 체감 난이도가 다를 수 있다.

- **A-3-14-5.** 초월 던전의 난이도에는 특별한 제한이 없다.
: 진행을 계속할수록 등장하는 몬스터들은 무한정 강해질 수 있다.

- **A-3-14-6.** 초월 던전은 하나의 던전 내에서 3가지 지역을 옮겨 다니며 진행하는 방식이다.
: 각 지역들은 몬스터의 소환 방식과 모양의 컨셉을 대조적으로 구성한다. 지역을 옮겨 다니는 방식은 완전히 무작위로 결정된다.

※ 지형의 모양이 길쭉하고 좁거나, 혹은 광활하고 몬스터가 사방에서 소환되는 등 컨셉이 매우 다르기 때문에, 어떤 지형의 구역에서 어떤 몬스터 조합이 등장하느냐에 따라 해당 몬스터 웨이브가 쉬울 수도 있고, 어려울 수도 있다.

- **A-3-14-7.** 하루에 3회까지만 플레이 할 수 있다.

- **A-3-14-8.** 일일 횟수 제한을 제외하고, 던전에 입장할 때마다 입장 자원을 소모하지는 않는다.

- **A-3-14-9.** VIP 레벨이 향상되면 일일 도전 횟수를 늘릴 수 있다.

- **A-3-14-10.** 초월 모드에서의 성적을 바탕으로 각 캐릭터들의 랭킹을 산정한다.

- **A-3-14-11.** 초월 모드를 진행한 성과를 바탕으로 초월 코인을 획득할 수 있다.
: 획득한 초월 코인은 초월 모드의 전용 상점에서 이용할 수 있다.

- **A-3-14-12.** 초월 모드의 전용 상점에서는 초월 코인만을 이용해서 살 수 있는 희귀한 상품들을 판매한다.
: 구매한 품목은 품절 상태가 된다. 이 때, 판매 목록을 갱신하고 싶으면, 일정량의 초월 코인을 요구한다. (판매 목록의 갱신 횟수는 제한되어 있고, 갱신이 반복될수록 요구하는 초월 코인의 비용도 높아진다.)

◆ A-3-15. 보물 던전(보너스 스테이지)

- **A-3-15-1.** 일반 스테이지보다 많은 골드를 벌어들일 수 있는 스테이지

- **A-3-15-2.** 가끔 등장하는, 보너스 스테이지의 개념이다.

- **A-3-15-3.** 다른 아무 스테이지의 결과 보상이 끝나고 난 뒤, 무작위 판정에 의해 등장한다.
: 판정 그 자체는 클라이언트에서 수행하지 않으며, 서버에서 판정한 결과를 클라이언트가 수용해서 등장하게 한다.

- **A-3-15-4.** 보너스 스테이지의 등장 여부는 각 클라이언트마다 개별적으로 작동한다. 따라서, 클라이언트가 접속 상태여야만 보너스 스테이지 등장 여부를 판정할 수 있다.

※ 이 점은 월드 보스 공략전과 중요한 차이점이다.
월드 보스 공략전은 전역적으로 모든 플레이어들이 실제 시간으로 같은 시간 안에 공략해야 하는 컨텐츠이기 때문에, 서버 상에서 시간을 점검하며, 클라이언트 접속 여부와 관계 없이 등장 여부를 결정한다. 심지어, 클라이언트가 접속하지 않은 상태라면, Push 메시지를 통해 알려주기도 한다.
그러나 보너스 스테이지의 등장 확률은 철저히 클라이언트가 접속해 있는 상태에서, 클라이언트가 현재 스테이지를 완료했음을 서버에게 알리는 그 순간에 판정이 이루어진다. 즉, 클라이언트가 접속하지 않은 상태에서는 이런 판정을 할 기회 자체가 없으므로, 당연히 그 클라이언트에 대해서는 보너스 스테이지도 작동하지 않는다.

- **A-3-15-5.** 보너스 스테이지가 등장하더라도, 플레이어는 보너스 스테이지를 플레이 할지 여부를 선택할 수 있다.
: 만약 플레이 하지 않는다고 선택하면, 보너스 스테이지 입장은 취소되고, 그냥 로비 장면으로 되돌아간다.

- **A-3-15-6.** 수행할 때마다 입장 자원을 소모한다.
: 보너스 스테이지이기는 하지만, 입장료 자체는 받는다.
만약 입장료가 부족하다면, 입장료를 추가로 구매할 것을 권유한다.

◆ A-3-16. 스테이지의 일회성 부여효과

- **A-3-16-1.** 스테이지를 들어갈 때, 그 스테이지를 손쉽게 공략할 수 있게 도움을 주는 부여효과들을 말한다.

- **A-3-16-2.** 플레이어가 스테이지를 선택하고 입장하기 전에, 그 스테이지에서만 사용할 일회성 부여효과들을 구매할지 선택할 수 있는 메뉴가 나온다.

- **A-3-16-3.** 일단 스테이지에 진입하면, 부여효과들은 그 스테이지를 플레이 하는 동안에만 유효하다.

- **A-3-16-4.** 스테이지를 어떤 이유로든 빠져나오면, 부여효과는 사라진다. 새로운 부여효과를 얻으려면, 다시 스테이지 일회성 부여효과를 구입해야 한다.

- **A-3-16-5.** 스테이지의 일회성 부여효과들은 각 종류당 한 번에 1회씩 구매할 수 있다.
: 예를 들어, 공격력 부여효과와 방어력 부여효과, 생명력 부여효과, 마력 부여효과가 있다고 치면, 이들을 전부 구매해서 4가지 부여효과들을 모두 장착하고 스테이지를 플레이 할 수도 있다. (물론, 그만큼 비용이 들어간다.)

- **A-3-16-6.** 같은 일회성 부여효과들을 여러 번 중복 구매해서 스테이지에 입장할 수 없다.

- **A-3-16-7.** 각 스테이지의 일회성 부여효과들은 무료로 사용할 수 있는 횟수가 주어질 수 있다.
: 대개 운영 이벤트 단계에서 보상의 일환으로 주게 될 것이다.

- **A-3-16-8.** 스테이지의 일회성 부여효과들을 사용하려고 할 때, 그 종류의 부여효과를 무료로 사용할 수 있는 횟수가 1회 이상이라면, 무료 사용 횟수를 먼저 소진한다.
: 무료 사용 횟수가 모두 소진되면, 그 때부터는 게임 화폐 등의 부여효과의 사용 비용을 차감하기 시작한다.

◆ A-3-17. 골드 채굴(SNG)

- **A-3-17-1.** 게임을 하고 있지 않은 상태에서도 지속적으로 (소소하게) 골드를 벌어들일 수 있는 컨텐츠이다.

- **A-3-17-2.** 실제 시간으로 골드가 얼마간 수집된다.
: 이는 플레이어의 접속 유무, 게임 플레이 유무에 관계없이 실제 시간으로 항상 일정하다.

- **A-3-17-3.** 로비 장면에서 플레이어는 골드 채굴 페이지를 열고, 수집 시간 및 단위를 골라서 골드 채굴을 시작할 수 있다.
: 최대 얼마간의 시간 동안, 얼마의 골드를 수집할지 선택해야 한다.

- **A-3-17-4.** 선택한 수집 시간을 초과하면, 더 이상 골드는 쌓이지 않는다.
: 추가적으로 수집을 하려면 해당 GUI를 방문해서, 수집되어 있는 골드를 획득하고, 다시 채굴 활동에 재배치해야 한다.

- **A-3-17-5.** 자주 방문해야 할수록 효율이 좋다.
: 1시간에 1번 방문하는 메뉴로 10회 돌리는 경우와, 10시간에 1번 방문하는 경우, 전자의 효율이 더 높게 설계한다.

- **A-3-17-6.** 채굴한 게임 플레이 화폐(골드)는 플레이어가 결과를 확인한 시점에 적용한다.
: 실시간으로 채굴한 게임 플레이 화폐를 합산해주지 않는다.

◆ A-3-18. 장비 아이템 연마(SNG)

- **A-3-18-1.** 게임을 하고 있지 않은 상태에서도 지속적으로 지정한 장비 아이템을 강화할 수 있는 컨텐츠이다.

- **A-3-18-2.** 로비 장면에서, 플레이어는 장비 아이템 연마 페이지를 열고, 연마할 장비 아이템을 아이템 보관함 내에서 선택한다.
: 연마할 장비 아이템은 캐릭터가 장착한 상태와 관계 없이 등록할 수 있다.

- **A-3-18-3.** 실제 시간으로, 등록한 아이템의 강화 점수가 얼마간 수집된다.
: 이는 플레이어의 접속 유무, 게임 플레이 유무에 관계없이 실제 시간으로 항상 일정하다.

- **A-3-18-4.** 플레이어는 장비 아이템을 연마할 시간 단위를 골라야 한다.
: 최대 얼마간의 시간 동안, 얼마의 강화 점수를 수집할지 선택해야 한다.

- **A-3-18-5.** 선택한 수집 시간을 초과하면, 더 이상 강화 점수는 쌓이지 않는다.
: 추가적으로 수집을 하려면 해당 GUI를 방문해서, 수집되어 있는 강화 점수를 획득하고, 다시 용병 활동에 재배치해야 한다.

- **A-3-18-6.** 자주 방문해야 할수록 효율이 좋다.
: 1시간에 1번 방문하는 메뉴로 10회 돌리는 경우와, 10시간에 1번 방문하는 경우, 전자의 효율이 더 높게 설계한다.

- **A-3-18-7.** 장비 아이템을 연마한 결과는 플레이어가 결과를 확인한 시점에 적용한다.
: 실시간으로 연마 결과를 반영해주지 않는다.

◆ A-3-19. 스킬 수련(SNG)

- **A-3-19-1.** 게임을 하고 있지 않은 상태에서도 지속적으로 지정한 스킬을 숙련하는 컨텐츠이다.

- **A-3-19-2.** 로비 장면에서, 플레이어는 스킬 수련 페이지를 열고, 수련할 스킬을 스킬 보관함 내에서 선택한다.
: 수련할 스킬은 캐릭터가 게임 플레이에 사용하기 위해 등록한 상태와 관계 없이, 수련을 위해

등록할 수 있다.

- **A-3-19-3.** 실제 시간으로 지정한 스킬의 숙련 점수가 얼마간 수집된다.
: 이는 플레이어의 접속 유무, 게임 플레이 유무에 관계없이 실제 시간으로 항상 일정하다.

- **A-3-19-4.** 플레이어는 수련할 시간 단위를 골라야 한다.
: 최대 얼마간의 시간 동안, 얼마의 숙련 점수를 수집할지 선택해야 한다.

- **A-3-19-5.** 선택한 수집 시간을 초과하면, 더 이상 숙련 점수는 쌓이지 않는다.
: 추가적으로 수집을 하려면 해당 GUI를 방문해서, 수집되어 있는 숙련 점수를 획득하고, 다시 스킬 수련에 메뉴에 재배치해야 한다.

- **A-3-19-6.** 자주 방문해야 할수록 효율이 좋다.
: 1시간에 1번 방문하는 메뉴로 10회 돌리는 경우와, 10시간에 1번 방문하는 경우, 전자의 효율이 더 높게 설계한다.

- **A-3-19-7.** 스킬을 수련한 결과는 플레이어가 결과를 확인한 시점에 적용한다.
: 실시간으로 수련 결과를 반영해주지 않는다.

◆ A-3-20. 스킬 획득과 장착

- **A-3-20-1.** 각 스킬마다 획득할 수 있는 최소 레벨이 정해져 있다.
: 처음 시작하는 플레이어는 점차 하나씩 새로운 스킬을 배우면서 그 캐릭터의 플레이에 익숙해지게 하는 컨셉이다.

- **A-3-20-2.** 각 스킬의 1레벨은 비용 없이 획득할 수 있다.

- **A-3-20-3.** 각 스킬 종류마다, 최대 숙련 레벨은 캐릭터의 성장 레벨과 같다.
: 캐릭터를 50레벨까지 성장할 수 있게 되어 있다면, 스킬도 50레벨까지 성장할 수 있다.

- **A-3-20-4.** 스킬에는 액티브 스킬과 패시브 스킬이 있다.
: 액티브 스킬은 게임 플레이에서 플레이어가 직접 선택해서 사용할 수 있는 스킬이고, 패시브 스킬은 게임 플레이에서, 플레이하는 캐릭터에게 기본적으로 뭔가 능력을 부여하는 스킬이다.

- **A-3-20-5.** 각 캐릭터마다 게임 플레이 스테이지에 들고 들어갈 수 있는 액티브 스킬과 패시브 스킬의 개수에는 최대 제한이 있다.

- **A-3-20-6.** 캐릭터마다 사용 가능한 액티브 스킬과 패시브 스킬의 개수는. 게임 플레이 스테이

지에 들고 들어갈 수 있는 최대 제한 개수보다 더 많을 수 있다.

- **A-3-20-7.** 로비 화면에서, 플레이어는 스킬 보관함 페이지를 열고, 자신이 플레이하는 캐릭터가 게임 플레이를 할 때 들고 들어갈 액티브 스킬과 패시브 스킬을 지정할 수 있다.
: 제한된 종류의 스킬만으로 플레이 해야 하기 때문에, 어떤 종류의 스킬들을 어떤 조합으로 구성하는지가 중요하다.

◆ A-3-21. 아이템 강화

- **A-3-21-1.** 각 아이템은 강화 점수로 불리는 점수 체계가 존재하고, 이 점수가 일정 이상 높아지면 아이템의 강화 레벨이 오른다.
: 캐릭터의 레벨과 경험치의 관계와 같다.

- **A-3-21-2.** 아이템을 강화하기 위해서, 플레이어는 강화할 장비 아이템에 다른 장비 아이템들을 재료로써 사용한다.
: 강화 재료로 등록한 각 장비 아이템들마다, 강화 재료로 '분해'하기 위한 비용을 게임 플레이 화폐로 내야 한다.

※ 즉, 강화 재료로 등록한 장비 아이템이 많을수록, 지불해야 하는 게임 플레이 화폐도 더 많아지게 된다.

- **A-3-21-3.** 각 아이템이 강화 재료로 들어갈 때 지불해야 하는 비용은, 아이템의 가치가 높을수록 더 커진다.
: 아이템의 보물 레벨과 등급에 따라 결정한다. 더 높은 보물 레벨과 등급을 가진 아이템들은, 아이템 강화를 위한 재료로 사용할 때, 더 많은 게임 플레이 화폐를 지불해야 한다.

- **A-3-21-4.** 아이템 강화는 로비 장면에서만 이용할 수 있다.
: 아이템 강화를 위한 UI는 로비 장면에서만 제공한다.

- **A-3-21-5.** 재료로 사용하는 장비 아이템들은 각자 일정한 강화 점수로 환원하여, 강화할 대상 아이템의 강화 점수에 더한다.

- **A-3-21-6.** 장착 가능한 레벨 대역이 높거나(즉, 보물 레벨 Treasure Level이 높거나), 등급이 높은 아이템일수록, 재료로 사용할 때 환원해주는 강화 점수가 많다.
: 희귀하고 값비싼 아이템을 재료로 쓸수록, 더 빨리 대상 아이템을 강화할 수 있다. (물론 어떤 때는 손해가 막심할 것이다.)

- **A-3-21-7.** 기본 아이템의 경우, 장착 가능한 레벨 대역이 높을수록 각 강화 레벨당 요구하는

강화 점수가 높다.

- **A-3-21-8.** 또한, 아이템의 등급(일반 < 마법 < 희귀 < 전설)이 높아질수록, 각 강화 레벨당 요구하는 강화 점수가 높다.

- **A-3-21-9.** 아이템의 강화는 오직 아이템의 주 능력만을 강화한다.
: 보조 능력 옵션들은 강화의 대상이 아니다.

※ 장비 아이템에서의 주 능력 옵션은, 아이템의 가장 위에 가장 크게 표시하는 능력을 말한다. 무기의 경우에는 공격력, 방어구의 경우에는 방어력의 수치가 바로 그 아이템의 주 능력이다. 아이템을 강화한다는 뜻은 이러한 주 능력에 대해서만 더 높은 수치로 향상시킨다는 뜻이다.

- **A-3-21-10.** 아이템 강화석을 아이템 강화의 재료로써 이용할 수 있다.
: 아이템 강화석은, 아이템 강화 외의 다른 용도로는 사용하지 못하는 소모성 아이템이다.
장비 아이템 대신, 아이템 강화석을 아이템 강화 재료로 쓰거나, 혹은 둘 다 섞어서 아이템 강화 재료로 쓸 수 있다.

※ 아이템 강화석이 강화 점수를 올려주는 방식에 따라 여러 단계가 존재하는지, 그리고 그러한 각 단계의 아이템 강화석을 재료로 사용할 때, 게임 플레이 화폐로 지불하는 금액에도 차등이 있는지 여부는 아직 결정하지 않았다.

※ 아이템 강화석은 한 가지의 특수한 용도로 밖에 사용할 수 없기 때문에, 서비스 운영 단계에서 제한적으로 플레이어들에게 보상을 줄 때 활용할 수 있는 재화 중 하나로 설계했다.

◆ A-3-22. 아이템 강화 레벨의 초월

- **A-3-22-1.** 각 아이템은 강화할 수 있는 강화 레벨의 한계치가 정해져 있다.
: 한계치까지 강화한 상태에서는 더 이상 어떤 아이템이나 아이템 강화석을 가져오더라도 강화 점수를 더 올릴 수 없다.

- **A-3-22-2.** 초월석을 통해서 해당 아이템에 설정되어 있는 최대 강화 레벨을 더 높일 수 있다.
: 최대 강화 레벨이 높아지면, 다시 추가적인 강화가 가능하다.
물론, 높아진 강화 레벨만큼 요구하는 강화 점수도 더 많아진다.

- **A-3-22-3.** 초월석의 종류마다 아이템 강화 레벨의 한계를 뛰어넘을 수 있는 정도가 다르다.
: 아이템 강화 레벨의 한계를 더 많이 상승시키는 초월석일수록 더 희귀하다.

※ 예를 들자면, '초월석 +5'는 아이템 강화 레벨의 최대 제한을 +5 증가시킨다. 그리고 '초월석

+10'은 아이템 강화 레벨의 최대 제한을 +10 증가시킨다.
'초월석 +5'가 있다고 해서, 아이템 강화 레벨의 최대 지한을 5 + 5 = 10처럼 증가할 수 없다.
아이템 강화 레벨의 최대 제한이 +10이 되기 위해서는 무조건 '초월석 +10'이 있어야 한다.

기술적으로는 아이템마다 '초월석 슬롯'이 있어서, 초월석 슬롯에 '초월석 +5'가 끼워진 경우와 '초월석 +10'이 끼워진 경우의 결과 처리 방식으로 처리 모델을 만들 수 있다.

- **A-3-22-4.** 아이템 강화 레벨을 초월할 때마다, 그에 대한 비용이 들어간다.
: 초월하는 레벨의 정도에 따라, 더 많은 비용을 지불해야 한다.

※ 아이템 강화 레벨을 초월할 경우의 구체적인 비용의 지불 방식은 아직 정해지지 않았다.

◆ A-3-23. 아이템 강화 보석

- **A-3-23-1.** 플레이어는 스테이지의 결과 보상으로써 보석을 획득할 수 있다.
: 일반적으로 주간 던전에서 고정적으로 획득하며, 다른 스테이지나 미션을 통해서 확률적으로 획득할 수도 있다.

- **A-3-23-2.** 아이템 중에서 보석을 장착할 수 있는 보석 소켓이 있는 아이템이 있다. 플레이어는 보석 소켓이 있는 아이템에 보석을 장착해서, 그 아이템의 능력을 더 강력하게 할 수 있다.

- **A-3-23-3.** 보석의 장착은 아이템 보관함 페이지에서 가능하다.
: 로비 장면에서만 아이템 보관함 페이지가 열리기 때문에, 보석의 장착 역시 로비 장면에서만 할 수 있다.

- **A-3-23-4.** 일반 보석
: 장착한 장비 아이템에 한 가지 능력 속성을 부여한다. 부여하는 능력 속성은 기존 장비 아이템의 옵션과 겹칠 수도 있다. (그런 경우에는 해당 능력의 수치가 더 높아지는 셈이다.)

- **A-3-23-5.** (스킬) 발동 보석
: 그 캐릭터의 일반 공격에 어떤 스킬 속성 하나를 부여하는 보석이다.
무기 아이템에만 장착할 수 있다.

- **A-3-23-6.** 각 보석의 종류마다 강화하는 능력치의 종류가 다르다.

- **A-3-23-7.** 각 보석의 종류마다 장착할 수 있는 장비 아이템 부위가 정해져 있다.
: 무기에만 장착 가능한 보석도 있고, 투구에만 장착 가능한 보석도 있고... 이런 식이다.

- **A-3-23-8.** 각 종류의 보석들은 여러 등급이 존재한다.

- **A-3-23-9.** 각 종류의 하위 등급 보석들은 일정한 개수를 모아서, 게임 플레이 화폐로 비용을 지불하고 바로 위 단계의 보석으로 합성할 수 있다.
: 재료와 비용만 충분하다면, 이를 통해 최하위 등급에서부터 최상위 등급까지 보석을 합성할 수 있다.

- **A-3-23-10.** 장비 아이템에 항상 보석 소켓이 달려 있지는 않다.
또한, 보석 소켓은 장비 아이템 당 최대 1개까지 존재할 수 있다.

- **A-3-23-11.** 보석 소켓 그 자체에는 특별한 조건이 없다. 보석의 종류와 등급을 막론하고 장착이 가능하다.

- **A-3-23-12.** 보석의 장착에는 비용이 들어가지 않는다.

- **A-3-23-13.** 장착한 보석을 교체하기 위해서는, 먼저 장착한 보석을 아이템으로부터 분리해야 한다. 이 과정에는 게임 플레이 화폐의 단위로 비용이 들어간다.
: 장착한 보석의 등급이 높을수록, 분리하는 비용이 늘어난다.

◆ A-3-24. 스킬 강화

- **A-3-24-1.** 각 스킬은 숙련 점수로 불리는 점수 체계가 존재하고, 이 점수가 일정 이상 높아지면 스킬의 숙련 레벨이 오른다.
: 캐릭터의 레벨과 경험치의 관계와 같다.

- **A-3-24-2.** 스킬을 강화하기 위해서, 플레이어는 스킬 인벤토리 페이지에서 강화할 스킬을 선택하고, 일정량의 게임 플레이 화폐를 지불한다.
화폐를 지불할 때마다, 일정한 단위 비율만큼 그 스킬의 숙련 점수가 증가한다.
스킬의 다음 레벨이 요구하는 것 이상으로 숙련 점수가 쌓이면, 그 스킬의 숙련 레벨이 오른 것이다. 그러면 스킬의 능력이 이전 레벨보다 더 강력해진다.

- **A-3-24-3.** 스킬 인벤토리는 로비 장면에서만 열 수 있으므로, 스킬의 강화 역시 로비 장면에서만 할 수 있다.

- **A-3-24-4.** 스킬의 레벨이 높아질수록, 그 스킬 레벨의 각 단위 비율만큼의 숙련도를 올리기 위해 요구하는 게임 플레이 화폐의 금액도 커진다.

- **A-3-24-5.** 스킬 룬을 스킬 숙련 점수를 올리는 재료로써 사용할 수 있다.

: 스킬의 숙련 점수는 기본적으로 게임 플레이 화폐를 지불함으로써 올리는 것이지만, 스킬 룬을 이용해서 추가적으로 숙련 점수를 더 올릴 수 있는 방식이다.

※ 아이템 강화와 다른 점이 이 부분이다.
아이템 강화는 어쨌든 재료로 쓸 다른 장비 아이템을 필수적으로 요구하는 반면, 스킬 강화는 재료로 쓸 스킬 룬을 필수적으로 요구하지 않는다.
이렇게 된 이유는 스킬 룬 시스템 자체가 게임 플레이 초반부터 얻을 수 있는 재화가 아니기 때문이다. 스킬 룬도 등급이 존재하기는 하지만, 가장 하위 등급의 스킬 룬이라도 어느 정도는 게임을 진행한 뒤에나 등장한다.
따라서, 스킬의 숙련 점수를 올리는 체계는 '기본적으로 게임 플레이 화폐로 올리는 것이지만, 스킬 룬을 추가적으로 재료로 사용하면 추가적인 스킬의 숙련 점수를 올릴 수 있다' 정도로 보면 된다.

- **A-3-24-6.** 숙련 재료로 등록한 각 스킬 룬마다, 스킬 숙련을 위한 재료로 '분해'하기 위한 비용을 게임 플레이 화폐로 내야 한다.

※ 즉, 숙련 재료로 등록한 스킬 룬이 많을수록, 지불해야 하는 게임 플레이 화폐도 더 많아지게 된다.

- **A-3-24-7.** 스킬 강화석을 스킬 숙련의 재료로써 이용할 수 있다.
: 스킬 강화석은, 스킬 강화 외의 다른 용도로는 사용하지 못하는 소모성 스킬이다.
추가적으로 스킬 숙련 점수를 높이기 위해 재료로 사용하는 스킬 룬들을 대신해서, 혹은 스킬 룬들과 섞어서 사용할 수 있다.

※ 스킬 강화석이 스킬의 숙련 점수를 올려주는 방식에 따라 여러 단계가 존재하는지, 그리고 그러한 각 단계의 스킬 강화석을 재료로 사용할 때, 게임 플레이 화폐로 지불하는 금액에도 차등이 있는지 여부는 아직 결정하지 않았다.

※ 스킬 강화석은 한 가지의 특수한 용도로 밖에 사용할 수 없기 때문에, 서비스 운영 단계에서 제한적으로 플레이어들에게 보상을 줄 때 활용할 수 있는 재화 중 하나로 설계했다.

~~◆ A-3-25. 스킬 룬 장착~~

~~- A-3-25-1. 스킬 룬은 캐릭터의 스킬에 장착할 경우, 보조 속성 하나를 추가해주는 재화이다.~~

~~- A-3-25-2. 어떤 스킬에 어떤 스킬 룬을 조합하는지에 따라 스킬의 유용성과 게임 플레이 스타일이 달라질 수 있음을 의도하여 제작한다.~~

- A-3-25-3. 스킬 룬 그 자체는 이 게임의 중급 이상 게이머들을 위한 컨텐츠로, **게임을 처음 배우고 익숙해지는 단계에서는 등장하지 않는다.**
: 제 아무리 하급 스킬 룬이라도 마찬가지다.

※ 게임을 처음 배우는 입장에서는, 제시된 하나의 길을 따라가도록 하는 편이 더 편안하게 느껴진다. 뭔지 잘 모르는 상태에서 선택을 강요하지 않도록 게임을 디자인하기 위해서, 스킬 룬과 같이 선택의 결과가 직관적이지 않은 컨텐츠들은 게임 진행의 중 후반부부터 등장하게 한다.
게임 진행이 막힌다거나, 뭔가 전투 효율을 더 좋게 하고 싶거나, 플레이 스타일을 좀 변화하고 싶을 때 찾는 느낌으로 제작할 컨텐츠다.

- A-3-25-4. 플레이어는 스테이지 보상, 혹은 뽑기 보상을 통해 스킬 룬을 획득할 수 있다.
: 초반부 스테이지에서는 등장하지 않는다.

- A-3-25-5. 각 스킬은 총 2개까지의 스킬 룬을 장착할 수 있다.
: 플레이어는 스킬 보관함 페이지에서, 선택한 스킬에 선택한 스킬 룬을 장착할 수 있다.

- A-3-25-6. 스킬 룬은 기획적으로 몇 가지 종류의 타입으로 분류한다.
그리고, 각 스킬들은 특정한 타입의 스킬 룬만 장착해야 할 수 있다.

※ 이는 스킬 룬이 스킬에 보조적인 속성을 추가하는 특성 때문이다.
특정한 스킬 룬들은 특정 스킬들과 결합하면 지나치게 강력한 시너지를 일으킬 수도 있다.
그렇다고 해서 스킬마다 일일이 사용이 금지되는 속성의 스킬 룬을 나열해서 표시하기도 어렵고, 플레이어 입장에서도 납득하기 쉽지 않을 것이다.
그래서, 상대적으로 자연스럽게 받아들일 수 있도록 '스킬 룬의 타입'을 두어서, 지나치게 강력해지는 속성을 가지는 스킬 룬들을 장착하지 못하는 타입의 스킬 룬으로 설정하는 식으로 밸런스를 맞춘다.

- A-3-25-7. 스킬 룬은 장비 아이템과 마찬가지로 등급이 존재한다.
: 스킬 룬의 등급은 그 스킬 룬의 옵션 능력의 강함에 의해 정해진다.

※ 장비 아이템과는 달리, 스킬 룬은 등급이 오르더라도 능력 옵션의 개수에는 변함이 없다.
최고 단계의 스킬 룬이라도 부여된 능력 속성의 종류는 1개 뿐이다. 다만 그 1개의 능력 속성의 능력 증가 값이 더 클 뿐이다.

※ 스킬 룬의 등급 단계는 장비 아이템의 등급 단계와 같을 수도, 아닐 수도 있다.

- A-3-25-8. 액티브 스킬, 패시브 스킬 여부에 따라 스킬 룬을 장착할 수 있는지 여부는 아직 결정되지 않았다.

~~- A-3-25-9. 스킬 룬의 장착에 비용이 들어가야 할까?~~

~~- A-3-25-10. 장착한 스킬 룬을 교체하기 위해서는, 먼저 장착한 스킬 룬을 스킬로부터 분리해야 한다. 이 과정에는 게임 플레이 화폐의 단위로 비용이 들어간다.~~
~~: 장착한 스킬 룬의 등급이 높을수록, 분리하는 비용이 늘어난다.~~

※ 스킬 룬 기능은 기획에서 배제하기로 하였다.

◆ A-3-26. 장착 아이템 제련(뽑기)

- **A-3-26-1.** 플레이어는 로비 장면에서, 아이템 제련소 페이지를 띄우고, 자신이 원하는 방식을 선택해서 장비 아이템을 뽑는다.

- **A-3-26-2.** 장비할 수 있는 종류의 아이템만 제련을 할 수 있다.
: 소모성 아이템은 뽑기의 대상이 되지 않는다.

- **A-3-26-3.** 아이템 뽑기에 지불하는 비용은 과금 화폐이다.
: 경우에 따라, 아이템을 무료로 뽑을 수 있는 제한된 횟수의 권한이 주어질 수 있다.

- **A-3-26-4.** 뽑기에 의해 등장하는 아이템의 옵션 종류와 등급은 완전히 무작위적이다.

- **A-3-26-5.** 일반 등급 아이템은 뽑기로 등장하지 않는다.
: 일반 등급 아이템은 능력 옵션이 딱 1개 밖에 없기 때문에(주 능력 옵션을 말한다. 공격력, 방어력 이런 거....), 무작위 뽑기를 하는 의미가 없다.

- **A-3-26-6.** 장비 아이템 뽑기는 한 번에 하나씩 할 수 있고, 뽑을 때마다 결과 화면에서 뽑힌 아이템과, 현재 장착 중인 같은 장비 종류의 아이템을 비교해준다.
: 이를 통해 즉석에서 더 좋은 아이템으로 교체할지 여부를 물어본다.

- **A-3-26-7.** 장착 아이템 뽑기(일반)
: 일반 아이템 뽑기를 할 때는, 캐릭터의 레벨을 고려해서, 등장할 아이템의 능력 옵션을 결정한다.
즉, 반드시 뽑기를 하는 캐릭터가 사용할 수 있는 종류와 능력을 가지는 아이템만 등장한다.

※ 이 게임에는 분명히 아이템마다 사용할 수 있는 최소 레벨 요구치가 존재한다.
그렇기 때문에, 종류와 옵션이 캐릭터의 최소 레벨 요구치를 넘어설 수 있는 아이템이 뽑기를 통해 등장하면, 캐릭터는 비용을 들여서(심지어 과금 화폐로 지불했는데!) 뽑은 아이템을 당장 사용할 수 없는 상황에 빠진다.

이런 상황은 사용자에게 불편과 불쾌함을 줄 수 있기 때문에, 애초에 뽑기를 시도하는 캐릭터가 사용할 수 있는 제한 범위 이내의 아이템으로 구성해야 한다.

※ 또한, 캐릭터의 레벨이 상승할수록 새로운 아이템 뽑기를 시도하게 만드는 요소 중 하나이기도 하다. 하위 레벨일 때 뽑은 아이템이 아무리 좋은 능력 옵션이 나왔다고 한들, 상위 레벨 아이템들과는 능력 옵션의 기본 수치 자체가 차원이 다를 정도로 차이가 난다.

- **A-3-26-8.** 장착 아이템 뽑기(고급)
: 일반 뽑기에 비해 비용이 훨씬 더 비싼 비용을 요구한다.
단, 고급 뽑기에 의해 등장하는 아이템의 능력 옵션은, 캐릭터의 현재 레벨에 맞춘 능력 옵션부터, 최고 레벨의 능력 옵션 사이에서 무작위로 골라진다.
또한 고급 뽑기로 뽑은 아이템은 **레벨 제한이 없다.**

※ 하위 레벨의 캐릭터일지라도, 상위 레벨의 아이템 능력 그대로를 사용하게 될 수도 있다는 뜻이다.

※ 소위 '시간을 돈으로 사고 싶은' 플레이어들을 위한 장치이다.
다만, 고급 뽑기를 한다고 해서 '일반적인' 뽑기에서 등장하지 않는 더 좋은 능력 옵션을 얻는다거나 하는 일은 없다.

- **A-3-26-9.** 무작위한 부위 뽑기
: 캐릭터가 장착할 수 있는 장비 부위 중에서, 무작위한 부위의 아이템을 뽑는다.
일반 <-> 고급 뽑기 방식과 별도의 분류이다.

※ 무작위한 부위라고 해서, 다른 직업 클래스가 장착할 수 있는 아이템이 등장하지는 않는다.
기본적으로, 뽑기를 수행하는 캐릭터가 사용할 수 있는 아이템 중에서만 뽑는다.

- **A-3-26-10.** 특정한 부위 뽑기
: 플레이어가 선택한 부위의 장비 아이템을 뽑는다.
무작위한 부위를 뽑는 메뉴보다 1회 당 뽑는 비용이 더 비싸다. (원하는 부위를 선택하기 때문이다.)
일반 <-> 고급 뽑기 방식과 별도의 분류이다.

- **A-3-26-11.** 아이템을 무료로 뽑을 수 있는 횟수가 있을 때, 무료로 뽑을 수 있는 권한을 먼저 소모한 뒤, 무료 뽑기 횟수가 전부 소진된 뒤부터 원래의 과금 화폐 비용을 차감한다.

◆ ~~A-3-27. 스킬 룬 제련(뽑기)~~

- A-3-27-1. 플레이어는 로비 장면에서, 스킬 룬 제련소 페이지를 띄우고, 자신이 원하는 방식을 선택해서 스킬 룬을 뽑는다.

- A-3-27-2. 스킬 룬 뽑기에 지불하는 비용은 과금 화폐이다.
: 경우에 따라, 스킬 룬을 무료로 뽑을 수 있는 제한된 횟수의 권한이 주어질 수도 있다.

- A-3-27-3. 뽑기에 의해 등장하는 스킬 룬의 능력 옵션 종류와 등급은 완전히 무작위적이다.
: 단, 캐릭터가 현재 사용할 수 있는 능력 옵션으로 등장한다.

※ 스킬 룬의 등급과 능력 옵션의 종류에 따라 최소 요구 레벨의 제한이 존재할지는 아직 결정되지 않았다.
스킬 룬은 능력 옵션의 강약으로 등급이 결정되므로, 만약 최소 요구 레벨 제한이 존재한다면, 사실상 스킬 룬의 등급 그 자체에 맞춰서 정하면 된다.

- A-3-27-4. 무작위 스킬 룬 뽑기
: 무작위 한 타입의 스킬 룬을 뽑는다.

- A-3-27-5. 특정 타입 스킬 룬 뽑기
: 스킬 룬의 옵션과 등급은 무작위 하지만, 스킬 룬의 타입은 고정해서 뽑는다.
무작위 스킬 룬 뽑기 기능보다 비용을 훨씬 비싸게 책정한다.

- A-3-27-6. 스킬 룬을 무료로 뽑을 수 있는 횟수가 존재한다면, 스킬 룬을 뽑을 때, 무료로 뽑을 수 있는 권한을 먼저 소모한 뒤, 무료 뽑기 횟수가 전부 소진된 뒤부터 원래의 과금 화폐 비용을 차감한다.

※ 스킬 룬 기능은 기획에서 배제하기로 하였다.

◆ A-3-28. 아이템의 형상 변환

- A-3-28-1. 이 게임은 초반에 얻을 수 있는 아이템이라고 해서 외형이 더 볼품없거나, 후반에 얻을 수 있는 아이템이라고 해서 더 웅장한 외형을 가지지 않는다.

- A-3-28-2. 형상 변환은 모델이 있는 장비 아이템에만 적용한다.
: 장신구들은 장비 아이템이기는 하지만, 아이템 보관함에 아이콘으로만 존재할 뿐, 캐릭터 모델의 일부로써 표현하지는 않는다.

- A-3-28-3. 형상 변환을 할 때는, 대상 아이템에 적용 가능한 형상들 중에서만 선택한다.
: 검에다가 갑옷의 형상을 입히거나 할 수는 없는 법이다.

- A-3-28-4. 플레이어는 로비 장면에서, 형상 변환 상점 페이지로 가서, 형상 변환을 원하는 아이템을 선택하고 원하는 방식으로 형상 변환을 한다.

- A-3-28-5. 플레이어는 선택한 형상 변환을 적용하기 전에 미리 볼 수 있다.
: 미리 보기를 통해 변화할 형상이 마음에 들지 않을 경우, 취소할 수 있다.

- A-3-28-6. 아이템의 형상은 변경하지 않는 한 지속적으로 유지된다.
: 기간 한정 형상 변환이 존재할지 여부는 결정하지 않았다.

- A-3-28-7. 형상 자체에는 소유권이 없다.
: 한 번 형상을 변경했으면, 이전 형상으로 다시 변경하고자 할 때도 마찬가지로 비용을 지불해야 한다.

- A-3-28-8. 형상 변경의 비용은 과금 화폐로 지불한다.
: 경우에 따라, 형상 변경을 무료로 할 수 있는 제한된 횟수의 권한이 주어질 수 있다.

- A-3-28-9. 무작위 형상 변환
: 일반적인 게임 플레이 단계에서 등장할 수 있는 아이템 형상들 중에서 무작위 한 형상으로 변환한다.

- A-3-28-10. 지정 형상 변환
: 플레이어가 변경할 형상을 직접 선택할 수 있다.
무작위 형상 변환보다 지불해야 하는 비용이 비싸다.

- A-3-28-11. 희귀 형상 변환
: 기본적으로 지정 형상 변환과 같으나, 일반적인 게임 플레이 단계에서 등장하지 않는 특별한 아이템 형상들도 선택이 가능하다.

※ 플레이어들의 소유욕과 과시욕을 자극할만한 외형을 가진 형상들이 여기에 속한다.

- A-3-28-12. 형상 변환을 무료로 뽑을 수 있는 횟수가 있을 때, 무료로 뽑을 수 있는 권한을 먼저 소모한 뒤, 무료 횟수가 전부 소진된 뒤부터 원래의 과금 화폐 비용을 차감한다

※ 아이템의 형상을 변환하는 기능은 기획에서 배제하기로 하였다.
아이템의 형상은 아바타 아이템(Avatar Item)이라고 하여, 외형을 변경할 수 있는 일종의 장착 가능한 아이템의 방식으로 변경하였다.

◆ A-3-29. 스킬의 형상 변환

~~- A-3-29-1. 스킬의 이펙트에 변화를 준다~~

~~- A-3-29-2. 스킬의 형상이 변해도, 능력은 달라지지 않는다.~~

~~- A-3-29-3. 플레이어는 로비 장면에서, 형상 변환 상점 페이지로 가서, 형상 변환을 원하는 아이템을 선택하고 원하는 방식으로 형상 변환을 한다.~~

~~- A-3-29-4. 플레이어는 선택한 형상 변환을 적용하기 전에 미리 볼 수 있다.~~
~~: 미리 보기를 통해 변화할 형상이 마음에 들지 않을 경우, 취소할 수 있다.~~

~~- A-3-29-5. 형상 자체에는 소유권이 없다.~~
~~: 한 번 형상을 변경했으면, 이전 형상으로 다시 변경하고자 할 때도 마찬가지로 비용을 지불해야 한다.~~

~~- A-3-29-6. 형상 변경의 비용은 과금 화폐로 지불한다.~~
~~: 경우에 따라, 형상 변경을 무료로 할 수 있는 제한된 횟수의 권한이 주어질 수 있다.~~

~~- A-3-29-7. 형상 변환을 무료로 뽑을 수 있는 횟수가 있을 때, 무료로 뽑을 수 있는 권한을 먼저 소모한 뒤, 무료 횟수가 전부 소진된 뒤부터 원래의 과금 화폐 비용을 차감한다~~

~~- A-3-29-8. 스킬의 형상 변환 시스템이 이 게임의 요소에 들어갈지에 대해 확실히 결정된 것은 없다.~~
~~: 만약 들어가더라도 최초 출시에는 넣을 수 없을 것이고, 업데이트 컨텐츠가 될 것이다.~~

※ 스킬의 형상을 변환하는 기능은 기획에서 배제하기로 하였다.

◆ A-3-30. 아이템 마법 부여

- **A-3-30-1.** 아이템의 능력 옵션을 다른 능력 옵션으로 변경하는 행위를 마법 부여라고 한다.

- **A-3-30-2.** 플레이어는 로비 장면에서, 아이템 마법 부여 상점 페이지를 열고, 원하는 아이템을 선택해서 마법 부여를 한다.

- **A-3-30-3.** 마법 부여는 비용을 지불할 때마다 1회씩 수행된다. 또한, 한 번 수행하면 취소할 수 없다.

- **A-3-30-4.** 마법 부여 행위 자체는 오직 아이템의 부가 옵션만 변경한다.

: 아이템의 종류, 등급, 착용 클래스, 주 능력치는 전혀 변경하지 않는다.

- **A-3-30-5.** 교체가 이루어질 부가 능력 옵션은, 소유자 캐릭터의 레벨을 고려해서 무작위 옵션으로 교체한다.
: 소유자 캐릭터의 레벨에 맞는 단계의 능력 옵션 중에서 무작위로 고른 옵션이 부여된다.

- **A-3-30-6.** 마법 부여의 비용은 과금 화폐로 지불한다.
: 경우에 따라, 마법 부여를 무료로 할 수 있는 제한된 횟수의 권한이 주어질 수 있다.

- **A-3-30-7.** 마법 부여를 하는 아이템의 보물 레벨과 등급이 높을수록, 1회 마법 부여를 수행하는 비용이 비싸진다.

- **A-3-30-8.** 전체 옵션을 무작위로 교체하는 경우
: 마법 부여를 수행할 때마다, 지정한 아이템의 모든 부가 능력 옵션들 전체가 바뀐다.

- **A-3-30-9.** 하나의 옵션을 골라서 1개 교체하는 경우
: 플레이어가 아이템의 옵션 중에서, 지정한 옵션 한 가지만 무작위로 다른 옵션으로 교체할 수 있다. 단, 이 때 지정한 옵션만을 반복해서 마법 부여로 교체할 수 있다.

- **A-3-30-10.** 각 장비 아이템마다 마법 부여의 방식을 한 번 선택하면, 이를 두 번 다시 바꿀 수 없다.
: 한 번 선택한 마법 부여의 방식만을 끝까지 사용해야 한다. 두 방식을 섞어서 번갈아 사용할 수 없다.

※ 위와 같은 제한사항들은, 반복적인 마법 부여로 인해 게임 컨텐츠의 효율 밸런스가 무너지지 않게 하기 위한 방침이다.
관련 명세 항목들에서 이 부분을 더 상세히 설명한다.

- **A-3-30-11.** 형상 변환을 무료로 뽑을 수 있는 횟수가 있을 때, 무료로 뽑을 수 있는 권한을 먼저 소모한 뒤, 무료 횟수가 전부 소진된 뒤부터 원래의 과금 화폐 비용을 차감한다

◆ A-3-31. 사용자 대전(P vs P)

- **A-3-31-1.** 전형적인 1 vs 1 방식의 캐릭터들의 대전
: 전형적인 방식이다. 진행이 지나치게 단순하게 느껴질 수도 있다.

- **A-3-31-2.** 플레이어가 키운 모든 캐릭터들이 참가하는 일종의 팀 단위 대전
: 좀 색다르긴 한데, 모두 '내 캐릭터'이기 때문에, 플레이어가 직접 조종을 해야 할까 싶은 면

이 있다. 과연 어느 캐릭터를 조종하게 할 것인지 결정하기 쉽지 않기 때문이다.
어쩌면 길드 전쟁과 유사한 방식으로 진행이 될 수 있다.

- **A-3-31-3.** 하루에 최대 5회까지 수행할 수 있다.

- **A-3-31-4.** 사용자 대전을 1회 플레이 할 경우, 일정한 시간 동안 대기 시간이 주어진다. 대기 시간 내에는 사용자 대전을 곧바로 다시 할 수 없다.
: 대기 시간이 모두 끝나면, 그 뒤부터 다시 사용자 대전을 수행할 수 있다.

- **A-3-31-5.** VIP 레벨이 향상되면, 사용자 대전을 다시 수행할 수 있게 되기까지의 대기 시간을 일정한 비용을 내고 곧장 초기화할 수 있다.

◆ A-3-32. 길드 던전(R vs E)

- **A-3-32-1.** **첫 출시에는 반영하지 않으며, 업데이트로 추가한다.**

- **A-3-32-2.** 길드 명예 점수를 얻는다.

- **A-3-32-3.** 구체적으로 어떤 방식으로 진행할지에 대해 기획이 결정되지 않았다.

◆ A-3-33. 길드 전쟁(R vs R)

- **A-3-33-1.** **첫 출시에는 반영하지 않으며, 업데이트로 추가한다.**

- **A-3-33-2.** 길드에 가입한 플레이어의 캐릭터는 길드 전쟁에 참여할 수 있다.

- **A-3-33-3.** 최대 참여 인원은 30 vs 30까지이고, 한 번에 전장에 나서는 인원은 10 vs 10까지 가능하다.

- **A-3-33-4.** 전장에 나선 인원이 사망하면 곧바로 나머지 인원이 자동으로 충원되는 방식이다. 더 이상 추가적으로 참여시킬 인원이 없을 때까지 인원 추가가 이루어진다.

<길드 전쟁의 개념 예시>

- **A-3-33-5.** 길드 전쟁에 참여하기 위해서는 그 길드에서 전쟁에 참여하겠다고 신청을 해야 하며, 어떤 캐릭터들이 전쟁에 참여할지 등록해야 한다.

- **A-3-33-6.** 길드 전쟁은 참여할 수 있는 실제의 시간 대역이 정해져 있다. 정해진 시간 대역 안에 길드 전쟁에 참가해야 한다.
  : 참가하지 않은 경우, 그 주기의 길드 전쟁에 대한 점수와 보상을 받지 못한다.

- **A-3-33-7.** 길드 전쟁의 결과는 명예 점수로 환산하며, 이를 토대로 길드 간 랭킹 순위를 산정하고, 랭킹 순위에 따른 보상을 받는다.

- **A-3-33-8.** 자신이 소속되어 있는 길드의 길드 전쟁 랭킹에 따라, 그 길드의 길드원 플레이어들은 게임을 진행할 때 도움을 주는 특별한 부여효과(Enchanted Effect)를 유지할 수 있다.

- **A-3-33-9.** 길드 명예 점수와 더불어 길드 코인을 획득할 수 있다.
  : 얻은 길드 코인은 길드 상점에서 희귀한 상품을 사는데 이용할 수 있다.

◆ A-3-34. 월드 보스 협공(World Boss Raid)

- **A-3-34-1.** 현실 시간으로 일정한 시간이 지날 때마다, 게임에 접속한 플레이어들 전체가 공략해야 하는 특수하게 강력한 보스 몬스터가 출현한다.
  : 정확하게는, 그러한 보스 몬스터가 등장하는 특수한 스테이지에 진입이 가능하다.

- **A-3-34-2.** 보스 협공 스테이지는 게임 세계에 접속해 있는 어떤 플레이어 캐릭터라도 접속이 가능하다.

- **A-3-34-3.** 보스 협공 스테이지가 '열리는' 시간은 사전에 플레이어들에게 공지한다.
: 플레이어들에게 공지하는 방법은 여러 가지가 있을 수 있다.

※ 기본적으로는 구체적인 시간을 알려주고, 접속 중이지 않은 사용자들에게는 푸시 알림을 전송하거나, 보스 협공 스테이지를 개시하는 시간이 가까워지는 주기마다 알림을 줄 수도 있다.

- **A-3-34-4.** 보스 협공 스테이지에 들어가면, 내 플레이어 캐릭터와 함께 무작위하게 등장하는 다른 플레이어 캐릭터들이 보인다.

- **A-3-34-5.** 다른 플레이어 캐릭터들은 등장하자마자 보스 몬스터를 향해 달려가서 공격하기 시작한다. 물론, 플레이어의 캐릭터는 수동 혹은 자동으로 선택적으로 조작할 수 있다.

- **A-3-34-6.** 월드 보스 몬스터가 사망하면, 공략에 참여한 플레이어들은 공략에 기여한 바에 의해 보상을 받을 수 있다.
: 공략에 기여한 정도는, 월드 보스 몬스터에게 얼마나 많은 피해를 줬는지 여부와 연속적인 공략 참여 여부, 마지막 타격 여부를 기준으로 하여 정량화한다. 그렇게 하여 계산된 수치를 바탕으로 플레이어들의 보상 서열을 나눈다.

※ 월드 보스 몬스터에게 더 많은 피해를 입힌 플레이어 캐릭터일수록 더 강력한 보상을 얻을 수 있으므로, 플레이어들의 참여 동기가 되기에 충분할 것이다.
또한, 피해를 더 많이 입히지는 못하더라도, 여러 번 참여를 유도하기 위해 일종의 노력상(...) 같은 개념의 추가적인 보상을 준다.

<월드 보스 협공의 개념 예시>

◆ A-3-35. 주기적인 플레이에 대한 보상

- **A-3-35-1.** 일일 단위 보상과 월 단위 보상이 존재한다.

- **A-3-35-2.** 계정 단위로 적용하고, 보상을 수여한다.
  : 게임에 접속해서 플레이 하는 그 자체에 대한 보상이기 때문이다.
  그래도 이 부분은 경우에 따라 변경될 수 있다.

- **A-3-35-3.** 오랫동안 플레이를 중단했다가 복귀한 사용자에게 복귀 기념 보상을 수여한다.
  : 구체적인 보상의 종류와 양은 정해야 한다.

- **A-3-35-4.** 보상은 우편함을 통해 수령 받는다.

◆ A-3-36. 일일 미션

- **A-3-36-1.** 매일 특정한 임무를 수행하면, 보상으로 재화를 수여한다.

- **A-3-36-2.** 캐릭터 단위로 적용하고, 보상을 수여한다.
  : 이 부분은 변경될 수도 있다.

- **A-3-36-3.** 보상은 우편함을 통해 수령 받는다.

◆ A-3-37. 스테이지의 보상 획득

- **A-3-37-1.** 스토리 모드에서 스테이지의 목표를 완료하면, ~~무작위하게 섞인 3개의 '카드' 중 플레이어가 하나를 선택해서 보상을 얻을 수 있다.~~ 사전에 정해진 보상 기준에 의해 무작위로 조합된 재화들을 보상으로 받는다.

- **A-3-37-2.** 획득한 보상은 즉시 아이템 인벤토리와 재화 단위에 쌓인다.
: 아이템이라면 아이템 인벤토리에 들어가고, 화폐 등은 화폐 단위에 쌓여서 수치가 증가한다.

~~- **A-3-37-3.** 보상 카드로부터 얻을 수 있는 보상의 타입은 아이템 외에도 스킬 룬, 게임 플레이 화폐, 과금 화폐도 가능하다.~~

~~- **A-3-37-4.** 플레이어가 고른 카드의 보상을 취득할 때, 플레이어가 고르지 않은 나머지 카드의 보상 종류를 확인할 수 있다.~~

~~- **A-3-37-5.** 만약 플레이어가 그 상태에서 보상을 얻지 않고 한 번 더 고르기를 원한다면, 일정한 비용(대개 얼마간의 과금 화폐가 된다.)을 지불하고서, 먼저 고른 카드를 제외한 나머지 카드 중에서 다시 무작위 뽑기를 할 수 있다.~~
~~: 이 과정은 고를 카드가 전혀 남지 않을 때까지 반복할 수 있다.~~

※ 수정(2015. 05. 18 조수운)
스테이지의 보상은 이제 더 이상 '카드 뽑기' 방식으로 이루어지지 않고, 자동 지급하는 방식으로 변경한다.
이 부분의 내용 역시 그에 맞춰서 개정하였다.

◆ A-3-38. 상점 이용

- **A-3-38-1.** 로비 장면에서 각 상점 페이지로 진입하는 버튼을 터치해서 상점 페이지로 들어갈 수 있다.

- **A-3-38-2.** 게임 플레이 장면 단계에서는 재화를 소비하는 활동 자체는 일부 가능하지만, 상점을 이용하지는 못한다.

※ 게임 플레이 과정에서 할 수 있는 대표적인 재화 소비 활동은, 스테이지를 진행하다 캐릭터가 사망 했을 때, 해당 스테이지를 종료하지 않고 재도전하는 ~~비용과, 스테이지를 완료해서 무작위~~

~~보상을 획득할 때 '한 번 더 다시 뽑기' 기능을 이용하는데 비용을 지불하는~~ 활동을 들 수 있다.

※ 수정(2015. 05. 18 조수운)
스테이지의 보상은 이제 더 이상 '카드 뽑기' 방식으로 이루어지지 않고, 자동 지급하는 방식으로 변경한다.
이 부분의 내용 역시 그에 맞춰서 개정하였다. ('한 번 더 뽑기'에 대한 내용이 불필요해짐)

- A-3-38-3. 화폐 구매
: 과금 화폐(보통 '보석(젬Gem)'으로 많이들 쓰며, 게임 플레이에서 얻기 매우 어렵거나 거의 얻을 수 없는)는 직접 현실의 통화(Real Money)를 지불해야 구매할 수 있다.
게임 플레이 화폐(보통 '골드(Gold)'로 많이들 쓰며, 게임 플레이에서 빈번하게 획득이 가능한)를 상점에서 구입하는 경우에는, 먼저 과금 화폐를 구매하고, 과금 화폐를 지불하고 게임 플레이 화폐를 구입한다.

- A-3-38-4. 입장 자원 / 일회성 아이템 구매
: 일반 스테이지 입장 자원이나, 스테이지 전투에 도움이 되는 부여효과를 주는 물약 셋 등의 1 회성 소모 아이템을 구매하는 항목이 존재한다.
이러한 소모 아이템들은 대개 게임 플레이 화폐를 통해 구입한다.

- A-3-38-5. 아이템 뽑기
: 스테이지에서 전투를 치르고 보상으로 아이템을 획득하는 대신, 상점에서 비용을 내고 무작위 능력 옵션을 가지는 아이템을 뽑을 수 있다.

~~**- A-3-38-6. 스킬 룬 뽑기**~~
~~: 스테이지에서 전투를 치르고 보상으로 스킬 룬을 획득하는 대신, 상점에서 비용을 내고 무작위 능력 옵션을 가지는 스킬 룬을 뽑을 수 있다.~~

※ 스킬 룬 시스템은 기획에서 제거되었다.

- A-3-38-7. 아이템 마법 부여
: 비용을 지불하고 아이템의 능력 옵션을 원하는 방식으로 다른 것으로 교체할 수 있다.

~~**- A-3-38-8. 형상 변환**~~
~~: 비용을 지불하고 아이템의 외형을 원하는 방식으로 다른 것으로 교체할 수 있다.~~

※ 형상 변환 시스템은 기획에서 제거되었다.

- A-3-38-9. 선물 뽑기

: 비용을 지불하고,, 재화에 속하는 것이라면 어떤 것이든 등장할 확률이 있는, 일종의 복권과 같은 성격의 선물 뽑기를 할 수 있다.

- **A-3-38-10. 초월 모드 상점**
: 초월 코인을 이용해 희귀한 재화나 상품을 구매할 수 있다.

- **A-3-38-11. 결투장 상점**
: 결투장 코인을 이용해 희귀한 재화나 상품을 구매할 수 있다.

- **A-3-38-12. 길드 상점**
: 길드 코인을 이용해 희귀한 재화나 상품을 구매할 수 있다.

- **A-3-38-13. 숨겨진 상점**
: 무작위 확률로 등장하는 숨겨진 상점에서 희귀한 재화나 상품을 구매할 수 있다.

◆ A-3-39. 아이템 강화

- **A-3-39-1.** 강화할 수 있는 아이템들은 전투에 필요한 일정한 능력치를 가지고 있고, 캐릭터가 장비 창 GUI에 장착할 수 있는 아이템이다.

※ 이러한 아이템들을 착용 가능한 장비 아이템(Wearable Equipment Item)으로 정의한다.
이에 대한 사항과 강화에 대한 사항은 관련 내용을 더 상세히 설명한 항목을 참고한다.

- **A-3-39-2.** 이런 아이템들은 스토리 모드 등의 전투 모드에서 일반 / 보스 몬스터를 사냥해서 얻거나, 스테이지의 보상으로써 얻을 수 있다.

- **A-3-39-3.** 장비 아이템들은 제각기 희귀한 정도에 따라 등급이 부여되고, 같은 등급의 같은 아이템일지라도 부가적인 능력 속성이 조금씩 다를 수 있다.
: 일반적으로 더 희귀한 등급의 아이템일수록 더 강력한 능력치와 더 많은 부가 속성을 가진다.

- **A-3-39-4.** 장비할 수 있는 아이템들은 아이템 별로 고유의 성장치를 가지며, 일정 정도 성장이 이루어지면, 장비 아이템이 보유하고 있는 능력이 더 강해진다.

- **A-3-39-5.** 장비 아이템을 강화하고자 한다면, 장비 아이템을 강화하는 메뉴를 선택하고, 강화할 대상 아이템과, 강화 재료로 사용할 아이템들을 골라야 한다.
: 특정한 GUI 슬롯에 위치 시켜야 할 수 있다.

- **A-3-39-6.** 강화할 대상이 될 장비 아이템과, 강화 재료 아이템들을 확인하고, 마지막으로 강화에 드는 비용을 확인한 뒤, 강화 확인 버튼을 누른다.

- **A-3-39-7.** 강화 재료가 되는 아이템들이 강화할 대상 아이템에게 흡수되었음을 보여주는 효과 애니메이션이 재생되고 나면(이런 애니메이션들은 늘 터치 한 번으로 건너뛸 수 있다.), 강화 재료로 사용한 아이템들은 모두 사라지고, 강화할 대상 아이템이 번쩍이는 효과와 함께 남는다.
  : 능력 옵션을 확인하면 능력 옵션의 값들이 더 큰 수치로 상향되었음을 볼 수 있다.

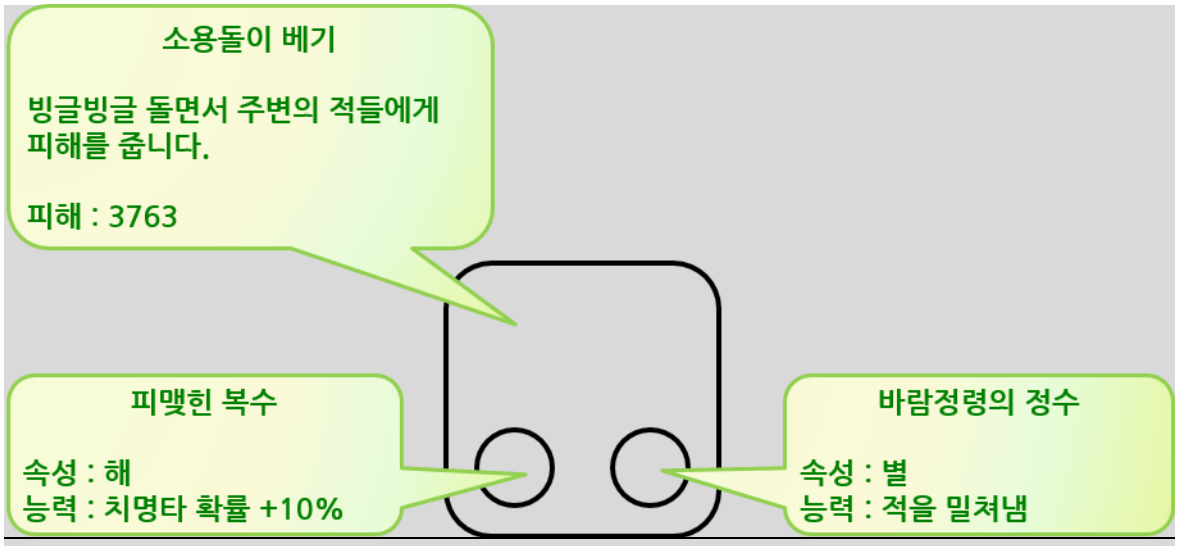
- **A-3-39-8.** 장비 아이템마다 흡수한 강화 재료 아이템으로부터 얻은 성장수치를 (프로그레스 바 등의 형태로)표기할 수도 있다.

◆ A-3-40. 스킬 강화

~~**A-3-40-1.** 각 스킬들은 스킬 룬(Skill Rune)을 장착하여 스킬의 능력을 강화할 수 있다.~~

~~**A-3-40-2.** 스킬 룬은 스킬에 추가적인 능력 옵션을 하나 더 주는 기능을 한다.~~

~~**A-3-40-3.** 스킬마다 스킬 룬을 최대 2개 씩 장착할 수 있다.~~



~~<스킬 룬의 간략한 개념 예시>~~

~~**A-3-40-4.** 스킬 룬들은 사전에 정의되어 있는 분류 집합이 있다. 그리고 스킬마다 지정된 분류 집합에 해당하는 스킬 룬만 장착할 수 있다.~~

※ 특정 옵션과 조합하면 능력이 지나치게 강해지거나, 게임 진행에 있어 일종의 꼼수가 되는 스킬과 스킬 룬의 조합을 방지하기 위한 장치이다.

~~- A-3-40-5. 스킬 룬 그 자체를 강화하는 기능은 제공하지 않는다.~~
~~: 다만, 스킬 룬들도 아이템처럼 등급이 있어서, 좀 더 강력한 스킬 룬과, 그렇지 않은 하급 스킬 룬으로 능력의 차별화를 두게 할 수 있다.~~

~~- A-3-40-6. 한 번 장착한 스킬 룬을 제거하기 위해서는 게임 상의 재화를 지불해야 한다.~~
~~: 자세한 내용은 관련 기획서의 내용을 참조한다.~~

※ 스킬 룬 시스템은 기획에서 제거되었다.

- **A-3-40-7.** 각 스킬의 강화 단계는 캐릭터의 최대 레벨 개수와 같다.

- **A-3-40-8.** (일단은) 모든 스킬들의 강화 단계 개수는 동일하다.

- **A-3-40-9.** 스킬의 레벨에 따른 비용을 지불하고 스킬을 강화할 수 있다.
: 스킬의 강화 단계가 높을수록, 강화에 필요한 비용도 더 증가한다.

- **A-3-40-10.** 스킬을 강화하는 방식은 스킬 레벨 1을 올려주는 스킬 포인트 방식이다.

◆ A-3-41. 게임 상의 친구 관리

- **A-3-41-1.** 친구 관리 서비스는 **사용자가 허가하지 않는 한, 다른 서비스로부터의 정보를 무단으로 사용하지 않고 이 게임 서비스 자체적으로 친구 관계를 유지 / 관리**한다.

※ 모바일 기기의 경우, 기기 내의 전화번호부에 등록되어 있는 지인 관계 정보를 들 수 있다.
사회 관계망 서비스(Social Network Service SNS)의 지인 관계 정보 역시 마찬가지 대상이다.
사용자가 허가할 경우에만 친구 관리에 사용한다.

- **A-3-41-2.** 로비에서는 주변의 친구가 될만한 플레이어 캐릭터들을 검색해주는 메뉴가 존재한다. 원한다면 친구 검색 버튼을 통해, 주변의 플레이어 캐릭터들의 목록을 보고 친구 신청을 할 수 있다.

- **A-3-41-3.** 다른 플레이어로부터 친구 신청이 올 수도 있다.
: 신청한 플레이어 캐릭터들의 목록을 확인할 수 있는 메뉴가 있으며, 언제 신청이 들어왔는지, 또한 이를 수락할지, 거절할지 결정할 수 있다.

- **A-3-41-4.** 친구 관리 메뉴에서는 플레이어의 현재 친구들의 목록을 볼 수 있다.

그리고 각 친구 플레이어들이 마지막으로 접속한지 얼마나 지났는지도 확인할 수 있다.

- **A-3-41-5.** 친구 목록, 친구 검색 목록에 등장하는 플레이어 캐릭터 목록들을 터치해서 그 플레이어 캐릭터의 로비에 진입할 수 있다.
: 그 캐릭터의 이름, 현재 레벨과 장비한 아이템, 보유한 아이템, 상세한 능력치 등을 확인할 수 있다.

※ 높은 레벨과 최고 수준의 아이템과 스킬 목록을 보유한 캐릭터들의 외형적인 '강력함'을 시각적인 면과 수치적인 면(능력치)으로 보여줌으로써 게임 플레이에 대한 향상심을 자극하려는 의도가 있다.

◆ A-3-42. 길드 가입 / 탈퇴

- **A-3-42-1.** **첫 출시에는 반영하지 않으며, 업데이트로 추가한다.**

- **A-3-42-2.** XX 레벨 이상이 된 플레이어 캐릭터는 길드를 창설할 수 있다.

- **A-3-42-3.** 또는, 길드 탐색 메뉴를 통해서 주변에 있는 여러 길드들의 목록을 보고, 적당한 길드를 선택해서 가입을 할 수도 있다.

- **A-3-42-4.** 길드를 창설한 플레이어 캐릭터가 길드의 장이 된다.

- **A-3-42-5.** 길드 탈퇴는 길드 원 개인의 의사만으로 탈퇴 버튼과 확인 버튼을 통해 가능하다.

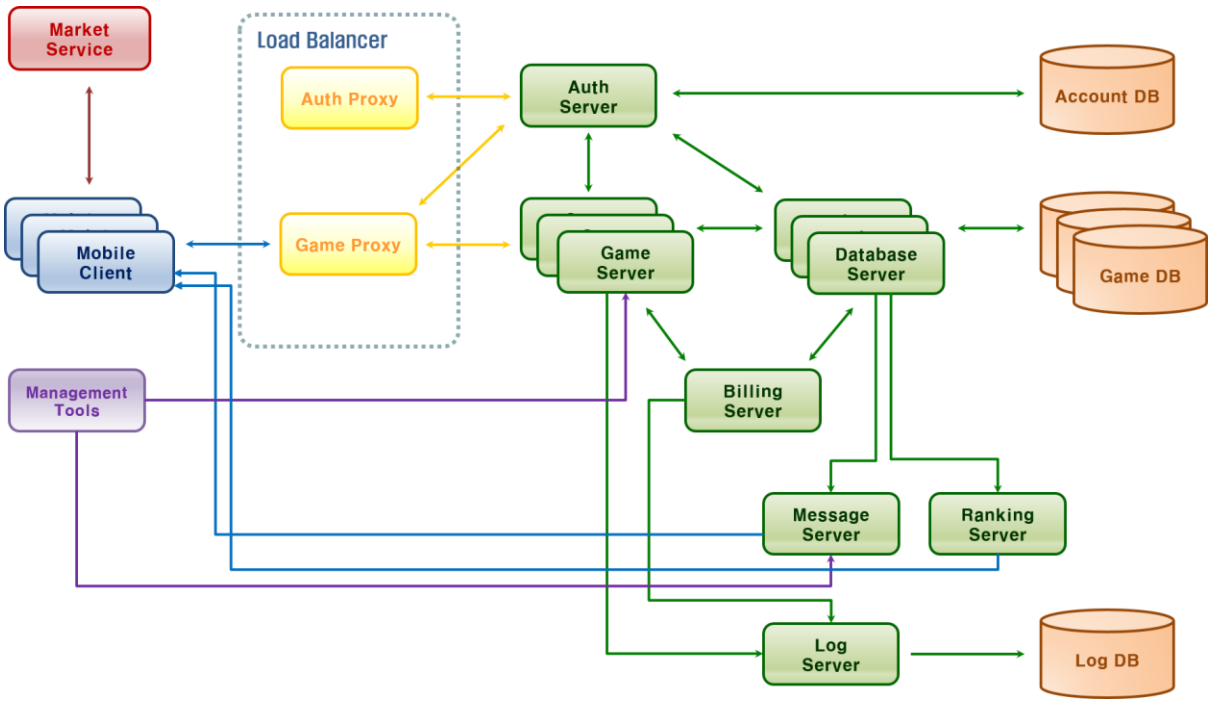
- **A-3-42-6.** 길드가 전쟁 준비에 등록되어 있거나 전쟁 중인 상태에서는 길드 가입 및 탈퇴가 불가능하다.

※ 길드 전쟁 중에 길드 원의 가입과 탈퇴가 가능하다면, 이를 서버 단계에서 적절히 처리하면서 관리하기도 어려울 뿐 아니라, 이런 시스템을 악용할 가능성도 있다.
: 가짜 회원들을 가입하게 한 뒤, 전쟁이 일어나기 직전에 싹 탈퇴하게 한다든가...

- **A-3-42-7.** 길드장은 길드장의 권한을 다른 길드원에게 위임할 수 있다.
: 길드장 권한이 있을 때에만 위임 여부를 결정할 수 있으며, 일단 길드장 권한이 넘어간 뒤에는 일반 길드원이 되기 때문에, 다시 어떤 방식으로든 길드장이 되기 전까지는 길드장의 고유한 명령을 더 이상 이용할 수 없다.

A-4. 시스템 모델

◆ A-4-1. 전체 시스템 구조



<서비스의 대략적인 조감도>

- A-4-1-1. 서버

: 서버는 게임 서비스 상에서 맡은 임무에 따라 여러 종류의 서버가 하나의 서비스를 위해 동작할 수 있는 구조로 제작한다.

서버는 일반 사용자들이 클라이언트 프로그램으로 접속해서 게임 상에서 활동했던 정보들을 저장하거나 내려 보내주는 역할을 담당한다. 또한, 게임 서비스에 있어 영업상 / 보안상 중요하거나 제어를 유지해야 할 필요가 있는 게임 내용에 대한 판정을 수행하기도 한다.

- A-4-1-2. 클라이언트

: 시장에서 사용자가 획득할 수 있는 프로그램은 게임 서비스에서 클라이언트로 사용하는 응용 프로그램 패키지이다.

일반 사용자 용 클라이언트는 모바일 기기를 통해 설치할 수 있는 버전만을, 해당 모바일 애플리케이션 마켓을 통해 제공한다.

- A-4-1-3. 데이터베이스

: 데이터베이스는 서버와 통신하며, 클라이언트로 접속한 사용자들의 게임 활동 내역을 기록하여 보관하는 역할을 한다.
기록 대상은 사용자의 계정 정보부터 시작해서, 게임 플레이에서의 진행 내역, 업적, 재화 보유 정도, 성장 단계, 로그 데이터까지 다양하다.
데이터베이스는 클라이언트에게 직접 노출하지 않으며, 신뢰할 수 있는 서버하고만 통신한다.

- **A-4-1-4.** **관리도구**
: 관리도구들은 게임 서비스 계약상 허가된 권한을 가진 관리자가 게임 서비스를 위한 활동을 할 수 있도록 돕는 소프트웨어 도구이다.
이러한 도구들은 일종의 특별한 클라이언트로써 작동한다.
일반 사용자에게는 절대로 배포하지 않으며, 웹 브라우저 기반 인터페이스로 작동한다.

◆ A-4-2. 개별 시스템의 임무 한계

- **A-4-2-1.** 게임에 대한 시각 / 청각적인 모델 리소스, 음향 리소스, 미디어 데이터와 이를 불러오는데 필요한 사항들을 제어하는 스크립트들은 클라이언트가 담당한다.

- **A-4-2-2.** 서버는 클라이언트의 접속 여부를 인증하고 각 클라이언트의 계정 정보를 최종 승인하고 저장한다.

- **A-4-2-3.** 게임 내의 전투와 관련된 판정 처리와 연산 처리는 개별 클라이언트가 판정 및 연산의 주체가 된다.

※ 완전한 보안을 위해서는 클라이언트가 판정과 연산의 주체가 되어서는 안되겠으나, 이 게임의 목표가 실시간으로 서버와 현재 플레이 상태를 통신하며 진행하는 게임이 아니기 때문에, 조건을 좀 더 완화한다.
서버에서 판정 연산을 하지 않기 때문에 발생할 수 있는 위험은, **클라이언트에 치트를 시도하더라도, 획득할 수 있는 보상의 한계치만큼은 서버가 직접 제어**하도록 하는 방식으로 관리할 생각이다.

- **A-4-2-4.** 게임의 진행 승인과 보상 판정에 관련된 내용들은 서버에서 판정한다. 클라이언트는 필요한 시점에 서버로부터 그 내용을 전달받아 반영한다.

- **A-4-2-5.** 서버는 각 클라이언트의 데이터를 최종적으로 저장하며, 클라이언트가 요청하는 시점에 저장하고 있는 데이터는 넘겨준다.

- **A-4-2-6.** 게임 플레이 상태를 복수의 클라이언트가 실시간으로 동기화해야 하는 방식의 게임 시스템은 제작하지 않는다.

: 모든 클라이언트들은 시스템 상으로 독립적으로 동작한다.
서버가 여러 클라이언트에게 같은 게임 요소에 대해 데이터를 갱신하더라도, 이는 각 클라이언트들에게 비동기적으로 갱신된다.

※ 이러한 제약 하에서는, 여러 명의 클라이언트가 하나의 '방'에서 서로의 데이터를 매 순간 동기화 하는 방식의 PvP 컨텐츠, RvR 컨텐츠 등은 가능하지 않다.
굳이 구현하고자 한다면, 각 클라이언트의 현재 게임 진행 상황을 매 순간 동기화하지 않는 방식으로 설계해야 한다.

◆ A-4-3. 공통 시스템

- **A-4-3-1.** 응용 프로그램의 버전 번호는 각 모듈이 공유한다.
: 즉, 특별한 경우가 아니라면 서버, 클라이언트, 관리 툴 등의 각 모듈 프로그램들은 하나의 버전 번호를 쓴다.

※ 각 모듈마다 다른 버전 체계를 가지는 경우에 비해서, 버전 번호만 보고 곧바로 호환 가능 여부를 알 수 있다는 점이 가장 큰 장점이다.
모듈마다 버전 번호 체계가 다른 경우에는 모듈마다 호환되는지 여부를 따로 기록해놓지 않으면 알기 힘들고, 그렇게 체계적으로 기록하기도 어렵다.
대신, 하나의 버전 체계일 경우에는 버전을 부여할 때 기준이 명확해야 하고, 하나의 모듈로 인해 버전을 판올림하게 되면, 다른 전체 모듈의 버전도 같이 판올림해야 하는 부담이 있다. (하지만 공통 라이브러리를 이용한다면 이런 부담은 사실 없는 거나 마찬가지다.)

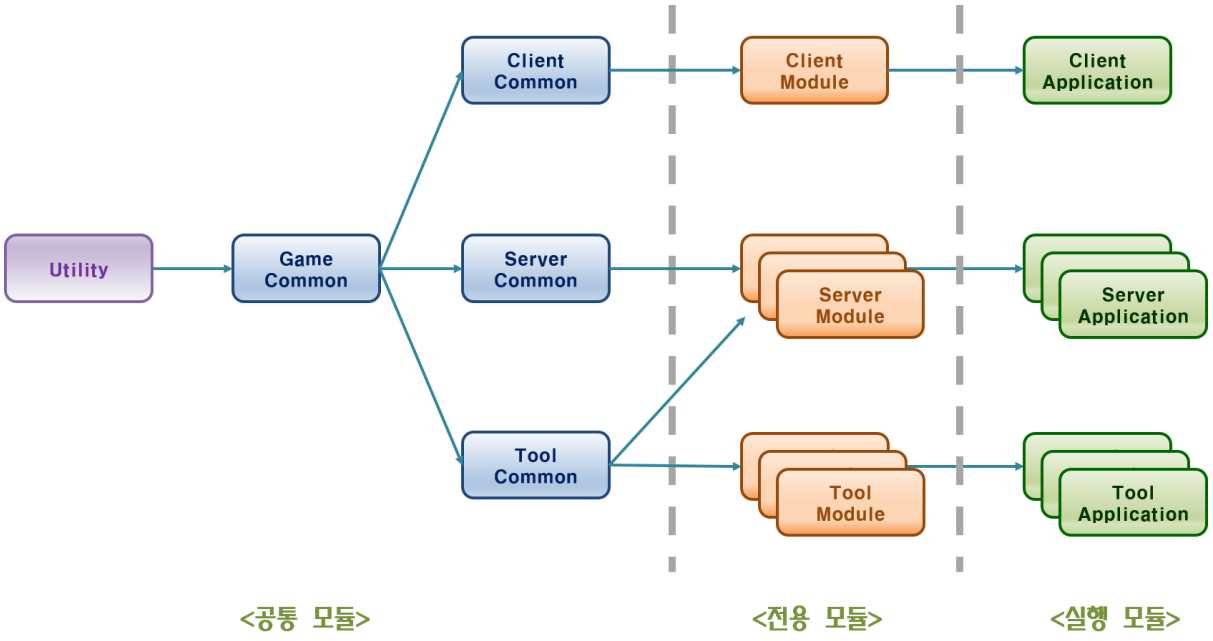
- **A-4-3-2.** 특정한 엔진 코드나 외부 라이브러리에 의존적이지 않은 코드들은 가급적 별도의 공통 모듈로 묶어서, 이를 링크해서 사용하는 방식으로 작성한다.

※ 여러 가지 역할을 하는 다수의 애플리케이션과 라이브러리들이 포함된 솔루션에서 제일 문제가 되는 점 중에 하나가 이와 관련된 문제들이다. 코드 베이스가 커지면서, 비슷한 기능을 하는 코드들이 클라이언트, 서버, 관리 툴, 저작 툴 등에 여기저기 따로 작성되어 있는 현상이다.
이는 코드를 관리하는 데 있어 큰 부담을 준다. 비슷한 소스 코드가 몇 벌씩 되기 때문에 버그를 고쳐도 다른 곳에서 쓰는 비슷한 코드에서는 고쳐지지 않았을 수도 있다. 심지어, 기능은 같은데 코드는 조금씩 다르기도 하다.
이에 대한 유일한 해결책은 공통적인 부분을 하나의 상위 라이브러리 프로젝트로 모듈화해서, 그것을 활용하는 모든 하위 모듈들이 그 상위 모듈의 라이브러리를 링크해서 사용하는 것이다.

- **A-4-3-3.** 공통 시스템 모듈은 **반드시 공유할 필요가 있는 모듈끼리만 묶어서 제작**한다.
: 전체 모듈(클라이언트 – 서버 - 관리 도구)가 모두 알아야 하는 사항들이 있고, 일부 모듈들끼리만 공유해야 하는 사항들이 존재할 것이다. (클라이언트 – 서버 관계에서만 알아야 하는 사항,

서버 – 관리 도구끼리만 알아야 하는 사항 등)

※ 관리용 도구는 서비스 통계 정보라든가, 관리자 클라이언트(소위 게임 마스터Game Master가 이용해야 하는 특수한 권한을 가진 클라이언트) 등을 포함한다. 이들은 게임 서비스에 사용하는 서버들과 통신하지만, 일반적인 사용자 클라이언트들과는 좀 다른 정보를 주고받아야 한다.
특히 이런 정보들은 **민감한 정보를 포함하거나, 사용자에게 가급적 노출하지 말아야 하는 정보도 있을 수 있기 때문**에, 애초에 사용자 클라이언트의 모듈에 포함하지 않으면서도 서버와는 정보 형식을 공유할 수 있도록 설계해야 한다.
그리고 대개 서버는, 서버가 아닌 모든 모듈들이 서버와 공유해야 하는 사항에 대해 알고 있어야 한다. 다른 모듈들은 단독으로 동작하지 않고, 다들 서버와 통신하면서 동작하기 때문이다.



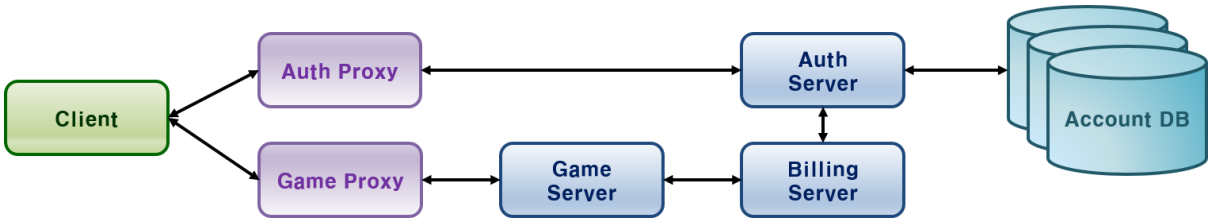
<시스템의 설계 모듈들의 관계 개념>

◆ A-4-4. 데이터베이스

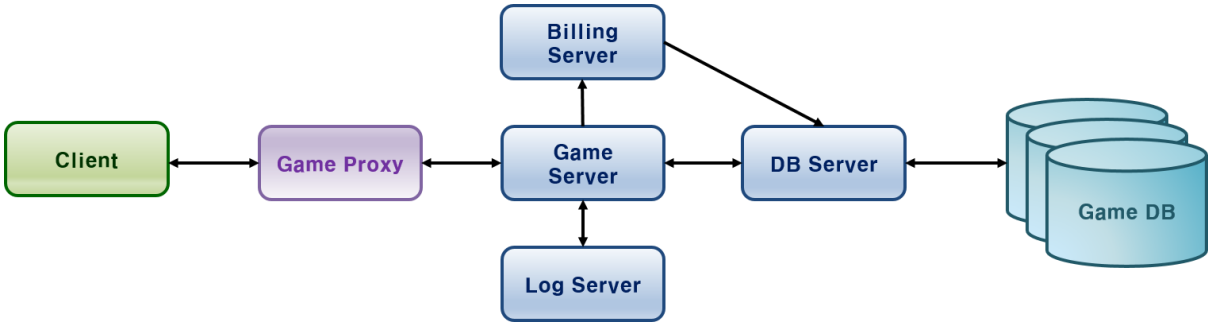
- **A-4-4-1.** 영구적인 저장이 필요하다고 분류한 데이터들은 데이터베이스에 저장한다.
: 계정 이름 / 패스워드, 복구에 필요한 연락처(E-Mail), 생성한 캐릭터에 대한 정보, 보유한 아이템 정보, 보유한 재화 정보, 각 캐릭터 별 게임 진행 정도와 스킬 선택 사항, 장착한 아이템 정보...

- **A-4-4-2.** 계정 인증에 대한 데이터베이스는 전세계에서 한 개만 운영한다.
: 만일 이 인증 데이터베이스가 별도로 운용되어야 한다면, 같은 이름의 계정이 각 인증 데이터베이스마다 존재할 수 있다.

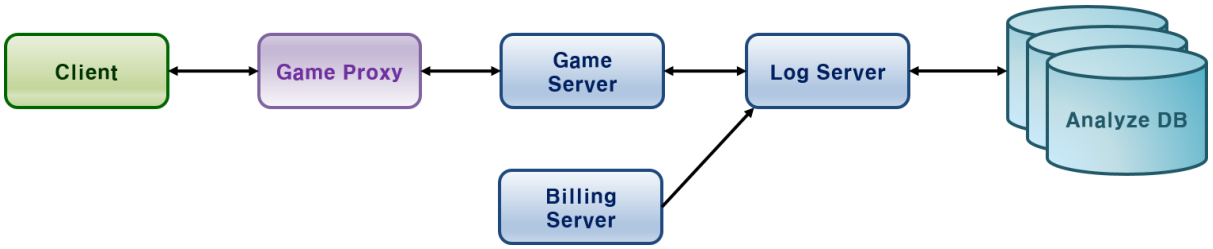
- **A-4-4-3.** 데이터베이스의 대략적인 조망



<인증 정보 부분>



<게임 정보 부분>



<분석 정보(통상 로그 데이터) 부분>

- **A-4-4-4.** 데이터베이스는 Master – Slave로 구성한다.

- **A-4-4-5.** 데이터베이스의 복제(Replication)

- **A-4-4-6.** 데이터베이스 분할(샤딩Sharding) 구조
  : 기능 별로 분할하고 (수직), 들어갈 아이템의 ID 구간 별로 분할한다.(수평)

- **A-4-4-7.** 데이터베이스는 데이터를 읽거나 쓰는 동작만 한다.

- **A-4-4-8.** 데이터베이스에게 질의(Query)를 요청할 경우, 반드시 저장 프로시저(Stored Procedure)를 통해서만 요청한다.
  : SQL 주입 공격(SQL Injection)을 방지하기 위한 기초적인 수단의 하나로써 사용한다.

※ 다만, 저장 프로시저의 내부 구문의 동작이 너무 복잡해지지 않도록 주의해야 한다.

가급적 복잡한 계산은 데이터베이스 외부에서 처리하는 게 바람직하다. 데이터베이스는 쉽게 확장하기가 어렵기 때문이다.

◆ A-4-5. 서버

- **A-4-5-1.** 서버 그룹의 모델
: 모델 그림을 넣어야 한다.

- **A-4-5-2. 인증 서버**
: 사용자들의 계정과 접속 허가 여부를 담당한다.
전체 게임 서비스 영역 중에서 1대만 존재해야 한다.

- **A-4-5-3. 게임 서버**
: 계정에 로그인 성공한 사용자들은 여러 대의 게임 서버가 나눠서 관리하면서, 사용자들의 게임 플레이 상황에 대해 통신을 주고받는다.

- **A-4-5-4. 패치 서버**
: 클라이언트의 버전을 확인하고, 필요한 데이터들을 내려 받을 수 있을 수 있게 조치해 주는 서버이다.
실제 구현에 따라, 게임 서버의 한 기능이 될 수도 있다.

- **A-4-5-5. 결제 서버**
: 사용자의 결제 내용을 처리하는 서버.
여기서 결제 확인이 되어야 구입한 재화를 플레이어에게 지급한다. 결제 확인이 되지 않은 경우, 확인이 될 때까지 클라이언트 측에서는 확인을 요청할 수만 있다.

※ 모바일 기기에서, 결제 행위 그 자체는 모바일 앱과 마켓 간의 통신으로 이루어지기 때문이다. 서버가 직접 마켓과 통신해서 결제 여부를 확인할 수 없기 때문에, 클라이언트 측에서 마켓으로부터 받아온 영수증 데이터를 기반으로 하여 결제 여부를 판단해야 한다.

- **A-4-5-6. 메시지 서버**
: 접속 중인 사용자들 간의 대화를 중계하고, 접속 중이지 않은 사용자들에게 필요한 공지 메시지를 전달한다.
채팅 행위 자체는 대개 부수적인 작업이기 때문에, 주된 게임진행 기능에 장애가 되지 않도록 별도의 서버로 구성한다.

※ 이렇게 사전에 분리를 해놓으면, 채팅 서버 쪽에 문제가 생기더라도, 채팅 기능만 정지되고, 나머지 게임 플레이는 정상적으로 진행이 가능해진다.

- **A-4-5-7.** **대전 서버**
: 플레이어들 간의 전투를 하는 컨텐츠가 존재할 때, 이 컨텐츠의 세부사항을 관리한다.
플레이어의 전투력 순위와 전투력 수치를 감안해서 대전을 성립해주고, 그 결과를 산출한다.

- **A-4-5-8.** **랭킹 서버**
: 사용자의 요청이 있을 때마다, 적합한 분야의 순위를 산출해서 전달한다.

- **A-4-5-9.** **로그 서버**
: 사용자들의 활동 내역과, 서비스 중 발생하는 각종 오류 상황들을 사전에 정의한 텍스트의 형태로 수집하는 일을 담당한다.
로그 서버는 게임 서버 - DB 하고만 통신을 해야 할까? 아니면 필요에 따라 클라이언트에게 직접 접근할 필요가 있을까? 아마도 전자라고 짐작이 되지만...

◆ A-4-6. 클라이언트

- **A-4-6-1.** 일반 사용자가 플레이 할 수 있는 클라이언트는, **모바일 기기 용으로 배포하는 클라이언트만 가능**하다.
: 가상 기계나 PC용(x86용 Android 운영체제 등)으로 클라이언트를 실행시킬 수 있을 수도 있지만, 일단 '공식적으로 지원하지는 않는다.'

- **A-4-6-2.** 각 클라이언트가 게임을 플레이 하기 위해서는, 클라이언트가 실행되고 있는 기기의 네트워크 서비스가 사용 가능한 상태여야 한다.

- **A-4-6-3.** 클라이언트는 서버와 상시 연결을 지속하지 않기 때문에, 덜 중요한 부분에 대해서는 클라이언트의 계산 결과를 대체로 신뢰하고 사용한다.
: 다만, 이에 대해 제한적으로 검증할 수 있는 수단을 마련한다.

※ 이를 위해, 클라이언트의 계산 결과가 조작되더라도, 그로부터 얻을 수 있는 이득이 미미하게 되도록 게임 방식을 설계한다.

- **A-4-6-4.** 보상과 재화에 대한 판정은 어떤 식으로든 절대로 클라이언트에서 진행하지 않는다.

- **A-4-6-5.** 클라이언트 그 자체로는 전체 게임의 데이터를 구성하지 못하도록 제작한다.
: 대표적인 부분은 아이템의 생성 방식, 확률, 능력 옵션을 부여하는 방식, 보상 패턴, 보상의 종류 등이다.

※ 그래픽 요소나 게임 엔진 내부의 기능과 전혀 관련이 없으며, 게임의 경제 시스템을 돌아가게

만드는 데 필요한 핵심적인 공식과 데이터들은, 공중(public)에 배포하는 클라이언트 측이 가지지 못하게 만들어야 한다.

◆ A-4-7. 관리도구

- **A-4-7-1.** 관리도구는 웹 브라우저 기반으로 제작한다.

- **A-4-7-2.** 관리도구의 실행은 웹 브라우저를 필요로 한다.
  : 별도의 플러그 인 도구가 필요한지 여부는, 제작 방식의 결정에 따른다. 플러그 인 선정에 대한 특별한 제약 사항은 없다.

- **A-4-7-3.** 다음 웹 브라우저에서는 필수적으로 실행이 가능해야 한다.
  : Microsoft Internet Explorer, Mozilla Firefox, Google Crome 의 최신 버전

※ Apple Safari나 Opera Software Opera 브라우저의 경우에는 좀 애매하다.
일반적인 관리 환경 상, '반드시' 지원해야 할 정도인지는 좀 애매하기 때문이다. 그냥, 필수적으로 지원해야 하는 브라우저와 마찬가지로 잘 동작하면 상관이 없고, 뭔가 문제가 생긴다면 그 때 필요성에 따라 다시 판단을 해봐야 할 것이다.
일반 사용자를 대상으로 하는 소프트웨어가 아니기 때문에, 모든 브라우저 환경에서 완벽하게 실행을 보장하도록 비용을 들이기가 어려운 측면이 있다.

- **A-4-7-4.** 웹 브라우저가 최신 버전인 경우에는 기능 수행을 100% 보장해야 한다.
  : 구형 버전인 경우에는 이러한 보장 의무가 없다. (억울하면 웹 브라우저를 최신으로 업데이트 하도록 한다. -ㅅ-)

◆ A-4-8. 가용성 관리

- **A-4-8-1.**

◆ A-4-9. 보안

- **A-4-9-1.** 게임 서비스와 관련한 보안 사항만을 관리한다.
  : 게임 서비스와 직접 관련이 없는, 사용자의 기기 자체에 대한 보안에 대해서는 책임을 지지 않는다.

- **A-4-9-2.** 계정 정보 보안

- **A-4-9-3.** 결제 시스템 보안

- **A-4-9-4.** 클라이언트 데이터 보안

- **A-4-9-5.** 코드 보안

- **A-4-9-6.** 루팅 방지 처리

◆ A-4-10. 시스템 백업

- **A-4-10-1.**

◆ A-4-11. 유지보수

- **A-4-11-1.**

B. 게임 시스템

B-1. 게임의 장면(Scene) 구조

◆ B-1-1. 장면의 특질

- **B-1-1-1.** 게임 진행 단계에서, 각 장면 단계들은 다른 장면 단계와 동시에 진행되는 일이 없이 **독점적으로 동작**한다.
: 예를 들자면, Intro 단계와 Lobby 단계, GamePlay 단계는 동시에 진행되는 법이 없다. 셋 중 하나만 진행해야 한다.

- **B-1-1-2.** 다만. 위에서 말했듯, **로딩 장면의 경우는 특별**하다.
: 로딩 장면의 기능 자체가 다른 장면과 장면을 이어주는 역할이기 때문에, 이 경우에만 특별하게 다른 장면과 공존이 가능할 수 있어야 한다.

- **B-1-1-3.** **장면과 장면 사이에서 사용하는 데이터들은 원칙적으로 직접적인 의존성을 가지지 않는다**.
: 일부, 장면과 장면 간 통신이 필요한 경우에는, 전역 관리자들을 통해 값을 저장하고 전달하는 방식을 사용한다.

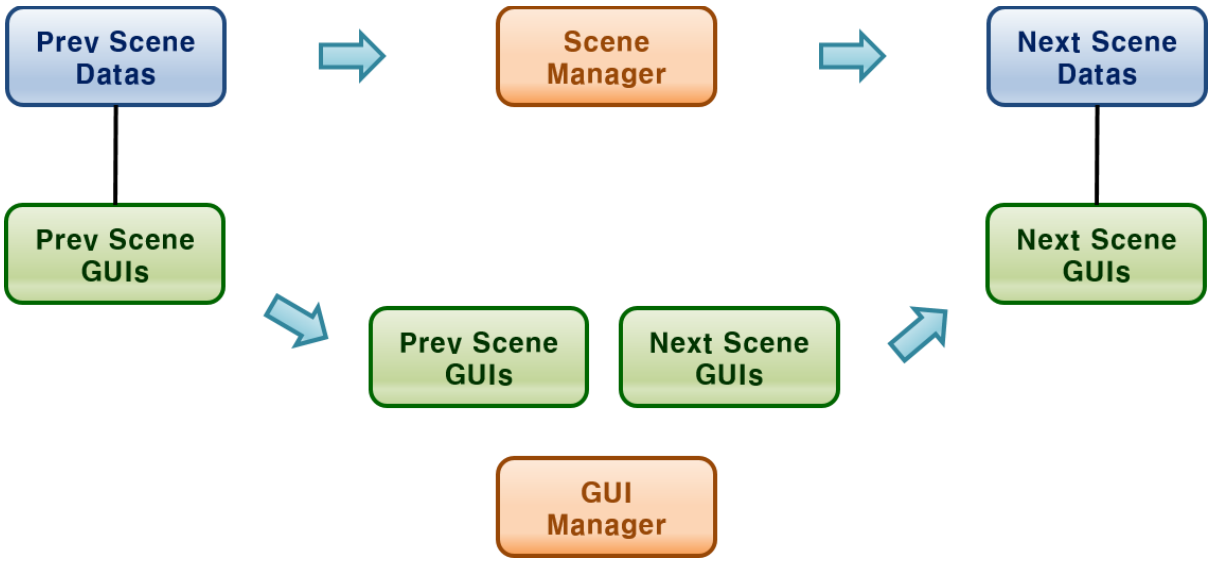
- **B-1-1-4.** 일반적으로, 장면을 전환할 때는 이전 장면을 불러들일 때 사용하는 데이터들을 모두 메모리에서 해제하고, 다음에 불러올 장면의 데이터들을 메모리에 올린다.

※ 장면에서 관리하는 데이터 대다수가 다른 장면과 의존성을 가지면 곤란한 이유이기도 하다.
장면 간에 의존성이 존재하는 데이터라면 아예 전역 데이터로 설정해야 하며, 장면 레벨에서 관리하면 메모리 사용량을 억제하기 어려워진다. (죄다 메모리에 올려놓는 수 밖에 없으니...)

예를 들면, 내 캐릭터의 능력치 정보는 게임 플레이 장면이 아니더라도 로비 장면, 상점 장면에서도 필요하다. 하지만 모델 데이터의 경우에는 게임 플레이 장면, 혹은 잘해야 추가적으로 로비 장면 정도에서나 필요하다.
그렇다면, 캐릭터의 능력치 정보는 전역 단위에서 사용하므로 항상 메모리에 올라가 있고, 캐릭터 모델 데이터는 특정 장면에서만 불러오도록 관리 방식이 분리되어 있어야 한다.

- **B-1-1-5.** 이전 장면에서 사용한 GUI들은 장면의 전환 시점이 아닌, **GUI 관리자가 필요로 하는 시점에서 메모리에서 해제**한다.
: 이는 장면 간 GUI 연출을 기획 의도에 맞출 수 있도록 유연하게 관리하고자 하기 위함이다.

※ 즉, 각 장면의 GUI들은 해당 장면과 관계에 의해 불러지지만, 장면을 구성하는 다른 객체들과 수명 자체를 같이하는 개념이 아니다.
예를 들자면, 이전 장면에서 다음 장면으로의 GUI 전환을 서서히 사라지고 서서히 드러나는 (Fade In / Fade Out) 방식으로 구현하고 싶다고 한다면, 이전 장면과 다음 장면의 GUI는 잠시 동안일지라도 같은 시점에 메모리에서 '공존'하고 있어야 한다.
만약, 장면의 GUI들이 다른 장면 객체들처럼, 다음 장면으로 전환될 때, 다음 장면의 객체들이 생성되기 직전에 '칼같이 해제된다면' 위와 같은 전환 효과를 연출하기는 매우 어려울 것이다.



<장면 전환 모델>

◆ B-1-2. 진입(Entry)

- **B-1-2-1.** 애플리케이션을 시작하면 가장 먼저 불러오는 장면이다.

- **B-1-2-2.** 이 단계에서는 이 게임 애플리케이션의 전체를 관리하는 전역 관리자만을 호출한다.

- **B-1-2-3.** 진입 장면의 존재 의의는, **게임 전역 관리자 객체를 확실하게 가장 먼저 호출하도록 보장**하기 위함이다.
: 진입 장면이 호출된 직후에, 전역 관리자에 의해 곧바로 다음 장면이 호출되기 때문에, 진입 장면 자체는 '순식간에' 지나가 버리며, 실제로도 뭔가 화면에 보여주는 역할 같은 것은 없다.

◆ B-1-3. 입장(Intro)

- **B-1-3-1.** 제작사와 발행사를 소개하는 로고가 등장하는 화면부터, 사용자가 게임을 켰을 때, 가장 처음 볼 수 있는 입장 화면까지 관리한다.

- **B-1-3-2.** 진입 장면이 호출된 뒤, 진입 장면의 전역 관리자에 의해 곧바로 호출되는 관계다.

- **B-1-3-3.** 실질적으로 사용자가 접하는 첫 화면을 관리한다.
  : 보통, 게임 제목과 화려한 배경 텍스처가 보이면서 'Game Start' 버튼이 깜박이고 있거나, 로그인 GUI 가 떠 있는 화면을 말한다.

- **B-1-3-4.** 캐릭터가 게임 시작 버튼 또는 계정 접속을 통해 서버로부터 로그인 인증을 받으면, 캐릭터 선택 / 생성 장면으로 넘어간다.

◆ B-1-4. 캐릭터 선택 / 생성

- **B-1-4-1.** 계정을 생성할 때, 사용자가 직접 계정 이름을 입력하지 않는 방식인 경우에는, 계정을 최초로 만든 사용자인 경우, 사용자의 별명을 먼저 생성해야 한다.
  : 별명을 생성하지 않은 계정은 다음 단계로 진행할 수 없다.
  사용자가 직접 계정 이름을 입력하는 방식인 경우에는 이 과정을 별도로 가질 필요가 없다.

※ 위에서 제시하는 두 가지 방식 모두, 서버 내부에서는 계정의 고유 번호를 부여하여 관리한다는 점에서는 동일하다. 어쨌든 서버는 사용자 계정을 고유 번호를 부여해서, 이 숫자를 통해서 인식한다.
그러나 사용자가 계정을 직접 입력하지 않는 경우에는 자동으로 할당되는 (아마도 열 자릿수가 넘을) 숫자를 기억해야 하기 때문에 직관적이지 않다. 또한 이런 숫자 코드 번호는 GUI를 통해 다른 사용자를 표시하기에도 적절한 방식이 아니기 때문에, '사람이 알아볼 수 있을만한' 식별 문자가 필요하다. 그런 이유 때문에 어떤 형식으로든 각 사용자 계정마다 '이름'을 부여해 달라고 요청을 하는 것이다.

- **B-1-4-2.** 캐릭터가 하나도 생성되지 않은 계정이라면, 캐릭터를 하나라도 생성하기 전에는 다음 단계로 넘어갈 수 없다.

- **B-1-4-3.** 모든 직업마다 캐릭터를 단 한 개씩만 생성할 수 있는 방식인 경우, 캐릭터의 이름을 별도로 지정해야 할 필요는 없다.
  : 각 캐릭터들을 구분하는 단위가 사실상 사용자 계정과 일치하기 때문이다.

※ 이와 같은 캐릭터 관리 방식을 쓰는 게임들 중에서, 일부 게임들은 게임 세계관의 고유한 캐릭터 이름을 사전에 부여하기도 한다.

- **B-1-4-4.** 캐릭터를 만들 수 있는 일정한 개수의 '캐릭터 슬롯'을 제공하는 방식인 경우, 각 캐릭터들은 고유한 이름을 정해야 한다.

: 그러한 각 캐릭터들은 인증 단위 내에서는 이름이 고유해야 한다.

※ 캐릭터 이름의 '고유함'을 어떻게 구현할지에 대해서는 다음과 같은 방식이 있다.
1) 전통적인(?) 방식
: 사용자가 입력한 이름 그대로 인증 단위 내에서 중복 검사를 하여 사용한다.
사람들이 잘 사용할 것 같은 인기 있는 이름들을 피해가면서 캐릭터 이름을 지어줘야 하는 불편이 따른다.

2) 사용자의 선택권을 넓히는 방식
: 쉽게 말해, '사용자의 계정 + 캐릭터 이름' 방식으로 인증 단위 내에서 중복 검사를 하여 사용하는 방식이다.
이렇게 하면 사용자들끼리 이름 겹침 스트레스를 피해서 캐릭터 별명을 지정할 수 있는 장점이 있다. 단점은, 같은 이름의 사용자를 구분하기 위한 장치가 필요하다는 점이다.

- **B-1-4-5.** 플레이어는 캐릭터를 하나라도 선택해야 로비 장면으로 진입할 수 있다.
: 즉, 로비 장면은 **플레이어가 최소한 1명의 캐릭터의 플레이 할 캐릭터를 선택했음을 전제**로 진행한다.

◆ B-1-5. 로비(Lobby)

- **B-1-5-1.** 게임 시작 버튼을 터치하거나, 사용자 계정 인증에 성공하는 등, 게임 시작 허가 절차를 거치면, 그 다음에 로비 장면이 호출된다.

- **B-1-5-2.** 로비 장면에서 할 수 있는 기능들, 그러니까 상점 이용, 친구 관리, 업적 보기, 랭킹 보기, 캐릭터 보관함 등의 기능을 단지 창 GUI의 전환을 통해서 표현할지, 아니면 별도의 로딩 화면을 거쳐서 전환하는 방식으로 개발 진행 상황에 따라 결정해야 한다.

※ 이게 무슨 말이냐 하면, 상점, 친구 관리, 업적, 랭킹, 캐릭터 보관함 등의 많은 로비 기능에 대한 데이터들을 한 번에 다 불러와서 메모리에 올려놓을 것인지, 아니면 기능별로 쪼갠 뒤, 사용자가 요청할 때마다 요청한 기능만 불러올 것인지 결정해야 한다는 뜻이다.

1) **한 번에 기능들을 다 불러오는 경우**
: 이 경우, 기능 간 전환 속도도 빠르고, 사용자에게도 비교적 전환 시간이 불편하게 느껴지지 않는 편이다.
다만, 기능이 많고, 그 기능들의 GUI 를 표현하기 위해 들어가는 리소스가 많을수록 한 번에 메모리에 올려야 하는 데이터도 늘어나기 때문에 문제가 된다. 구동하는 장치의 가용 메모리가 충분하지 않다면, 운영체제에 의해 강제 종료 되는 등의 문제가 생길 가능성이 높아진다.

2) **필요한 기능만 요청할 때마다 불러오는 경우**
: 이 경우에는 로비에서 제공하는 기능 집합이 많고, 그에 따른 리소스 데이터들의 크기가 커지더라도, 메모리 사용량을 효과적으로 억제할 수 있다. 정확하게 필요한 기능에 대한 장면 내용만 메모리에 올리고, 나머지는 메모리로부터 해제하기 때문이다.
단점은, 로비에서 사용자가 필요로 하는 기능을 제공하는 장면으로 이동할 때마다 [현재 사용하지 않는 기능 장면의 리소스 데이터를 메모리에서 삭제] -> [새로 이동한 기능 장면의 리소스 데이터를 저장장치로부터 메모리에 올리기] 과정을 거쳐야 하기 때문에, 기능간 전환 시간이 몹시 답답하게 느껴질 것이라는 점이다.
특히, 상점 -> 캐릭터 정보 -> 인벤토리 -> 상점 등 기능 간 전환을 자주 하는 사용자일수록 더욱 그렇게 느낄 것이다.

* **그 외 절충안**
: 2)의 기능을 기본으로 하되, 로비에서 다른 기능 장면을 불러오더라도, **기존에 한 번 불러온 (안 쓰게 되는)기능 리소스들을 제거하지 않고 남겨두는 기법**을 쓸 수도 있다.
이렇게 할 경우에는, 활성 메모리 사용량은 점점 늘어나겠지만, 사용자가 로비에서 기능 장면 전부를 돌아보지 않는 한, 로비 장면에서 써야 하는 최고 메모리 사용량까지 치고 올라가지는 않을 가능성이 높다.

또는, 이를 좀 더 응용해서, **미리 사용자가 가장 자주 쓸만한 기능 집합에 대한 장면 데이터들은 처음부터 로비 장면을 불러들일 때, 같이 메모리에 불러들이고, 나머지 '잘 안 쓰는' 기능 집합의 장면들은 사용할 때만 불러오게 하는 방법**도 쓸 수 있다.
예를 들자면, 대개 캐릭터 보관함과 캐릭터 정보, 아이템 / 스킬 / 펫 강화 창, 상점 이용은 매우 자주 쓰지만, 나머지 기능들은 그다지 자주 이용하지 않을 가능성이 높다.
이렇게 하면, 자주 쓰는 기능만 돌아다니는 사용자들은 GUI 의 전환 과정에서 별도의 로딩 화면을 볼 일이 거의 없으며, 잘 안 쓰는 기능을 사용하고자 할 때만 로딩 화면을 보게 되는데, 이 정도는 사용자로서도 충분히 감수할 만 할 수 있다.

◆ B-1-6. 로딩(Loading)

- **B-1-6-1.** 장면에서 장면을 이동하는 경우, 혹은 같은 장면 내에서도 많은 양의 리소스 로딩, 네트워크를 통한 내려 받기 등으로 인해 시간이 꽤 많이 소요되는 경우에는 로딩 장면으로 넘어가서 처리한다.

- **B-1-6-2.** 로딩 장면은 다른 장면과는 달리, 반드시 독점적으로 동작해야 할 필요가 없다.
: 로딩 장면은 장면과 장면 사이에서 공존한 상태로 존재할 수 있으며, 필요에 따라서는 그저 GUI 이벤트의 하나로써 구현할 수 있다.

※ 관념적으로는 '로딩 장면'이라고 하지만, 사실, 로딩이 게임 진행 프로세스에 있어서 독립적인 과정으로 존재할 수 없는 한, 일반적인 장면과는 다르게 취급할 수 밖에 없다.
장면 간 전환을 한다고 말하는 과정이란, 이전 장면에 사용했던 모든 객체와 데이터들을 메모리에서 해제하고, 다음 장면에서 사용할 객체와 리소스 데이터들을 메모리에 올리는 과정이다. 때문에 실제로 독립적인 진행 구조와 데이터를 가지지 못하는 로딩 과정에게 '장면'이라는 것은 따로 없는 셈이다.

- **B-1-6-3.** 로딩이 완료된 뒤에는, 장면 전환에 필요한 모든 데이터를 불러온 뒤, 내부 실행 논리에 의한 준비까지도 완전히 끝난 상태에서 로딩 장면에 대한 데이터들을 해제해야 한다.

- **B-1-6-4.** 로딩 장면의 GUI는, **시스템 알림을 위한 Modal 팝업을 제외하고는 어떤 GUI보다도 계층적으로 높은 우선 순위를 가져야** 한다.
: 즉, 로딩 장면의 GUI는 나머지 다른 장면의 GUI들을 '가려야' 한다.

※ 시스템 알림 팝업 GUI 가 예외인 이유는, 로딩 장면 중에도 시스템 적으로 문제가 생겼을 때, 이를 처리할 수 있게 하기 위해서다. 실행 도중 뭔가 문제가 생겨서 애플리케이션을 종료해야 하는데, 종료 알림 팝업 GUI 를 로딩 화면이 다 가려 버리고 있으면 문제가 된다.

- **B-1-6-5.** 로딩 장면에서 데이터를 불러들이는 부분과, 로딩 진행 정도를 표시하고 배경을 그리는 부분은 **비동기적으로 보이도록** 동작해야 한다.
: 데이터를 불러들이는 부분과 진행을 표시하는 부분이 결합되어 있으면, 데이터를 불러들일 때마다 장면 전체가 멈춘 것처럼 보일 것이다.

※ 이러한 기능이 '기술적으로' 비동기 방식으로 작동해야만 하는 의무는 없다.
로딩이 진행되는 정도를 주기적으로 화면과 GUI 를 통해 알리는 이유는, 사용자가 응용 프로그램의 동작이 멈췄다고 오해하지 않게 하기 위해서다. 내부적으로야 완전히 다른 기능이 봉쇄된 채로 로딩이 끝날 때까지 다른 어떤 동작도 할 수 없다고 한들, 사용자 입장에서는 불러오기 기능이 진행되고 있다고 느낄 수 있으면 아무런 문제가 없다.

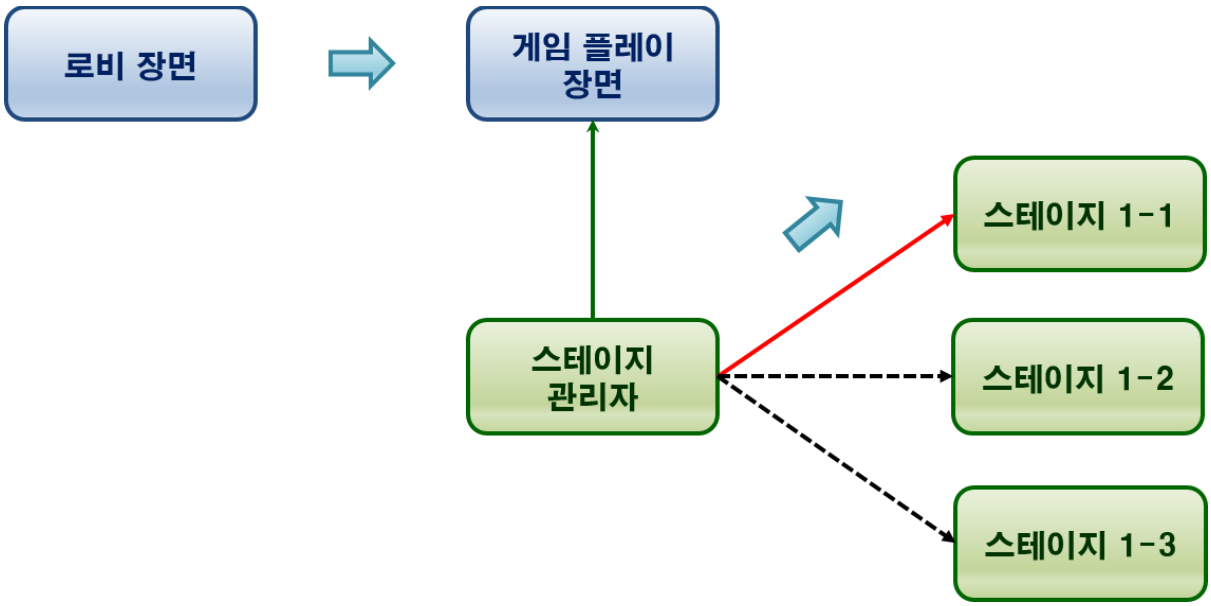
◆ B-1-7. 게임 플레이(GamePlay)

- **B-1-7-1.** 게임 플레이 장면은 사용자가 플레이 할 캐릭터와 장비 아이템, 스킬 등을 선택하고, 게임 모드 및 스테이지를 고른 뒤에 게임 플레이 시작 확인을 하는 순간에 불러온다.

- **B-1-7-2.** 게임 플레이 장면은 실제 플레이가 이루어지는 개별적인 스테이지 그 자체를 표현하지는 않으며, 그러한 실제 게임 플레이를 할 스테이지의 정보와 더불어 게임 플레이에 필요한 객체와 논리 정보들을 관리하는 역할을 한다.

※ 이는 Entry -> Intro & Lobby 장면 관계와 유사한 점이 많다.
장면 구조상에서의 게임 플레이 장면은, 물론, 게임 플레이에 필요한 각종 실행 논리들을 소유하고 있기는 하다. 하지만, 엄밀히 말해서 사용자에게 뭔가를 보여주는 역할은 하지 않는다.
사용자에게 뭔가를 보여주기 위한 계통 구조와 정보들은 스테이지 데이터들이 담당하고 있다.

게임 플레이 장면이 하는 일은, 사전에 규격화된 방식으로, 현재 필요한 스테이지 데이터를 불러와서 게임 플레이를 '보여줄 수 있게' 관리하는 임무를 가진다.



<게임 플레이 장면과 스테이지의 관계>

- B-1-7-3. 게임 플레이 중에 스테이지는 동시에 1개만 돌아간다.
: 여러 개의 스테이지를 동시에 불러와서 실행하지 않는다.

※ 여기에는 아래와 같은 두 가지 이유가 있다.

1) **굳이 여러 개의 스테이지 데이터가 동시에 등장해야 할 이유가 없는 게임**이다.
지형 맵이 극단적으로 넓은 MMORPG 같은 게임이 아니고, 스테이지와 스테이지는 별도의 로딩 과정을 거쳐도 되는 게임이므로, 복잡하게 여러 개의 스테이지를 동시에 띄우고, 스테이지 관리자를 둘 필요가 없다.

2) **개발 환경인 Unity3D 에서 이를 손쉽게 구현하기에 장애가 되는 문제들이 존재**한다.
지형 Mesh 와 재질, 텍스처 정보들은 손쉽게 같은 게임 공간에 여러 개를 추가해서 동시에 올릴 수 있지만, 조명맵(Lightmap), Occlusion Culling, 길 찾기 메쉬(Navigation Mesh) 같은 정보는 그럴 수가 없다. 아예 불가능한 것은 아니지만, 구현하기에는 꽤 성가신 문제들 중 하나이다.

◆ B-1-8. 퇴장(Outro)

- **B-1-8-1.** 애플리케이션이 종료되기 직전에 수행해야 하는 이벤트를 담당하는 장면이다.

※ 예를 들자면, 마케팅 상의 이유로 인해 종료 직전에 개발사 / 발행사의 로고를 한 번 더 노출을 한다거나, 광고를 둘 수도 있을 가능성이 있다.

- **B-1-8-2.** 사실, 이와 같은 기능이 꼭 필요한지는 모르겠다. 그냥 아무 것도 안 하고 바로 종료해도 상관없다면, 굳이 이 장면을 거치는 과정을 만들 이유는 없다.

※ 어쩌면, 이런 기능이 대부분의 앱에서 보이지 않는 이유는 마켓의 출시 정책에 위배되기 때문일 수도 있다. 이 부분에 대해서 확인이 필요하다.

◆ B-1-9. 전역 관리 객체들

- **B-1-9-1.** **게임 전역 관리자(장면 관리자, Game Manager)**
  : 게임 애플리케이션이 시작하면 가장 먼저 생성되는 객체이며, 게임 전체를 실행시키는 시작점이자, 게임 애플리케이션이 종료될 때는 가장 마지막에 해제되는 객체이다.

- **B-1-9-2.** **게임 데이터 관리자(Game Data Manager)**
  : 게임 애플리케이션 구동 전체적으로 영향을 미치는 각종 전역 값들을 관리한다.

※ 대표적으로 게임 실행 옵션 데이터들을 들 수 있다. 내부적으로는 각종 전역 변수들을 정의하기도 한다.

- **B-1-9-3.** **네트워크 관리자(Network Manager)**
  : 클라이언트 – 서버 간 연결 정보를 담당하는 관리자이다.

- **B-1-9-4.** **게임 리소스 정보 관리자(Game Resource Information Manager)**
  : 게임에서 사용할 리소스 데이터들을 어떻게 찾아서 불러올지에 대한 정보들을 담고 있다.
  이 정보들은 텍스트 기반 데이터들로부터 불러오며, 게임 애플리케이션이 실행되면 가장 먼저 불러오는 데이터 중 하나이다.

※ 게임은 이 텍스트 기반 데이터들로부터 비롯된 연결 정보들을 가지고 있다가 필요한 시점에 필요한 리소스 데이터들을 저장 장치로부터 불러들일 수 있다.

- **B-1-9-5.** **내 정보 관리자(My Information Manager)**

: 게임 애플리케이션이 시작된 직후부터, 접속한 사용자에 대한 정보를 관리한다.

- B-1-9-6. 이벤트 관리자(Event Manager)

: 게임 애플리케이션을 실행할 때 발생하는 이벤트 논리들의 실행을 관리한다.
즉시 실행하는 이벤트, 얼마 간의 대기 시간 이후에 실행하는 이벤트 등을 모두 이벤트 관리자를 통해 주고받는다.

B-2. 스테이지 구조

◆ B-2-1. 구성 요소

- B-2-1-1. 스테이지 구역 - 지형 맵(Stage Area - Terrain Map)
: 사용자 캐릭터가 돌아다니는 세계를 표현한다.

- B-2-1-2. 조명 정보(Light Object)
: 미리 구워진 Light Map 정보, 또는 Light Probe 정보들을 말한다.

- B-2-1-3. 길 찾기 데이터(Navigation Datas)
: Navigation Mesh 데이터와, 게임 플레이 도중에 반드시 거쳐 가야 하는 특정한 지점들을 표시하는 좌표 데이터들의 목록을 말한다.

※ Unity 에서 지형 모델 데이터의 경우에는 장면 파일(*.unity)을 더하는 방식으로 불러들이는 기법을 이용해 여러 개의 장면 파일로 하나의 큰 지형 모델을 구성하는 방식도 가능하다.
그러나 조명 정보들과 길 찾기 데이터들은 그런 방식으로 불러올 수 없으며, 무조건 특정한 장면으로 완전히 전환해야만 한다.
스테이지 1 개 = 맵 1 개로 제한하는 가장 큰 이유 중 하나다. (이런 상황을 회피하는 게 불가능한 건 아니지만, 리소스 제작 및 장면 구성 프로세스가 매우 복잡해진다.)

- B-2-1-4. 트리거 데이터(Trigger Datas)
: 물리쳐야 할 몬스터들이 맵 어디에 배치되어 있는지에 대한 정보라든가, 특정한 위치에 도달했을 때 발생하는 이벤트 정보들을 가지고 있다.

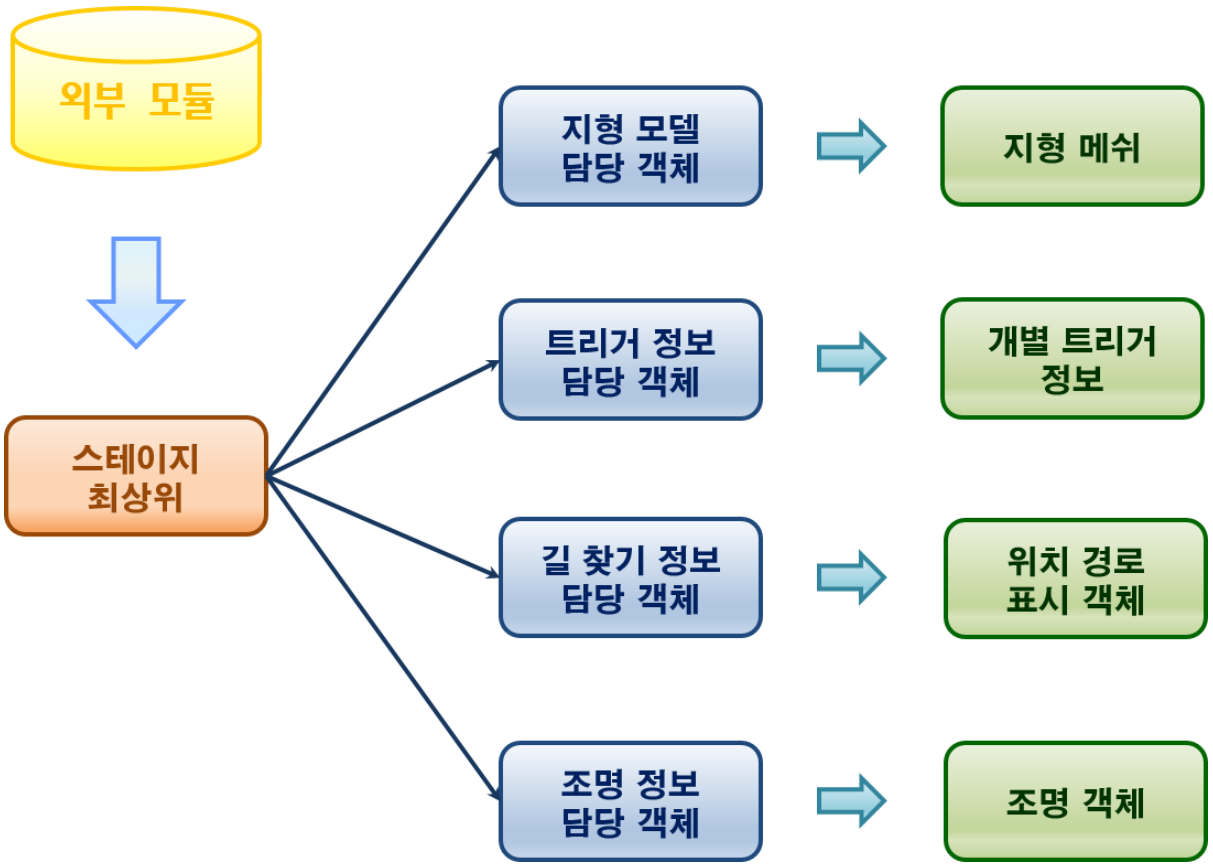
- B-2-1-5. 미션 데이터(Mission Data)
: 스테이지에 진입하는 목적을 표현하는 정보들의 집합이다. (스테이지에 들어가는 이유가 있어야 할 거 아닌가?)

◆ B-2-2. 계통 구조

- B-2-2-1. 최상위 객체(Root Object)
: 각 스테이지는 장면 그래프의 최상위에 하나의 객체를 가진다.

그리고 스테이지의 세부적인 정보들을 표현하는 객체들은, 그러한 **객체들의 세부적인 계통 구조에 관계없이 어떤 단계로든 모두 장면 그래프 최상위 객체의 하위에 소속**된다. ()
이렇게 해서 특정 장면에 대한 접근 통로를 일원화한다.

- **B-2-2-2.** **각 종류별 데이터 셋에 접근하기 위한 관리용 객체(Linker Object, Router Object)**
: 지형 맵의 모델 정보나 트리거 정보, 길 찾기 경로 정보에 접근하기 위해서는 각 정보의 종류에 맞는 관리자 객체를 통해서 접근해야 한다.
(이는 스테이지의 각 정보들에 접근할 때, 그 스테이지 장면 파일의 최상위 객체를 통해 접근하는 원리와 같다.)



<스테이지 객체들의 계통 구조>

◆ B-2-3. 생성과 해제

- **B-2-3-1.** 각 스테이지들은 스테이지의 세부적인 데이터를 담고 있는 별도의 장면 파일로 관리된다.
: 위에서 설명한 리소스 모델 데이터, 트리거 배치 정보, 조명 설정 정보 등을 말한다.

- **B-2-3-2.** 각 스테이지들은, 그 스테이지를 나타내는 고유한 식별 번호가 있어, 이 값을 통해

구분을 한다.

- **B-2-3-3.** 게임에서는 **한 번에 하나의 스테이지만 불러올 수 있다.**
: 여러 개의 스테이지 정보들을 불러들여서 동시에 메모리에 유지하지 않는다.

- **B-2-3-4.** 스테이지에서 빠져 나오거나 다른 스테이지로 전환할 때는, 해당 스테이지를 표현하기 위해 생성한 모든 데이터를 전부 메모리에서 제거한다.
: 즉, 스테이지 데이터가 삭제될 때는 게임 플레이 장면 자체도 종료하는 것으로 간주한다.

※ Unity 에서는 *.unity 파일을 불러올 때, 그 장면 파일에 포함된 조명맵(Lightmap) 정보나 길찾기 메쉬(Navigation Mesh) 정보 등도 같이 불러온다.
이 정보들은 다른 장면 파일들로 전환하기 전까지는 메모리에서 사라지지 않는다. 단순히 지형 메쉬 정보와 게임에서 사용한 객체 정보만 제거한다고 해서, 해당 스테이지의 모든 정보가 '완전히' 초기화되는 게 아니다.

◆ B-2-4. 스테이지의 구역(Stage Area)

- **B-2-4-1.** 때로는 디자인 상 이유로 인해, 아주 넓은 스테이지와 많은 개체 / 트리거 배치 정보를 하나의 스테이지에 포함하고 싶을 수가 있다.
: 이러한 경우, **각 스테이지들은 구역 별로 쪼개어 만들어야 한다.**

※ 구역을 나누는 기준은, 나눈 각 구역의 지형 정보 및 개체 / 트리거 배치 정보 등이 성능 요구사항에 부합하는지 여부와, 게임 디자인 상 이유를 모두 고려한다.

- **B-2-4-2.** 스테이지의 구역(Stage Area)을 구성하는 지형 모델은 반드시 한 번에 불러오는 단위여야 한다. 즉, **스테이지 구역과 지형 모델은 1 : 1의 관계**여야 한다.
: 여러 개로 쪼개어진 지형 모델을 합쳐서 사용할 수 없다.

- **B-2-4-3.** 스테이지를 플레이 할 때는, **한 번에 하나의 스테이지 구역만 불러와야 한다.**
: 여러 개의 스테이지 구역을 한꺼번에 불러와서 합치는 방식으로는 사용할 수 없다.

※ 즉, 게임 논리상으로는, 스테이지에 속한 여러 개의 스테이지 구역을 모두 다 돌파해야 그 스테이지 1 개를 완료한 것으로 간주한다.
그렇지만, 게임 플레이 장면에서는 한 번에 스테이지 구역 1 개 단위로 불러오면서 게임을 진행하는 셈이다.

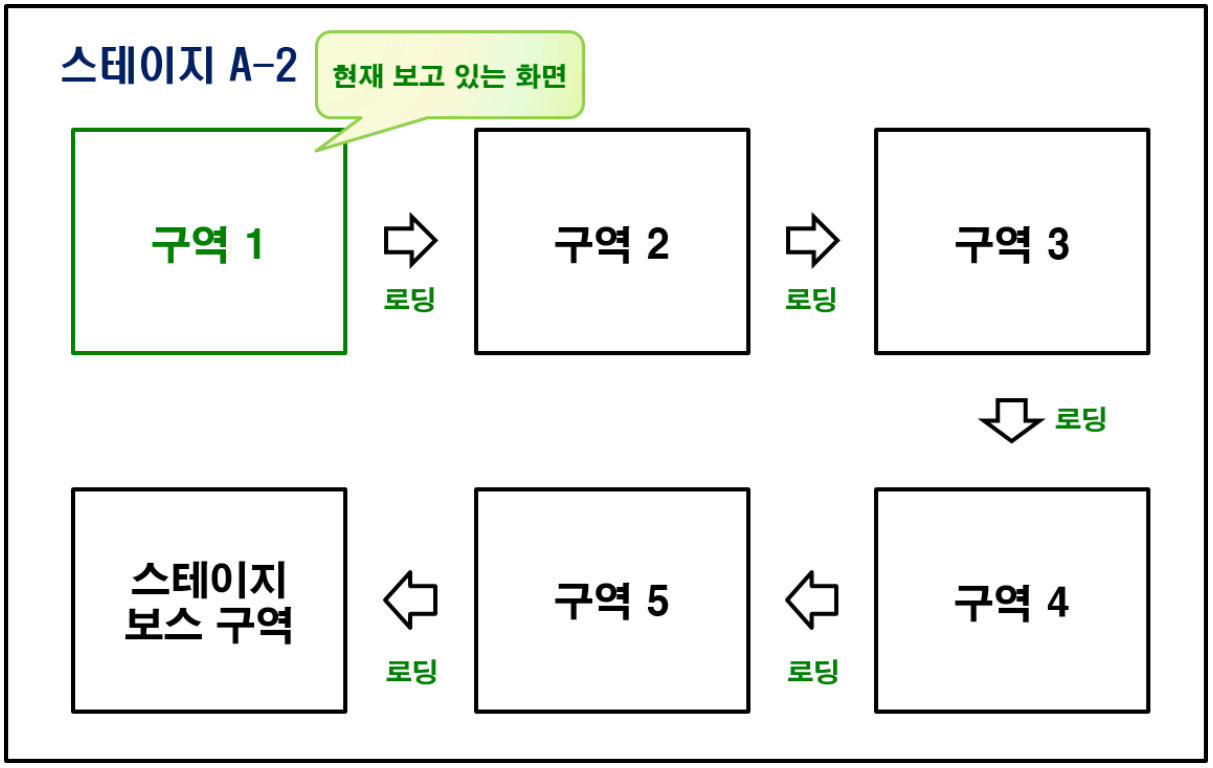
- **B-2-4-4.** 어느 하나의 스테이지 구역(맵 하나)에서 다른 스테이지 구역을 통과하고자 한다면, 로딩 과정을 거쳐야 한다.

: 즉, 전술한 바와 같이 스테이지 구역 정보(맵 정보)들도 동시에 2개 이상 메모리에 올려두고 유지할 수 없기 때문에, 스테이지 구역끼리 넘나들 때도 로딩 화면을 보는 과정을 거쳐야 한다.

※ 스테이지 구역 1개만을 동시에 유지하도록 제한하는 이유의 대부분은 **성능상의 이유**이다.
만약, 이전 스테이지로 되돌아갈 수 있어야 한다면, 게임 관리자는 해당 스테이지의 모든 구역과 모든 맵에 대한 정보를 유지한 상태로 있어야 한다. 왜냐하면, 이미 지나간 스테이지에 몬스터가 몇 마리 남아 있고, 그 몬스터들의 체력 상태가 어떤지, 살았는지, 죽었는지, 현재 위치가 어떤지, 부여효과가 걸려 있는지 아닌지 등의 여부를 모두 유지해야만 '이전 스테이지로 되돌아갔다.'고 할 수 있기 때문이다.
더구나, 이렇게 구현한다면, 굳이 스테이지 구역마다 로딩하는 과정도 별 필요가 없다. (어차피 메모리에 들고 있으니...)

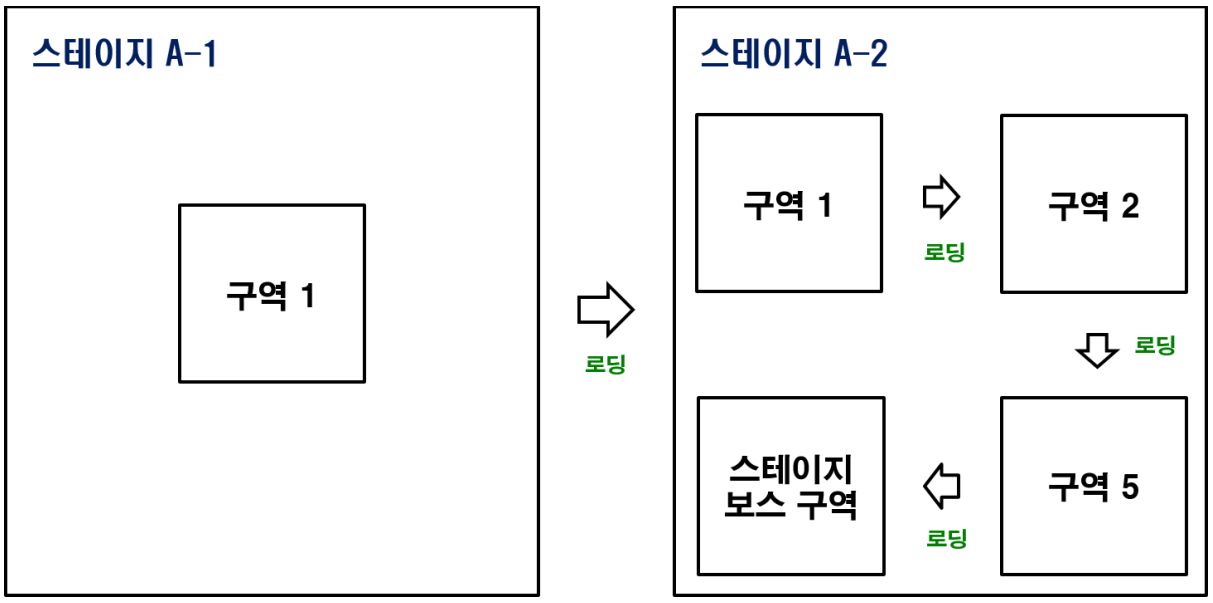
당연하게도, 이런 모든 정보들(모든 스테이지 영역의 캐릭터 모델과 데이터, 모든 스테이지 영역의 지형 모델 데이터)을 유지하는 비용은 아주 클 수 밖에 없다. 모바일 게임의 특성상 가용 메모리는 제한적이기 때문에, 모델링 데이터까지 한꺼번에 유지하기는 무리이다. (사실, 이는 PC라고 해도 다르지 않다.)
그럼에도 불구하고 이를 구현하고자 한다면, 모델링 데이터와 관련이 없는 별도의 지형 정보를 이용해서 위치 정보를 다룰 수 있어야 한다. (캐릭터 데이터나 이펙트 데이터의 경우에도 마찬가지겠지만, 캐릭터의 경우에는 이미 모델링 데이터와 관계없이 상호작용할 수 있는 시스템을 구축하도록 되어 있다.)
Unity 엔진에서 이러한 기능을 직접 제공해주지는 않으므로, 엔진과 연동할 수 있는 자체적인 지형 정보 시스템을 구현하기 위한 비용은 막대하리라고 예상한다.
이러한 내용들은 끊김 없는(seamless) 지형 로딩이 필수적인 게임들(MMORPG라든가...)에는 핵심 사항이겠지만, 적어도 이 게임에서는 그렇지 않다. 이 게임은 끊김이 없이 로딩되는 넓은 지형과 세계를 보여주는 점을 게임의 핵심적인 재미 요소로 고려하는 건 아니다.

그러므로, 구현 난이도와 비용에 비추어, 스테이지의 구역은 한 번에 하나씩 유지하며, 한 번 벗어난 스테이지 구역의 정보는 완전히 제거되어 같은 게임을 진행하는 도중에는 다시 이용할 수 없게 하는 편이 적절하다고 본다.



<하나의 큰 스테이지의 구역 별 표현>

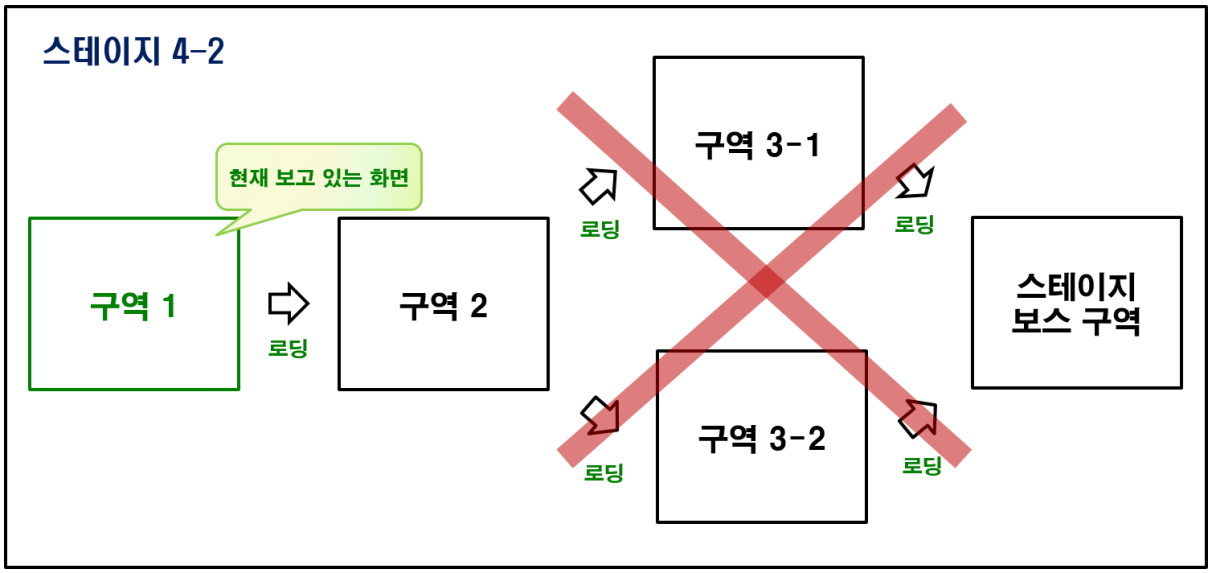
- **B-2-4-5.** 스테이지 -> 스테이지 전환 과정과 스테이지 구역 -> 스테이지 구역 전환 과정은 매우 비슷해서 혼동이 올 수 있을 것이다.
: 기술적인 구현은 거의 동일하며, 단지 게임 논리상으로 스테이지를 완료했는지 아닌지에 대한 차이점만 있다.



<뭐가 됐든, 로딩한다는 점에서는 똑같다>

- **B-2-4-6.** 스테이지 구역에서 다음 스테이지 구역으로 넘어가는 과정은 순차적이어야 하며, 스

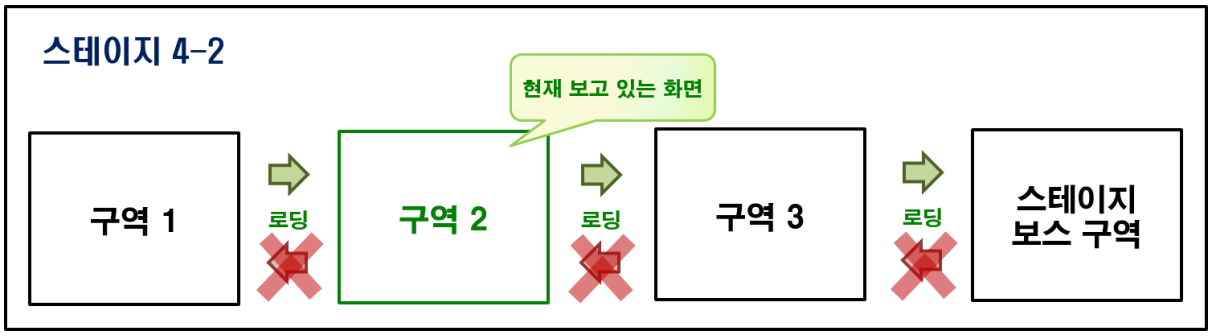
테이지 영역에 대해 분기를 둘 수 없다.



<스테이지 구역은 선택지를 만들 수 없다.>

- B-2-4-7. **한 번 다음 스테이지 구역으로 넘어가면, 이전 스테이지 구역으로 되돌아갈 수 없다.**
: 이전 스테이지 영역으로 되돌아가기 위해서는 이전에 플레이 했던 스테이지의 정보가 고스란히 남아 있어야 하는데, 한 번에 하나의 스테이지 구역의 정보와 지형 맵을 유지하도록 되어 있는 시스템에서는 그렇게 할 수 없다.

※ 스테이지 영역을 벗어나면, 벗어난 스테이지 영역의 정보는 아예 메모리에서 지워진다. 만약, 이전 스테이지 영역으로 되돌아가고자 한다면, 그건 사실상 새로운 스테이지를 불러오는 것과 다름이 없는 셈이다.
이는 미션 / 퀘스트의 진행 조건을 꼬이게 할 수 있기 때문에 허용하지 않는다.



<들어올 때는 마음대로였겠지만, 나갈 때는 아니란다.>

※ 이 사항에 대한 결정은 잠시 보류한다.
왜냐하면, 캐릭터 개체들의 관리 방식에 따라, '뒤로 되돌아가기'는 가능할 수 있기 때문이다.
가장 많은 실행 메모리를 차지하는 데이터는 지형 맵 정보와 생성된 캐릭터 개체들의 모델링 정보들이기 때문이다. 그러한 리소스 데이터들과 직접 관련이 있는 정보들을 빼놓을 수 있다면,

훨씬 더 적은 메모리를 가지고서도, 전체 스테이지에 대한 캐릭터 개체들의 정보를 관리할 수 있다.
또한, 현재 플레이 중인 구역이 아닌 스테이지 구역에 대해서는 몬스터들의 위치 변환이나 인공지능 수행 등을 전혀 할 필요가 없기 때문에, 캐릭터 관리자는 현재 스테이지의 다른 구역에 대해서는 단지 그 구역 몬스터들의 마지막 좌표 지점과 마지막 상태값 정보 등만 가지고 있으면 그대로 복원이 가능하다.

◆ B-2-5. 스테이지 블록(Stage Block)

- **B-2-5-1.** 하나의 스테이지 구역에서, 플레이어 캐릭터가 장소를 옮겨 다닐 때마다 동적으로 생성되는 영역이다.

- **B-2-5-2.** 한 번의 전투가 일어나는 단위가 되는 영역

- **B-2-5-3.** 한 블록의 전투가 끝나기 전에는 다음 블록으로 넘어갈 수 없게 막혀 있다.

- **B-2-5-4.** 전투를 완료하면 막혔던 길이 뚫리면서 다음 블록으로 진행할 수 있다.

- **B-2-5-5.** 실질적인 구현은 트리거 영역에 들어섰는지 여부와, 특정 영역 내의 전투가 완전히 종료되었는지 여부로 판정하는 방식으로 구현하면 될 듯...

◆ B-2-6. 마을 스테이지

- **B-2-6-1.** 마을의 형태로 표현한다.

◆ B-2-7. 스토리 용 스테이지

- **B-2-7-1.**

◆ B-2-8. 혼돈 던전 스테이지

- **B-2-8-1.**

◆ B-2-9. 초월 모드 스테이지

- B-2-9-1.

◆ B-2-10. P vs P 스테이지

- B-2-10-1.

◆ B-2-11. 길드 전쟁 스테이지

- B-2-11-1.

◆ B-2-12. 월드 보스 공략전 스테이지

- B-2-12-1.

B-3. 객체 관리

◆ B-3-1. 객체의 분류

※ 이 항목에서는 게임 시스템을 구현하기 위해 필수적인 객체들만 간략하게 개요로써 다루고 있다. 가장 정확하고 상세한 내용은 문서의 각 세부 항목들과, 설계 및 인터페이스를 참고해야 한다.

- B-3-1-1. 관리자(Manager)

: 관리자 객체들은 여러 개의 객체 인스턴스들에 대한 접근과 생성, 삭제를 담당한다.
따라서, 관리자 객체들은 프로그램 내에서 단 하나의 인스턴스만 생성되며, 전역적으로 접근할 수 있는 인터페이스를 제공한다. (싱글톤 인터페이스)

- B-3-1-2. 장면(Scene)

: 각 장면마다 그 장면의 시작과 끝을 담당하는 Scene 객체들이 존재한다.
예를 들면, 게임 플레이 장면은 GamePlayScene 객체에서 최초의 실행 함수가 호출되고, 이 객체가 파괴될 때 종료된다.

- B-3-1-3. 개체(Entity)

: 게임 내 캐릭터 개체 하나를 표현하는 객체이다.

- B-3-1-4. 개체 컴포넌트(Entity Component)

: 게임 내 캐릭터 개체들은 실로 다양한 특질을 가진 객체들이 필요하다. 이들은 담당하는 특질마다 각각 Component로 구분한 객체로 표현한다.
이러한 Component들은, 그 컴포넌트들을 사용하는 주체가 되는 객체들이 조립해서 사용한다.

- B-3-1-5. 트리거(Trigger)

: 트리거는 특정한 조건들을 만족하면 결과 이벤트를 발동하게 하는 객체이다.
이벤트의 주체가 명확하지 않거나, 전역적으로 조건을 감시해야 하거나, 조건이 복합적이어서 이벤트의 소유자를 지정하기 어려운 경우에 사용한다.

- B-3-1-6. 트리거 컴포넌트(Trigger Component)

: 트리거의 컴포넌트들은 트리거의 조건과 판단, 실행 부분을 실질적으로 수행한다.
각 트리거들은 조건이 여러 개일 수 있고, 판단하는 부분과 이벤트 실행하는 부분도 마찬가지로 여러 개일 수 있다.

- **B-3-1-7. 메시지(Message)**
: 메시지 객체들은 메시지 수신자들이 받을 수 있는 형태로 정보를 가공해서 중계한다.
특정 객체를 찾아서 직접 정보를 전달하는 게 아닌, 허공에 뿌리고서 적절한 객체가 나타나서 찾아가기를 기다리는 방식이라고 할 수 있다.

- **B-3-1-8. 메시지 수신자(Message Listener)**
: 메시지 수신자들은 메시지 객체들이 중계될 때마다 그것을 수신할 수 있다.
메시지 수신자들은 그들에게 들어오는 모든 메시지를 처리해야만 하는 의무가 없다. 그저 자기 자신에게 처리가 필요한 메시지를 발견하면, 그 메시지만 처리하면 된다.

※ 사실, Unity 엔진의 내부 객체들은 위에 나열한 객체들의 특성들을 이미 지니고 있거나 외부 인터페이스로 사용하도록 노출한 경우가 많다. 그럼에도 불구하고 도구들을 다시 발명(?) 하는 데는 이유가 있다.

1) 성능 문제
: UnityEngine.GameObject.GetComponent<T>() 함수들은 **실행 중에 계속해서 호출하면 성능에 악영향**을 미치며, 이는 공식 문서에서도 경고하고 있다.
유니티 엔진에 내장된 컴포넌트 기능들은, 사용자가 기존에 없는 타입의 컴포넌트를 일관된 인터페이스로 추가하거나 접근할 수 있게 해주는 데 편리하다. 그러나, 이미 타입을 알고 있는 종류의 컴포넌트를 빠르게 접근하는 용도는 아닌 셈이다.
엔진 내부의 객체와 컴포넌트 설계는 이와 유사한 문제가 존재한다. 범용성에 있어 매우 뛰어나지만, 사용자가 미리 정의한 범위 내의 문제에 대해 빠른 속도를 제공하는 방법도 필요하다.

2) 운용 적합성
: UnityEngine.GameObject같은 엔진 내부의 기본 객체들은 매우 일반화된 사용을 가정하고 있기 때문에, 특정 컨텐츠에 특화된 기능을 표현하거나 접근하기 위해서는, 추가적인 코드가 많이 필요하다. 또한, 엔진 내장 객체들은 사용자들의 확장을 수용할 수 있도록 하기 위해, 범용성 대신 속도를 희생한 기능들이 많다. (미리 저장하지 않는 컴포넌트, 문자열 기반으로 함수를 찾아서 메시지를 전달하는 기능 등...)
결국, 객체의 성격을 손쉽게 구분하고 구현 생산성에 도움이 된다면, 개념적으로 옳더라도 과도하게 추상화된 객체보다는, 설계상 개념이 좀 겹치는 부분이 있더라도, 실제 구현할 내용에 더 가깝게 구체화된 객체들로 분류하는 게 더 낫다고 판단했다.

◆ B-3-2. 객체 수명주기 관리의 목표

- **B-3-2-1.** 게임 플레이 도중에 끊김 현상이나 초당 프레임 수(FPS, Frame Per Second)의 하락을 줄이는 게 가장 우선되는 목표이다.

- **B-3-2-2.** 활성 메모리 사용량을 줄인다.
: 모바일 기기의 활성 메모리는 출시된 지 오래된 기기일수록 심하다. 다양한 기기 시장을 확보하기 위해서는 가급적 한 번에 사용하는 메모리의 양을 줄일 필요가 있다.

- **B-3-2-3.** 불가피한 경우를 제외하면, **메모리 사용량보다는 빠르게 처리할 수 있는 객체 관리 방식을 쓴다.**

※ 현실적으로, 모바일 기기의 메모리 용량은 점점 커지고 있기 때문에, 이미 처리 시간에 대한 비용 위험이 메모리 캐싱에 들어가는 용량에 대한 비용보다 저렴한 경우는 거의 없다.

◆ B-3-3. 객체 관리 전략

- **B-3-3-1.** 원칙적으로, 게임의 실행 도중에는 힙(Heap) 메모리 할당을 피한다.
: 경험상, 게임 플레이의 업데이트 도중에 힙 메모리에 대한 할당이 발생할 때, 끊김 현상이나 FPS 하락을 많이 느낀다.

- **B-3-3-2.** 할당이 필요한 경우에는 가급적 로딩 장면 단계에서 모두 처리한다.
: 로딩 장면 단계에서는 사용자들도 기다려야 하는 시간임을 알기 때문에, 하드웨어에 의한 시간이 오래 걸리는 작업들은 가급적 이 시점에서 처리한다.

- **B-3-3-3.** 인스턴스로 생성한 객체들은 모두 관리자를 통해 관리한다.
: 인스턴스 화할 객체에 대한 모든 생성, 삭제에 대한 사항을 그 인스턴스 객체 타입을 관리하는 관리자 객체에 위임하고, 관리자 객체를 통해서 호출한다.

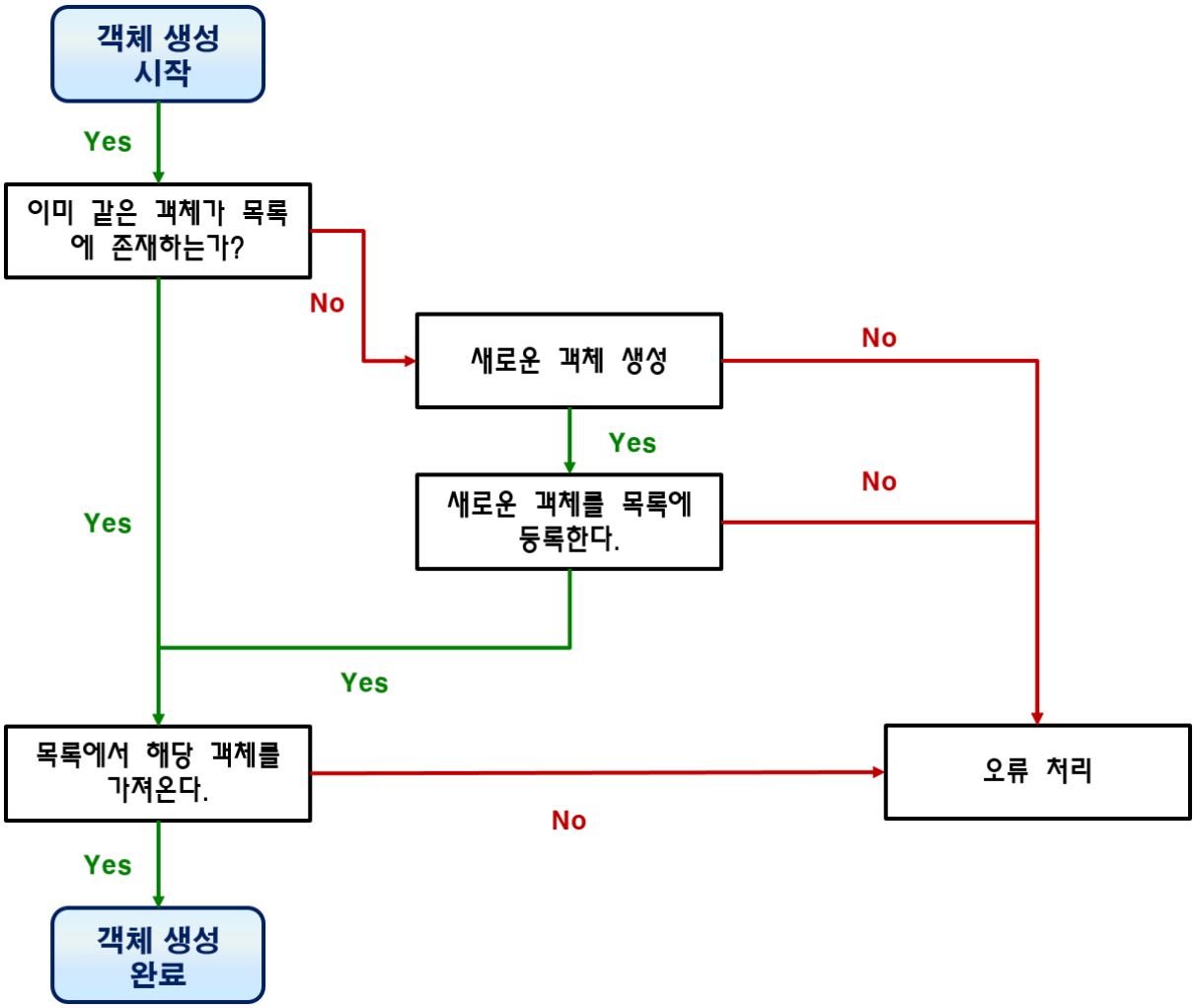
- **B-3-3-4.** 인스턴스 화 하는 객체들은 일단 생성되고 나면, **가급적 재활용해서 반복적으로 사용**한다.
: 객체 관리상 판단에 의하기 전까지는, 엔진 내부 프레임워크의 임의 판단에 의해 메모리에서 해제되지 않게 한다.'

※ NULL 참조 대입 등으로 인해, 메모리 쓰레기 수집(Garbage Collect)에 의해 탐지된 뒤에 메모리에서 지워지는 현상을 말한다.

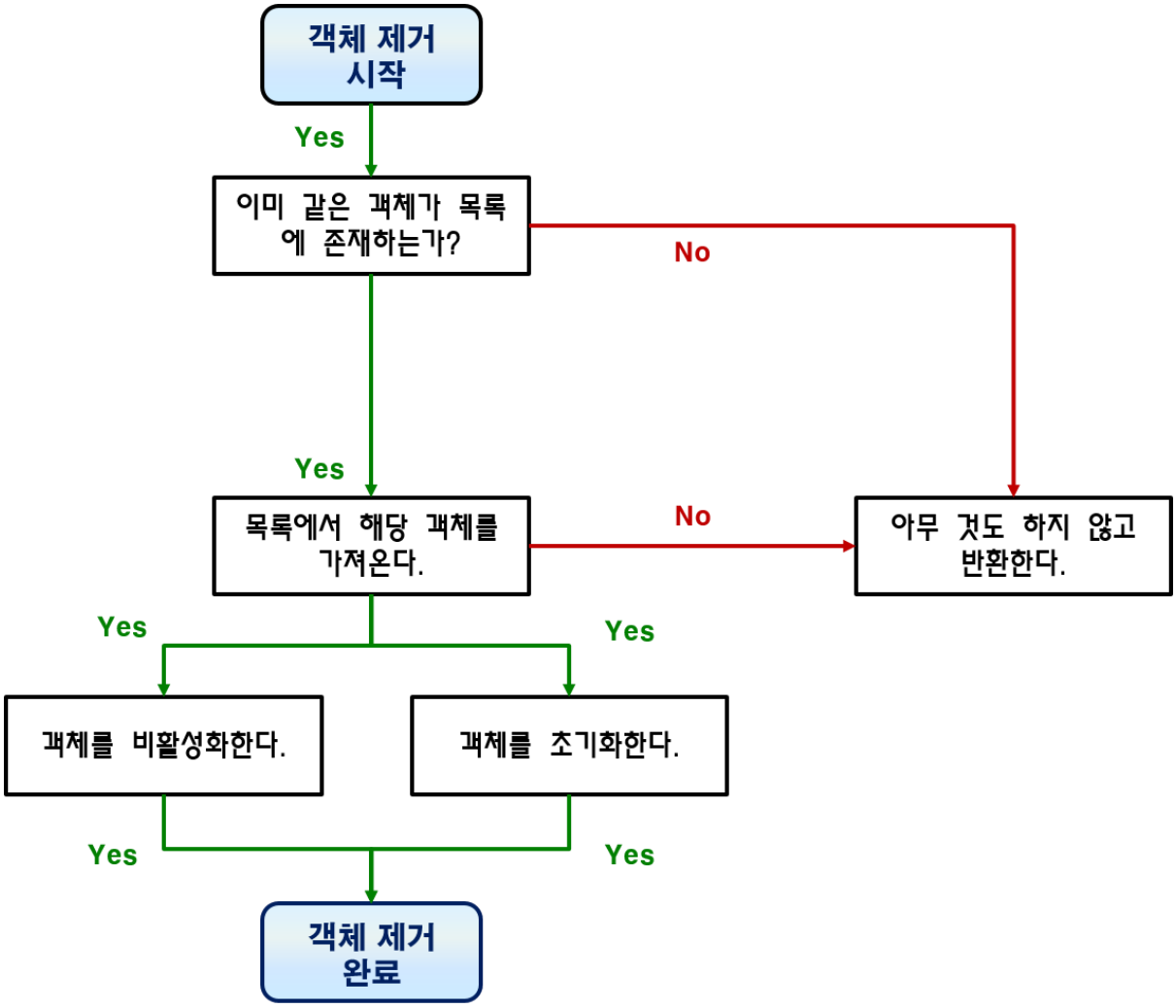
- **B-3-3-5.** 인스턴스 화 하는 객체들은 고유한 활성 / 비활성 함수들을 보유한다.

- **B-3-3-6.** 인스턴스로 생성한 객체가 '사라져야' 하는 경우에는, 실제로 객체를 메모리에서 제거하는 대신, 비활성화해서 '보이지 않게' 한다.
마찬가지로, 사라졌던 객체가 다시 생성되어야 하는 시점에서는 비활성화했던 객체를 활성화

하여 '살아나게' 한다.



<인스턴스 객체의 '생성' 프로세스>



<객체의 '제거' 프로세스>

- **B-3-3-7.** 물론, 인스턴스 객체가 허용 기준보다 많이 쌓였거나, 실제로 사용하지 않는 시간이 허용 기준보다 길다면, 이를 점검해서 메모리로부터 제거하는 정책을 사용할 수 있다.
  : 관련 기준은 필요한 시점에 정한다. 현재는 이러한 객체 관리 기능까지는 필요하지 않다고 생각하기 때문에 기준을 정하지 않는다.

B-4. 게임 내 메시지 프로세스

◆ B-4-1. 메시지 프로세스에 사용하는 객체들

- **B-4-1-1. 메시지 관리자(Message Manager)**
: 메시지 관리자는 게임 플레이 도중에 여러 캐릭터 개체 혹은 트리거 등의 인스턴스들이 만들어내서 뿌리는 메시지 객체들을 수집해서 모아두었다가, 올바른 수신자에게 전달하는 역할을 한다.

※ 기술적으로 엄격하게 따지자면, 메시지 프로세스는 반드시 게임 플레이 장면일 때만 동작해야 하는 법은 없다.

- **B-4-1-2. 메시지 수신자(Message Listener)**
: 만약 어떤 객체가 메시지를 수신하고 싶으면, 메시지 수신자 인터페이스를 정의하고 구현해야 한다. 그리고 나서, 구현한 메시지 수신자를 메시지 관리자에 등록하면 게임 플레이 관련 메시지들을 수신할 수 있다.

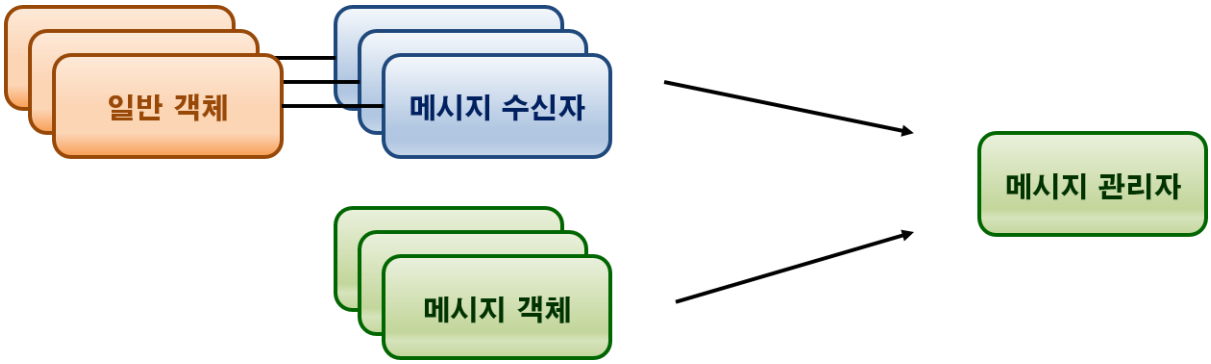
※ 이런 방식의 메시지 처리 기반 구조는, Unity 엔진 내부에서 메시지 처리 방식으로 사용하는 SendMessage() 함수 등으로 이루어지는 메시지 시스템과는 아예 별개로 돌아간다.

- **B-4-1-3. 메시지 객체(Message Object)**
: 게임 상에 어떤 대상, 또는 월드 전체에게 특정한 동작을 하도록 요구하고 싶은 경우, 그러한 동작을 위한 루틴을 호출해야 할 것이다.
이러한 호출 동작은 요구 시점에서 직접 찾아서 하는 방법도 있겠지만, 누군가에게 대신하도록 전달하고 넘어가고 싶은 경우도 있다. 특히 '몇 초 뒤에 실행' 같은 경우에는 더욱 그렇다.
메시지 객체는 이렇게 **다른 누군가로부터 전달받은 이벤트 동작에 대한 내용들을 가지고 있다가, 정확한 시점에 정확한 대상에게 전달하는 역할**을 한다. 그러니까, 말하자면 일종의 심부름꾼 객체이다.

◆ B-4-2. 메시지 객체 구조

- **B-4-2-1.** 메시지 관리자는 모든 메시지 수신자 목록과, 현재 게임에서 생성된 게임 관련 메시

지 객체들의 모든 목록을 받아놓고 통제한다.



<게임 메시지에 관한 모든 길은 메시지 관리자로 통한다.>

- **B-4-2-2.** 매우 당연하게도, 그렇기 때문에 메시지 관리자는 게임 안에서 유일하게 존재한다.

- **B-4-2-3.** 메시지 객체에 전달할 매개변수들을 구현할 때는, 기본 메시지 객체를 상속한 자식 메시지 객체에 멤버 변수로 지정하여 구현한다.
: 즉, 공통 타입으로 사용할 메시지 객체가 존재하고, 세부적인 매개변수들은 공통 타입 메시지 객체를 상속받은 자식 메시지 객체가 구현한다.

※ 즉, 의사 코드로 표현하면 대략 다음과 같은 관계가 된다.

```
기본 메시지 클래스
class MessageBase
{
    Type messageType;          // 메시지 타입
    SendType sendType;         // 보내는 방식
    float sendTimeInterval;    // 얼마나 시간이 지난 뒤에 보낼지 정한다.
}

특정 좌표로 이동하게 할 때 쓰는 메시지
class MessageMoveTo : MessageBase
{
    Vector3 targetPosition;      // 목표 지점 좌표
}

누군가를 공격하게 할 때 쓰는 메시지
class MessageAttackTo : MessageBase
{
    Int attackSkillID;           // 공격에 사용한 스킬의 ID
```

```
        Entity target;                  // 목표 대상
    }
```

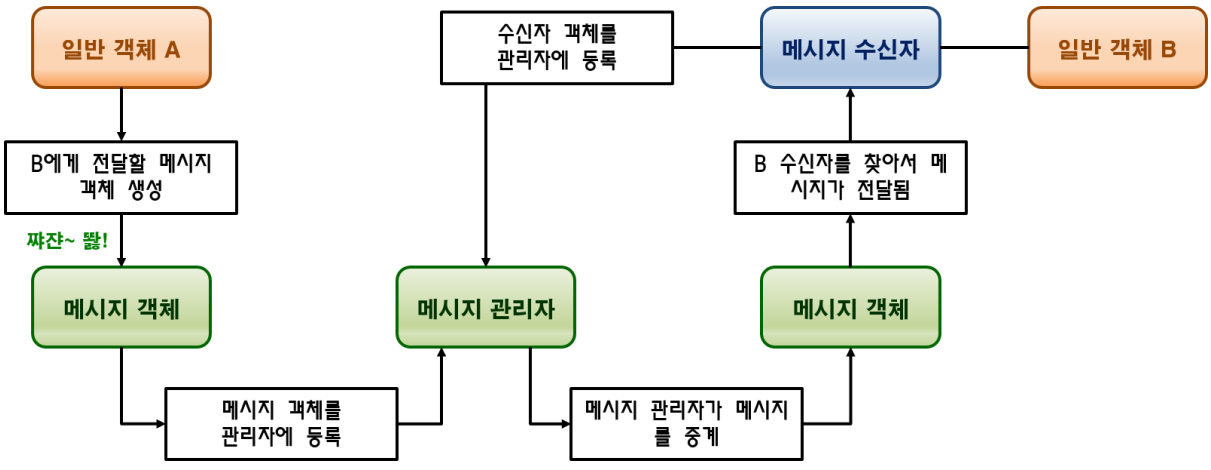
※ 기술적으로는 매개변수를 void*의 목록(C#이라면 object 의 목록)으로 표현하는 것도 가능하다. 사실 이렇게 하는 편이 써야 할 코드 양도 줄어들고 코드도 깔끔하긴 하다.
그렇지만 여기에는 매개변수 하나를 전달할 때마다 모든 매개변수를 힙에 할당해야 하는 문제가 있는데, (C#일 경우, boxing / unboxing) 이 때는 int, float 등의 원시 값(primitive type) 객체들마저도 힙에 할당해야 하므로 성능상 불이익이 발생한다.
그것을 감수할 정도로 범용성을 요구할 의미가 없으므로, 좀 번거롭더라도 필요한 매개변수 집합은, 그에 해당하는 자식 메시지 클래스를 새로 제작해서 구현하는 방식을 사용한다.

- **B-4-2-4.** 모든 메시지 객체들은 최대한 재활용해서 사용한다.
: 일반적인 객체들의 재활용 전략과 동일한 선상으로 보면 된다. 새롭게 메모리에 할당하는 동작을 최소화하고, 이미 존재하는 같은 종류의 메시지 객체들의 값을 초기화해서 다른 용도로 재사용한다.

- **B-4-2-5.** 메시지 객체들은 필요에 의해 내부 값을 초기화할 수 있어야 하고, 초기화된 상태인지 여부를 확인해주는 인터페이스를 제공해야 한다.
: 그래야 재활용할 때 안심하고 쓸 수 있다.

◆ B-4-3. 메시지 핸들링

- **B-4-3-1.** 메시지를 주고 받는 과정은 대략 다음과 같다.



<게임 메시지 전달 과정>

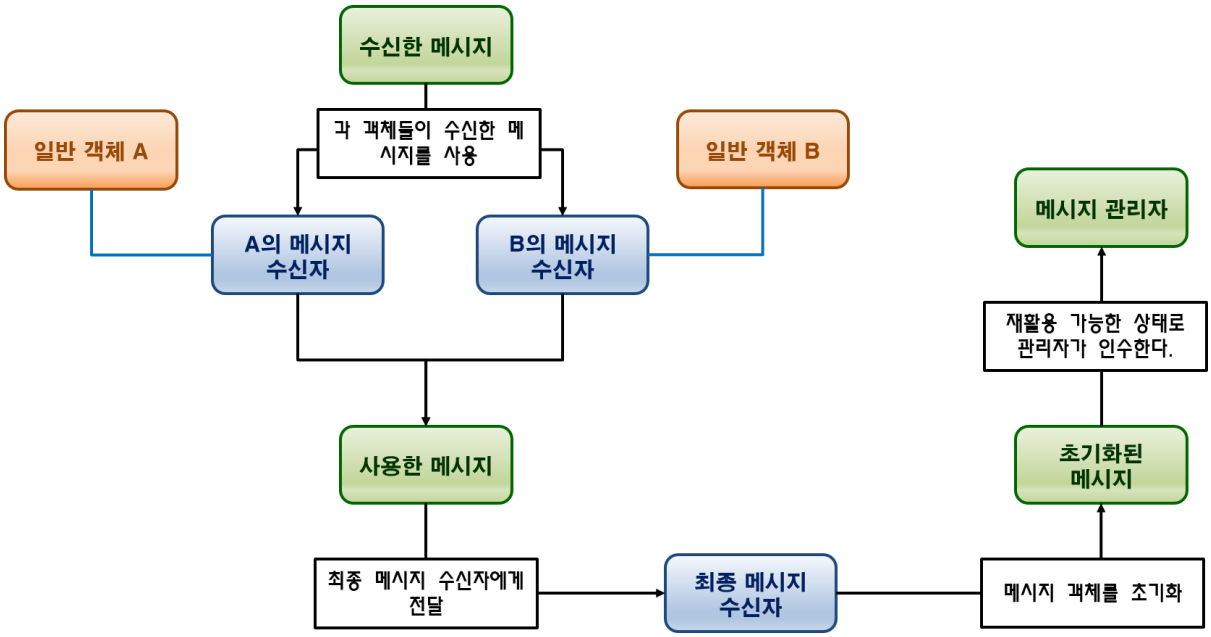
- **B-4-3-2.** 메시지를 수신 받기 위해서는, 해당 객체가 메시지 수신자의 내용을 구현하고, 메시지 관리자에 등록을 해야 한다.

※ 이를 기술적으로 구현하기 위해 메시지 관리자는 인터페이스(interface)로 구현한다.
만약, 인터페이스 문법이 없는 경우에는 대리자 패턴(Delegate Pattern)을 통해서 구현하면 될 것이다.

- **B-4-3-3.** 메시지는 즉시 전달받거나, 혹은 특정한 시간 뒤에 전달받을 수 있다.
: 즉시 전달하는 메시지는 굳이 메시지 관리자가 보관하지 않고 즉각 처리해도 된다.

※ 좀 더 일반화할 수 있다면, 시간 외 다른 조건 타입으로도 전달 시점을 조절하게 할 수는 있을 것이다. 하지만 그것을 실제로 구현하기에는 조건도 애매하고, 효용 가치도 얼마나 높을지 의문이다.
그 정도로 복잡하고 흔치 않은 전달 조건이라면, 차라리 그런 조건은 직접 구현하는 게 맞을 것이다. 설계는 가장 흔한 사용 패턴을 위주로 고려하자.

- **B-4-3-4.** 메시지 객체를 수신한 수신자 객체들은, 사용한 메시지 객체를 최종 메시지 수신자에게 전달해야 한다.
: 이는 메시지 객체의 사용 종료를 통지해서, 그 메시지 객체를 재활용할 수 있게 하려는 목적이다.



<메시지 객체 사용 후 처리>

※ 이는 Win32 API 에서 Window Procedure 를 처리하는 과정과 유사하다.
만약, 메시지 객체를 전달하기만 하고, 전달한 메시지 객체의 종료를 추적할 수단이 없다면, 그 메시지 객체를 어느 시점에 재활용할 수 있는지는 누구도 알 수 없다. (아, 엄밀하게는 엔진 내부에서 더 이상 참조가 없음을 알고 GC 에 연락해서 삭제하는 경우만 뺀다면...)

- **B-4-3-5.** 모든 메시지 객체는 내부에서, 메시지 객체를 이용하는 대상의 개수를 참조하는 카운트를 가진다.

- **B-4-3-6.** 메시지 관리자가 메시지를 각 수신자에게 전달할 때마다 참조 카운트는 증가하고, 메시지 사용이 끝나서 최종 메시지 수신자에게 전달되는 시점마다 참조 카운트가 감소한다.

- **B-4-3-7.** 참조 카운트가 0인 메시지는 초기화되며, 다음 메시지를 위해 재활용 가능하다.

※ 즉, 수신자가 10 개면 메시지는 각 수신자에게 전달이 끝난 시점에서 +10 의 카운트를 가진다.
이 상태에서, 각 수신자들이 메시지를 가지고 볼 일을 보고 난 후, 최종 메시지 수신자에게 다시 메시지를 던질 것이다. 최종 메시지 수신자는 수신된 메시지마다 메시지 내부의 참조 개수를 -1 씩 깎는다.
10 번째 메시지가 최종 메시지 수신자에게 들어오면, +10 의 카운트는 0 이 되었을 것이고, 이 순간부터 메시지는 초기화 및 재활용을 해도 안전하다. (아무도 사용하는 곳이 없으므로)

- **B-4-3-8.** 메시지 객체의 수명은 해당 장면에서만 작동하는 경우와, 현재 장면의 종료와 상관없이 게임 애플리케이션이 끝나기 전까지 작동하는 경우를 구분할 수 있어야 한다.
: 사실 용도 상으로는, 대개 메시지 객체는 특정 장면 안에서만 필요한 경우가 대부분이다.

- **B-4-3-9.** 메시지 관리자는 장면 프로세스의 전환 여부와 관계없이 게임 애플리케이션의 실행 – 종료 시점 내에서 항상 실행한다.

- **B-4-3-10.** 메시지 관리자는 장면이 전환될 때, 종료되는 장면 내부에서만 사용하는 메시지들을 골라서 선택적으로 종료하게(또는 메모리에서 해제하게) 할 수 있어야 한다.

B-5. 캐릭터 개체

◆ B-5-1. 캐릭터 개체의 특질

- **B-5-1-1.** 프로그램 코드에서의 캐릭터는, 게임 플레이를 진행하기 위한 능력치를 보유하고 있고, 경우에 따라 움직이거나, 사용자가 조종하거나, 상호작용할 수 있는 객체들을 일컫는다.
  : 보통 **캐릭터 개체(Character Entity)**, 줄여서 **개체(Entity)**로 많이 표현한다.

※ 그저 객체, 게임 객체라고 표현해도 좋지만, **객체(Object)**라는 말 자체가 프로그래밍 영역에서 워낙 광범위하게 쓰이는 용어이고(OOP, 즉, 객체지향 프로그래밍부터 해서...), 뉘앙스 역시 게임 캐릭터와는 매우 다르기 때문에 용어의 혼동을 피하기 위해서 위와 같은 용어를 도입하였다.
캐릭터(Character)는 인간 형태인 경우를 나타내는 뉘앙스가 강하고, 유닛(Unit)은 원래 의미 그대로, 수학에서의 단위를 나타내야 할 때가 있으므로, 그나마 의미가 덜 겹치고 혼동이 덜 되는 용어를 찾다 보니 개체(Entity)가 좋겠다고 판단했다.

- **B-5-1-2.** 모든 캐릭터 개체들은 위치, 회전, 크기에 대한 변환이 가능해야 한다
  : 즉, **게임 세계를 표현하는 공간의 특정 지점에 위치를 가질 수 있어야 한다.**.

- **B-5-1-3.** 캐릭터 개체의 외형을 표현하는 모델 데이터는 실행 도중에 필요에 따라, 다른 모델 데이터로 교체 / 교환이 가능해야 한다.
  : RPG에서의 캐릭터 개체는 파츠를 갈아입거나, 변신 기능이 요구되는 등, 현재 표현 중인 모델에 대한 변경이나 변형이 필요한 경우가 있다.

※ 당장 만들기에는 캐릭터 개체마다 하나의 모델 데이터와 강하게 결합시키는 방식으로 설계하는 게 편하고 직관적이라고 생각할 수 있다.
그러나 이러한 설계 방식은 나중에 게임 컨텐츠를 업데이트하거나 확장하고자 할 때는 기본 설계 상의 실수로 작용할 수 있으므로, 신중하게 명세를 정해야 한다. 한 번 정해지면 코드가 성장한 뒤에는 바꾸기가 쉽지 않고 비용이 많이 드는 부분 중 하나다.

- **B-5-1-4.** 캐릭터 개체를 구성하는 외형, 행동 제어, 능력 속성들의 요소들은, **각 요소가 공통적으로 표현하고자 하는 목표에 맞춘 컴포넌트**들로 나누고, 이러한 컴포넌트들의 집합으로 캐릭터 개체를 표현하는 방식으로 설계한다.
  : 캐릭터 개체를 구성하는 개별 요소들이 각기 가지는 임무가 워낙 많고 다양하기 때문이다.
  이를 상세히 다루기 위해 별도의 절을 마련하였다.

- **B-5-1-5.** 게임 캐릭터는 크게 나누어, **캐릭터의 외형을 표현하는 객체**와, **캐릭터의 능력 / 상태 속성들을 표현하는 객체**로 나눌 수 있다.

- **B-5-1-6.** 캐릭터의 외형을 표현하는 객체는 일반적으로 생각하는, 캐릭터의 모델링 데이터를 포함하는 요소들의 집합을 말한다.

- **B-5-1-7. 캐릭터 기본 정보(Character Foundation Information)**
: 캐릭터의 능력 / 상태 속성들을 표현하는 객체를 이렇게 부르기로 한다.
이 정보들은 눈에 보이는 무언가가 아니라, 그 캐릭터가 게임 플레이에서 사용할 정보들만을 나열하는 데이터 덩어리로 비유할 수 있다.

※ 즉, 철학적인 관점에서 사람을 육체와 정신의 총합으로 보는 것처럼, 캐릭터의 외형 요소들을 육체 요소로 보고, 캐릭터의 상태 / 속성 데이터를 정신 요소로 비유할 수 있겠다.

+

능력치	값
레벨	골드 15
직업	생명력 회복 물약
힘	85
민첩성	80
건강	125
지능	70
공격력	240 ~ 295
방어력	862

상태효과	값
공격력 증가	+10, 5분
이동속도 증가	+15%, 2분
체력 회복	초당 +0.76
마력 회복	초당 +0.54

장비	아이템
머리	매 감시자의 투구
얼굴	기본형 1번
몸통	박쥐군주 날개
오른손	원한 맺힌 쐐기칼
왼손	없음
목걸이	성당수호자 목걸이
반지	공허의 속삭임
벨트	거인의 허리띠

슬롯	아이템
00	없음
01	1005
02	없음
03	2102
04	4004
05	없음
06	없음
• • •	• • •

<육체와 정신이 하나로(...)>

- **B-5-1-8.** 캐릭터 개체(Character Entity). 즉 개체(Entity)는 사전에 정의된, 혹은 저장된 데이터로부터 넘겨받은 캐릭터의 능력 / 상태 속성을 기반으로 생성한다.

- **B-5-1-9.** 정신에 해당하는 캐릭터 정보들은 게임 플레이 장면과 관계없이 존재할 수 있지만, 육체에 해당하는 캐릭터의 외형 요소들은 게임 플레이 장면 및, 사전에 정의된 장면에서만 등장할 수 있다.
: 게임 플레이 장면은 아니지만, 캐릭터 모델 데이터를 포함한 외형 요소들이 자주 등장하는 장

면은 캐릭터 로비 장면, 캐릭터 선택 / 생성 장면이다.

※ 모든 비유가 그렇듯, 실제와 완전히 같지는 않다는 점이 여기에 있다.
게임 캐릭터의 기본 정보는 캐릭터 개체를 생성하기 위한 설계도, 혹은 재료의 의미도 있다.
생성되어 있는 캐릭터 개체들은 그 개별 캐릭터 개체에만 적용되는 캐릭터 기본 정보들의 현재 값을 유지하고 있다. (즉, 눈에 보이게 만드는 껍데기에 대한 정보만 있는 게 아니다.)

- **B-5-1-10.** 이렇게 캐릭터 기본 정보와 캐릭터 개체를 나누는 이유는, 캐릭터 개체의 정보는 기본적으로 매우 무거운 정보이기 때문이다.
: 상대적으로 가벼운 정보들은 게임 애플리케이션에서 전역적으로 유지하고, 무거운 정보들은 필요할 때만 유지하는 방식이다.

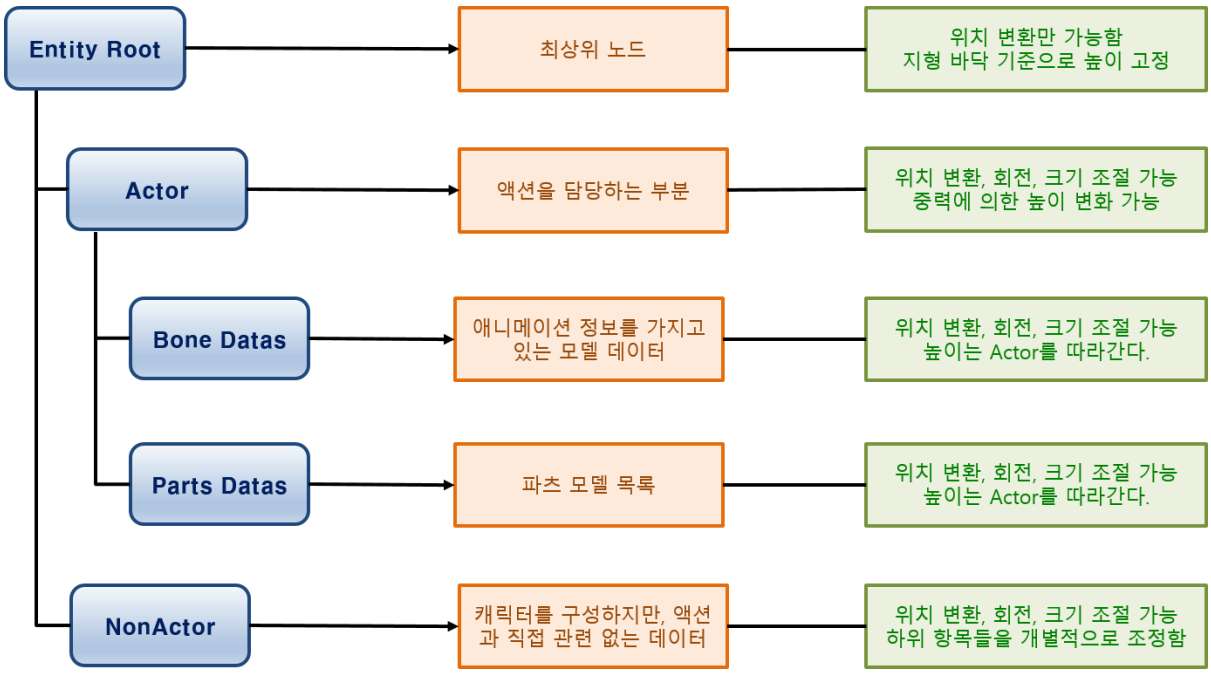
※ 캐릭터 개체는 근본적으로 최소한 게임 플레이 장면에서만 쓰기 위해 제작하는 정보들이다.
모델 리소스 데이터와 텍스처 정보, 액션 정보, 인공지능에 대한 사항들은 오직 게임 플레이 단계, 좀 더 넓게 잡더라도 클라이언트 단계에서만 필요하다. 특히, 서버는 그 어떤 외형과 관련된 정보도 필요가 없다.
또한, 이런 외형 정보들은 실행 메모리에서 차지하는 크기도 매우 큰 편이어서, 항상 실행 메모리에서 들고 있기에는 매우 부담스럽다.

반면, 캐릭터 개체를 만들기 위해 사용하는 캐릭터 기본 정보들은 그 캐릭터에 대한 핵심 정보만을 담고 있고, 크기도 매우 작은 편이다. 이 핵심 정보들은 완전히 불필요하게 되지 않는 한, 되도록 메모리에 들고 있는 전략을 취해도 메모리에 그다지 부담이 없다.
핵심 정보들은 매우 빈번히 참조되며, 이를 다시 불러오기 위해 저장장치 입 / 출력이나 네트워크 통신을 기다리는 비용을 발생시키는 것보다는 메모리에 항상 들고 있으면서 빠르게 내용을 참조할 수 있게 하는 편이 더 품질 면에서 이득이다.

◆ B-5-2. 캐릭터 개체(Character Entity)의 구조와 제한 사항

- **B-5-2-1.** 캐릭터 개체가 게임 세계의 공간에 존재할 때는 다음과 같은 객체의 서열 구조를 통해 조직화한 상태로 생성한다.
: 특별한 이유가 없는 한, 어떤 종류의 캐릭터 개체를 막론하고 다음과 같은 객체의 계통 구조로써 표현한다.

- 최상위 노드(Root Node)
- 활동 객체(Actor)
- 뼈대 정보 데이터(Bone Datas)
- 파츠 데이터(Parts Datas)
- 비-활동 객체(Non-Actor)



<캐릭터 개체의 세부 구조 컨셉>

- **B-5-2-2. 최상위 노드(Root Node)**

: 모든 캐릭터 개체들은 최상위 노드로부터 객체 구조가 시작된다.

- **B-5-2-3. 활동 객체(Actor)**

: 모델 리소스를 인스턴스 화한 데이터와, 모델의 애니메이션 인스턴스 데이터를 포함하고 있는 객체가 활동 객체이다.

- **B-5-2-4. 뼈대 정보 데이터(Bone Datas)**

: 활동 객체의 하위 구조로써, 애니메이션 정보를 담고 있는 객체들을 모아놓는 노드이다.

- **B-5-2-5. 파츠 정보 데이터(Parts Datas)**

: 활동 객체의 하위 구조로써, 파츠를 구성하는 메쉬(Mesh) 데이터들을 모아놓는 노드이다.

- **B-5-2-6. 비-활동 객체(Non-Actor)**

: 캐릭터 개체의 애니메이션 데이터가 아닌, 다른 부수적인 역할을 하는 객체들을 모아놓는 노드이다.

쉽게 생각해서, 그냥 잡동사니들 모음(...) 같다고 보면 된다.

- **B-5-2-7. 캐릭터 개체마다 최상위 노드는 단 하나만 존재**해야 한다.

: 캐릭터 개체를 게임 세계 공간에서 식별하거나, 일괄적으로 변환하기 위한 출발점이기 때문에, 캐릭터 개체의 인스턴스 당 1개를 넘어서면 안 된다.

- **B-5-2-8.** 비-활동 객체의 하위에 들어가도록 되어 있는 캐릭터 개체의 구성요소들은, 캐릭터 개체의 활동 객체(Actor)로부터의 상대적인 위치, 높이 값, 회전량, 크기 변화 상태 등을 똑같이 따르지 않아도 된다.
: 이들이 활동 객체의 현재 변환 상태(Translation)를 같이 따를지 아닐지, 따른다면 어느 정도를 반영할지 등은 모두 하위 구성 요소의 내부 구현에 따라 선택할 수 있게 한다.

※ 그림자 메쉬의 동작을 예시로 들 수 있다.
그림자 메쉬는 캐릭터 개체의 발 밑에 따라다니는 형태로 표현하는 경우가 대부분이다.
그렇지만, 캐릭터가 점프를 하거나 스킬 등에 의해 공중에 띄워진다고 해서, 캐릭터의 그림자 메쉬가 캐릭터 발바닥에 달라붙은 채(...) 공중에 같이 떠오르지는 말아야 한다. 즉, 그림자는 캐릭터가 땅 위에 서 있는 지점만 참고할 뿐, 캐릭터가 공중에 뜬 상태에 대한 위치는 참고하지 않는다.
비단 그림자 뿐 아니라, 캐릭터에게 어떤 모델이 붙고, 그 모델이 어떤 연출을 해야 하는지에 따라 캐릭터 개체의 활동 객체의 위치, 회전, 크기 변환을 적용하는 방식은 달라질 수 있다. 심지어 붙는 모델의 종류마다 다 제각각 다른 방식이어야 할 수도 있다.

이와 같은 내용을 캐릭터 객체의 모형틀 단계에서, 사전에 미리 구조적으로 정의하기는 불가능하다고 볼 수 있다. 따라서, 캐릭터 개체의 비-활동 객체는 그러한 하위 요소들에게 변환과 관련된 내용 일체를 각자 처리하도록 위임한다.

◆ B-5-3. 캐릭터 개체의 관리

- **B-5-3-1.** 캐릭터 개체는 게임 플레이 장면에서만 관리한다.
: 그 외 장면에서는 게임 플레이 장면에서 필요한 각종 컴포넌트 정보들을 유지할 필요가 없기 때문이다.

※ 게임 플레이 장면이 아니라고 해서, 활성 메모리가 펑펑 남아돌 정도로 한가하지만은 않다고 봐야 한다.
단지, 게임 플레이 장면처럼 초당 화면 갱신률(Frame Per Second)에 민감하지 않아도 될 뿐이지, GUI 관련 리소스 데이터는 게임 플레이 장면들보다 훨씬 더 많을 경우가 대부분이다.
결국, 게임 플레이 장면을 벗어나면, 게임 플레이 장면과 관련된 모든 데이터들을 메모리에서 제거하는 게 활성 메모리의 용량 부담을 줄일 수 있으므로, 캐릭터 개체들의 관리 정보 역시 게임 플레이 장면에서 벗어나면 일괄적으로 해제하는 게 맞다.

- **B-5-3-2.** 몬스터의 경우, 현재 로딩되어 있는 스테이지 영역(Stage Area, 관련 명세의 정의를 참고하라.)의 개체들만 관리하면 된다.

※ 스테이지와 관련된 명세를 보면 알겠지만, 스테이지 영역은 한 번 지나치면 다시 되돌아갈 수 없기 때문에, 현재 로딩하고 있지 않은 스테이지의 나머지 영역에 분포하고 있는 몬스터들의 정보를 관리해야 할 필요가 없다.

- **B-5-3-3. 내 캐릭터에서 일정한 거리 밖에 있는 캐릭터 개체들은, 어떠한 업데이트도 하지 않는다. 단, 그 일정한 거리는 최소한 카메라를 통해 플레이어가 보는 영역보다 넓어야 한다.**
: 어떤 방식으로든 게임을 다른 사용자들과 동기화할 필요가 없고, 화면 밖의 지역에서 벌어지는 행동들이 게임 플레이에 영향을 미치지도 않기 때문에 이러한 기준을 정한다.

※ 캐릭터의 실행 논리(캐릭터의 상태, 인공지능 수행 등)를 업데이트하는 대상을 제약하는 관리 방식은 게임 성능에 큰 이득을 준다.
게임 캐릭터 간 상호 작용은 상당한 성능 부담을 유발하기 때문에, 보통 한 화면에 최대로 등장할 수 있는 캐릭터 개체의 개수를 정하고, 그 기준에 대략 맞춰지도록 레벨 디자인을 하게 된다.
그런데, 다른 사용자들과 같은 게임 화면을 공유해야 하는 경우에는, 내 캐릭터 기준에서 멀리 떨어진 지역에 있는 캐릭터들을 '꺼 두는' 전략이 잘 먹히기가 힘들다. 내 캐릭터에게는 멀리 있지만, 다른 사용자들의 캐릭터에게는 가까울 수 있기 때문이다.
심지어, 네트워크 동기화의 요소가 없더라도 실시간 전략 시뮬레이션(RTS)의 경우에도 특정 캐릭터만 업데이트하는 방식을 사용하기는 어렵다. 카메라 시점이 특정한 캐릭터에게 고정되어 있지 않고 옮겨 다니는데다, 화면 밖에서 벌어지는 전투도 많고 이 결과가 게임 플레이에도 영향을 미치기 때문에, 카메라에 보이지 않는 지역이라고 해서 업데이트를 하지 않을 수가 없다.

결과적으로, 이러한 정책(캐릭터의 업데이트를 켜고 끌 수 있는 범위를 정하는 것)은 **어느 범위만큼의 캐릭터들이 상호작용할 때, 게임 플레이의 결과에 영향을 주느냐**에 따라, 가능한 정도가 다르다고 보면 된다.

- **B-5-3-4.** 캐릭터 개체를 관리 대상에서 제거할 때는, 그 캐릭터 개체가 사망한 상태가 아니라, 별도로 관리 대상에서 제거해도 되는 상태일 때만 수행해야 한다.
: 게임에서의 캐릭터 사망은, 곧 캐릭터를 메모리에서 제거해도 된다는 의미가 아니다.

※ 당장, 최근의 어떤 게임을 보더라도, 게임에서 캐릭터가 사망하면, 사망에 대한 액션을 취하고, 캐릭터의 죽은 시체 모델이 한동안 남아 있기도 하며, 부활할 수 있는 경우도 있다.
그러니, 캐릭터의 사망이라는 건 그냥 상태가 [살아 있는 상태] -> [죽어 있는 상태]로의 변화일 뿐이지, 캐릭터 개체의 정보를 메모리에서 지워도 된다는 의미와 같지 않다.
따라서, 캐릭터의 생존 / 죽음 상태와 객체 관리의 상태는 반드시 구분해서 별도로 유지하는 방식으로 설계해야 한다.

◆ B-5-4. 캐릭터 개체의 세력

- **B-5-4-1.** 모든 캐릭터 개체는, 최소한 하나의 특정한 세력에 소속되어야 한다.
  : 이러한 세력은 가상의 팀으로 볼 수 있다.

- **B-5-4-2.** 소속되어 있는 세력 내의 개체들끼리는 우호적 관계 밖에 형성할 수 없지만, 다른 세력과는 관계가 다를 수 있다.
  : 세력끼리는 서로 친밀할 수도, 적대적일 수도 있다.

- **B-5-4-3.** 세력간 친밀도는 점수로 관리하지 않고, 친밀 / 중립 / 적대 관계 중 택일하는 방식으로 설정한다.
  : 이 게임은 세력 간 평판 점수 같은 게 필요가 없는 게임이기 때문에, 최대한 단순하게 구현하는 게 좋다.

- **B-5-4-4.** 캐릭터가 소속되어 있는 세력을 나타내는 값은, 객관적으로 어느 쪽에서도 동일한 값으로 표현할 수 있는 방식이어야 한다.
  : 쉽게 말하자면, 내 팀 – 적 팀과 같은 방식을 쓰면 안 되고, 특정하게 어떤 팀이라고 객관적으로 지목할 수 있어야 한다.

※ '내 팀 – 적 팀'과 같은 구분 방식은 상대적인 구분 방식이다. 만일 이를 세력 구분을 하는 데 쓰는 값 타입으로 설정한다면, 같은 실행 코드를 가지고도, 기기마다 내 캐릭터가 소속되어 있는 세력에 따라 내부 값이 반대가 되기 때문에 디버깅할 때 번거롭고 문제가 될 소지가 있다.
물론, '어떤 캐릭터나 팀이 내 편인지 아니면 적인지' 여부를 확인하는 인터페이스는 있는 게 좋다. 하지만 이는 **실행 중에 팀 세력 값을 비교해서 확인해보는 함수의 형식**이어야 하는 것이지, '내 팀'이라는 타입의 값 상수가 따로 존재하는 방식이어서는 안 된다는 점이다.

◆ B-5-5. 플레이어 캐릭터

- **B-5-5-1.** 일반적으로 **플레이어가 조종하는 캐릭터**를 플레이어 캐릭터라고 한다.
  : 단, 여기에서 플레이어라고 함은 나 자신이 아닌 다른 플레이어들도 포함한다.
  즉, 내가 직접 조종하지 않더라도, 다른 누군가가 조종하는 캐릭터라면 그 캐릭터 역시 플레이어 캐릭터다.

- **B-5-5-2.** 플레이어 캐릭터의 게임 플레이 진행 방식은 사용자의 조종에 의하거나, 혹은 사용자의 선택에 의해 일부, 혹은 전체가 자동화될 수 있다.
  : 플레이어 캐릭터도 인공지능 모듈을 소유하고 있고, 인공지능에 따라 움직일 수도 있다.

- **B-5-5-3.** 플레이어 캐릭터들은 비-플레이어 캐릭터와 관리 기준을 다르게 줄 수 있어야 한다.
 : 실제

※ 플레이어 캐릭터는 소스 코드 객체의 본질적인 의미에서는 몬스터의 좀 특별한 유형에 불과하다고 볼 수도 있다.
 게임을 제작할 때는, 성능 이유로 인해 화면에 캐릭터를 XX개까지만 보이도록 한다거나, 캐릭터가 어느 거리까지 접근했을 때 내 기기에서 보이게 만들지를 조정하는 옵션을 제공할 수 있다.
 심지어는, 내 캐릭터에서 멀리 떨어져 있어서 화면에서 보이지 않는 다른 캐릭터들은 아예 모든 업데이트 자체를 중단해서, 실행 성능을 확보하기도 한다.

◆ B-5-6. 비-플레이어 캐릭터(Non-Player Character)

- **B-5-6-1.** **플레이어가 조종하지 않는 모든 캐릭터**는 기본적으로 비-플레이어 캐릭터이다.
 : 보통 **NPC**로 많이 줄여서 표현한다.

- **B-5-6-2.** 플레이어가 조종을 하더라도, 그 수단이 간접적이거나, 존재 유무가 플레이어 캐릭터로 설정되어 있는 캐릭터에게 의존적이라면, 그 캐릭터는 비-플레이어 캐릭터이다.

※ 게임에 따라 소환수나 탈 것은 직접 조종도 가능하지만 이를 플레이어 캐릭터처럼 취급하지는 않는 경우가 많다.
 이러한 캐릭터들은 플레이어 캐릭터의 존재가 이미 있지 않다면, 처음부터 생성되지도 못하는 특성이 있다. 즉, 별다른 조건 없이 직접 조정하는 유형은 아니기 때문에 플레이어 캐릭터로 간주하지 않는다.

- **B-5-6-3.** 캐릭터 개체 중에서 몬스터를 분류하는 특별한 기준은 없다.
 일반적으로, 게임 플레이에 등장하는 모든 몬스터들은, 게임 플레이 모드에 관계 없이 비-플레이어 캐릭터로 분류한다.
 : 플레이어 캐릭터에게 적대적인 성향의 NPC를 몬스터로 정의한다고 봐도 좋다.

◆ B-5-7. 내 캐릭터

- **B-5-7-1.** 플레이어 캐릭터들 중에서, **나 자신이 직접 조종하는 플레이어 캐릭터**는 내 캐릭터로 분류한다.
 : 게임 플레이를 구현할 때, 내가 조종하는 캐릭터인 경우를 구분해서 처리해야 하는 기능들이 매우 많기 때문에 이와 같은 분류가 필요하다.

- **B-5-7-2.** 내 캐릭터의 능력과 관련된 정보와, 게임 플레이 장면에서 게임 세계에 나타나는 캐

릭터는 코드 상에서는 별개로 취급하는 객체들이다.
: 내 캐릭터의 정보는 게임이 실행된 이후부터, 종료하기 전까지 유지하지만, 캐릭터 개체는 게임 플레이 장면에 들어가야 생성되고, 게임 플레이 장면을 빠져나오면 파괴된다.

- **B-5-7-3.** 게임 플레이 장면에서 등장하는 내 캐릭터 개체는, 전역적으로 활성화되어 있는 내 캐릭터의 능력과 상태 정보를 기반으로 생성한다.

※ 내 캐릭터의 정보는 눈에 보이는 무언가가 아니라, 내 캐릭터의 능력치, 아이템 보유 상황, 부여효과의 보유 상황 등의 정보를 담고 있는 보이지 않는 데이터 덩어리에 비유할 수 있다.
눈에 보이는 3D 모델 객체로 구성된 캐릭터 모델은 게임 플레이에서만 사용하는 표현 형태로 생각하면 된다. 이러한 관념은 비단 내 캐릭터에만 국한하지 않으며, 다른 플레이어의 캐릭터들 역시 마찬가지다.

- **B-5-7-4.** 내 계정이 캐릭터를 선택해서 진입한 이후의 장면에서는, 코드에서 전역적인 방식으로 내 캐릭터 정보에 접근할 수 있는 인터페이스를 제공해야 한다.

◆ B-5-8. 캐릭터 개체 컴포넌트

- **B-5-8-1.** 게임에 등장하는 각 캐릭터들은, 그 캐릭터의 상태나 행동을 분류한 컴포넌트들을 조합하여 구성한다.

※ 캐릭터 개체는 게임 플레이가 실행되는 도중에 사용자 입력부터 시작해서 게임 세계와 복합적이고도 복잡한 상호작용과 업데이트를 끊임없이 수행해야 한다.
그러한 특성을 지니고 있기 때문에, 많은 하위 구성 요소들이 필요하다.
이를테면, 캐릭터 개체는 그 캐릭터를 외형적으로 표현할 모델이 필요하고, 또 이를 애니메이션 하는 기능이 필요하다. 뿐만 아니라, 전투에 관련된 능력치들을 관리해줘야 하고, 사용자의 조종에 반응하는 기능도 필요하다. 사용자가 직접 조종하지 않는 경우에는 인공지능을 이용해 스스로 플레이를 결정해야 할 수도 있다.

그리고 그러한 하위 구성 요소들은 기획에서 정의한 캐릭터의 고유한 특징에 맞춰야 하기 때문에, 손쉽게 수직적인 계통 방식으로 서열화하기는 어렵다. 그래서 구현할 때도 Is-a 관계로 통용할 수 있는 상속 방식보다는, has-a 관계를 설정하는 컴포넌트 방식이 더 어울린다.

- **B-5-8-2.** 캐릭터의 구성 컴포넌트들은 캐릭터의 동작에 필요 요소로 작동해서는 안되고, 충분 요소로 작동해야 한다.
: 특정 컴포넌트가 없다고 해서 실행 자체가 안 되어버리거나, 실행 오류가 나서는 안 된다.

※ 컴포넌트 기능이 존재하지 않는 경우에는, 단지 컴포넌트 기능만 이용할 수 없을 뿐, 나머지

기능은 이상 없이 실행하도록 사전에 조건 처리를 하는 방식으로 구현해야 한다.

- **B-5-8-3.** 캐릭터의 구성 컴포넌트들은 실행 도중에 초기화하거나 다른 컴포넌트로 바꾸거나, 제거할 수 있어야 한다.
: 그렇게 컴포넌트들을 중간에 교체하더라도, 실행 도중에 오류로 인해 응용 프로그램이 강제 종료되면 안 된다.

※ 캐릭터의 기능들은 간혹 게임 플레이가 수행되는 도중에 바뀌어야 할 경우가 있을 수 있다.
몇 가지 예를 들자면, 몬스터가 어느 이벤트를 거친 뒤에 플레이어의 동료가 되어야 한다든가, 플레이어 캐릭터가 변신을 했는데, 날아다니는 기능이 새로 생겼다든가 하는 경우 등이 있다.

이러한 변화를 다루는 방법은 두 가지 정도의 갈래로 나눌 수 있다.
캐릭터의 구현 내용에 미리 예상 가능한 범위의 기능들은 다 정의하고 필요한 경우마다 스위치를 켜는 방식이 그 중 하나이다.
또는, 예상 가능한 범위의 기능들을 컴포넌트로 나눈 뒤, 필요한 시점이 되면, 필요한 컴포넌트들을 찾아서 바꿔 끼우는 방식이 있다.

전자의 경우, 설계는 그다지 정교하지 않아도 되지만, 게임 캐릭터의 특성에 따라 조합이 안 맞는 경우가 생기는 경우가 많아서 예외적인 처리를 많이 해야 할 수 있다. 기능 모듈의 직교성도 떨어질 가능성이 높고, 기능 모듈 하나에 모든 내용이 집중될 확률이 높으므로 거대한 코드를 지닌 클래스 객체가 나올 가능성이 높다.
그래서 컴포넌트 방식으로 캐릭터 개체를 표현하는 이상, 후자의 방식이 더 적절하다고 판단했다. 후자의 방식은 설계를 좀 더 엄밀하게 해야 하고, 호출 단계가 깊어지기는 하지만, 기능을 조합하는 방식이기 때문에 재활용 가능성이 높고, 뭔가 문제가 발생하더라도, 문제가 생긴 기능 집합만 추적하면 되기 때문에, 상대적으로 디버깅이 쉬울 가능성이 높다.

◆ B-5-9. 캐릭터의 군집

- **B-5-9-1.** 캐릭터, 특히 비-플레이어 캐릭터들은 공동의 목표를 위해서 군집을 형성할 수 있다.
: 특히, 몬스터로 등장하는 캐릭터 개체들을 그렇게 표현할 수 있다.

※ 공동의 목표라는 건 그렇게 어렵거나 낯선 개념이 아니다.
당장 수많은 RPG들을 해보면, 몬스터들은 한 마리, 혹은 여러 마리가 몰려다니면서 일정한 지역을 '지키고 있다.' 또한, 그 무리 중 한 마리를 공격할 경우, 공격 당하지 않은 근처의 나머지 몬스터들도 공격자를 인식하고 반격하도록 되어 있는 경우도 많다.
이러한 행위들은 모두 공동의 목표에 근거했다고 볼 수 있다.

- **B-5-9-2.** 군집에 속한 캐릭터들은 게임 플레이가 수행되는 도중에, 군집을 옮길 수 있으며, 혹

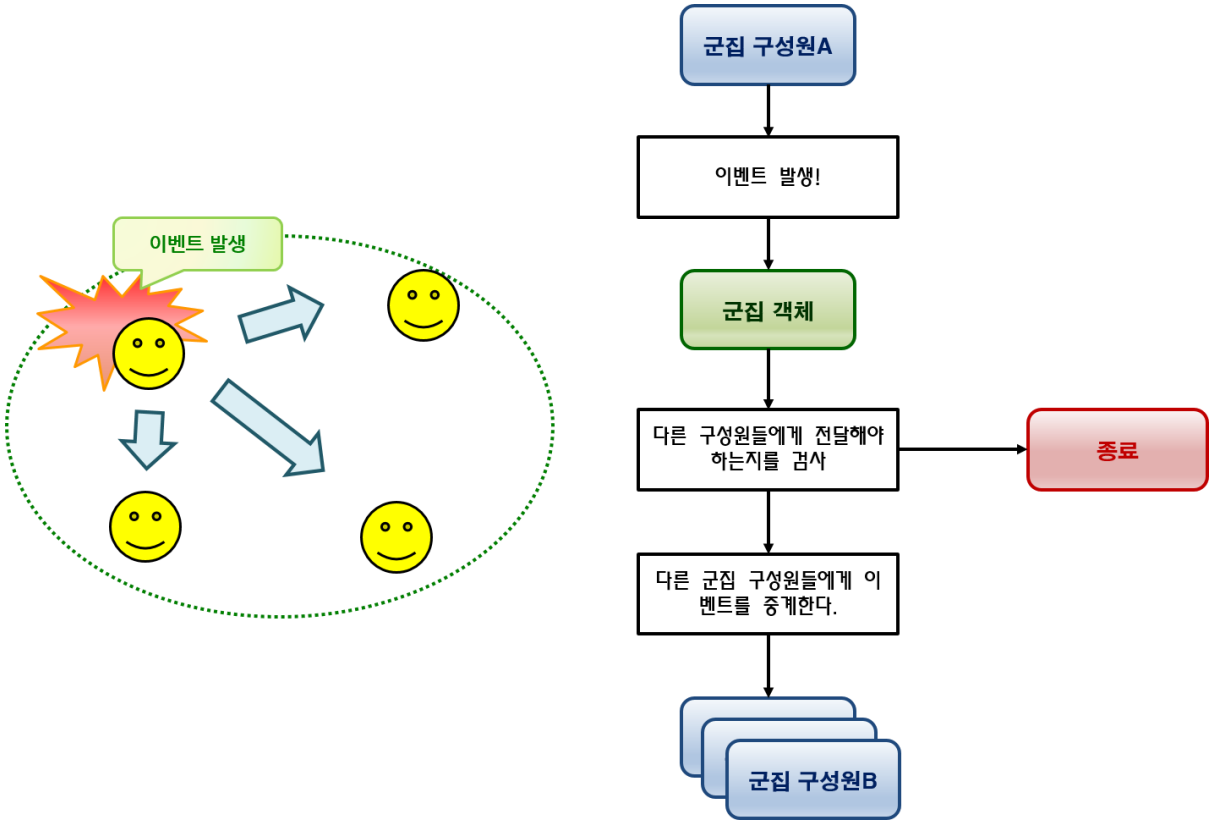
은 여러 개의 군집에 동시에 속할 수도 있다.
: 캐릭터 개체들이 여러 군집에 속하며 각 군집들의 목표를 공유하는 방식으로 설계할 수도 있다고 본다.

※ 다만, 여러 개의 군집에 소속되는 건 생각을 좀 해봐야 한다.
군집의 목표가 서로 상충되거나 우선 순위에 대해 경쟁 상태에 놓일 수 있기 때문에, 인공지능 논리를 구현할 때 어려움으로 작용할 수 있다.

- **B-5-9-3.** 캐릭터 개체에게 어떤 이벤트가 발생했을 때, 그 캐릭터 개체가 속해 있는 군집 객체는, 필요한 경우에 그 **발생한 이벤트를 군집에 소속되어 있는 다른 캐릭터 개체들에게 중계**하는 임무를 가진다.
: 군집 내 다른 캐릭터 개체들이 반응해야 하는 이벤트인 경우에는 군집 객체가 그 이벤트를 자신의 소속 멤버 개체들에게 전해줘야 다른 개체들이 자신의 그룹 내에 무슨 일이 벌어졌는지 알 수 있다.

※ 이 방법의 가장 흔한 예는, 무리 내의 한 구성원이 공격받으면 다 같이 반격하는 방식으로 행동하는 몬스터 무리들이다. 보스 몬스터와 졸병 몬스터의 관계 역시 같은 방법이다. 두 경우 다, 소속 구성원들에게 직접 공격을 받지 않더라도, 직접 공격을 받은 소속 구성원으로부터 이벤트를 전달받아서 공격자를 인식하고 공격 태세로 전환한다.
이는 RPG에서 같은 종족으로 설정되어 있는 몬스터들의 행동 패턴을 정의할 때에도 흔하게 쓰인다.

구현 관점에서 보자면, 직접 공격을 받은 캐릭터가 자신의 군집(존재한다면) 내 다른 캐릭터들에게 자신이 공격 받았음을 직접 전달하는 방식으로 구현할 수 있다고 생각할 수 있다. 그렇지만, 캐릭터들마다 반드시 군집이 존재한다고 볼 수도 없고, 어떤 이벤트를 다른 주변 캐릭터들에게 중계를 해줘야 할지를 판단하는 과정을 직접 다루면 이벤트 함수들이 대체적으로 몹시 길고 구질구질해질 것 같다.
그러느니, 이벤트를 받았으면, 자신이 군집에 속해 있는지에 대해서만 확인하면, 소속되어 있는 군집들에게 자신의 이벤트를 무작정 던지고, 군집이 받은 이벤트를 분석한 뒤, 알아서 판단해서 다른 플레이어들에게 중계할지를 결정하는 게 더 객체지향적이지 않을까 싶다.



<개체 그룹 내에서 벌어지는 이벤트 프로세스의 개념>

B-6. 지형

◆ B-6-1. 제한사항

- **B-6-1-1.** 지형 객체 모델들은 전체적으로 3D 모델로 제작한다.

- **B-6-1-2.** 지형 모델과 그에 따른 정보들은 한 번에 한 종류만 메모리에 불러들일 수 있다.
  : 다른 두 개 이상의 지형 모델과 관련 정보를 한꺼번에 메모리에 불러들일 수 없다.

※ Unity 엔진 내부의 장면 파일(*.unity) 구조 때문에 이러한 규칙을 지키도록 해야 한다.
조명맵(Lightmap)이나 길 찾기 메쉬(Navigation Mesh) 정보들이 *.unity 파일에 고정되어 있기 때문에, 이러한 정보가 필요하다면 반드시 한 번에 한 장면 파일만 불러들여야 한다..

- **B-6-1-3.** 따라서, 지형 모델 1개는 스테이지 전체 영역을 표현하거나, 혹은 스테이지의 구역 하나를 표현해야 한다.
  : 여러 개의 지형 모델을 '합쳐서' 로딩하는 방식을 쓸 수 없다.

※ 현재 버전의 Unity 엔진(현재 4.5.x)의 기능으로는 여러 개의 지형을 합치는 방식을 관리하기가 불가능한 것은 아니나, 매우 까다롭고 관리 계통과 구조도 복잡해진다.
(단, Unity 5 가 등장한 이후에는 이 내용은 바뀔 수 있다. Unity 5 에서는 여러 개의 *.unity 파일의 내용을 다룰 수 있는 Multi-Scene Editing 기능이 탑재될 예정이다. 물론, 모바일 기기에 적합한지 여부나 실제 운용상 어려움 등은 현재로써는 예상할 수 없다.)

- **B-6-1-4.** 가급적, 사용하는 텍스처에 투명도 값(Alpha) 속성을 포함하지 않게 한다.
  : 경험상, 모바일 기기에서의 Alpha Testing 성능 비용이 매우 비싸기 때문이다.
  특히, 화면에 투명도 값을 사용하는 텍스처가 항상 노출되는 상황은 평균적인 화면 그리기 성능을 크게 떨어뜨린다.

※ 상당히 많은 경우에서, 차라리 정점(Vertex) 개수를 훨씬 늘리더라도, 투명도 값을 쓰는 재질을 사용하지 않게 하는 편이 대체적으로 성능에 있어 더 나은 경우가 많았다.

- **B-6-1-5.** 사용하는 텍스처 한 개의 크기가 **2048 × 2048 픽셀**을 넘지 않게 한다.
  : 아이폰 4 이하의 기기들은 하드웨어 적으로 2048 × 2048 픽셀이 넘어가는 텍스처들을 처리하지 못한다. 또한, 활성 메모리 사용량을 가급적 억제하기 위해서라도 어느 정도 제한이 필요하다.

◆ B-6-2. 정적 지형

- **B-6-2-1.** 정적인 메쉬를 이용해서 제작하는 지형 모델을 말한다.

- **B-6-2-2.** 정적 지형들은 게임이 진행되는 동안 지형을 구성하는 메쉬의 모양이 변형되거나, 움직이거나, 충돌 영역이 변하지 않는다.

- **B-6-2-3.** 정적인 지형이라도, 텍스처의 UV 좌표를 이용한 애니메이션이나, 광원의 변화에 의해 빛이 반사되는 효과의 결과 등은 달라질 수 있다.
  : 즉, 물리적이지 않은 변화는 가능하다.

- **B-6-2-4.** 일반적인 지형들은 정적인 지형들이다.

※ Unity 엔진에서 지형을 표현할 때, 성능에 부담을 주지 않으면서 품질을 높일 수 있는 기능들(조명 맵 Lightmap, 길 찾기 메쉬 Navigation Mesh 등)은 지형이 고정된 메쉬(Static Mesh)로 설정되어 있어야 하는 경우가 많다. (사실 대부분의 엔진이나 미들웨어들도 이런 점에서 크게 벗어나지 않는다.)
따라서, **고품질의 지형을 더 낮은 성능의 하드웨어에 표현하기 위해서는, 되도록 많은 지형 메쉬들을 정적인 메쉬로 표현**해야 한다.

◆ B-6-3. 동적 지형

- **B-6-3-1.** 동적인 지형들은 게임 플레이 과정에서 있다가 없어지거나, 혹은 없다가 새로 생기거나, 파괴 가능한 등의 특성을 가진 지형이다.
  : 이런 지형지물은 대개 장애물(Obstacle)로써 동작한다.

- **B-6-3-2.** 이런 지형들은 (대개 이동할 수 있는 속성이 존재하지 않는), 일종의 특수한 캐릭터 개체라고 볼 수 있다.
  : 물론, 캐릭터 개체를 표현하기 위한 많은 정보들을 다 가지고 있을 필요는 없을 것이다.

◆ B-6-4. 물리 기능 처리

- **B-6-4-1.** 걸어 다녀야 하는 하는 지형에서는, **캐릭터들이 지형 메쉬의 외형 아래로 뚫고 들어가지 못하도록 처리**해야 한다.

- **B-6-4-2.** 지형의 좌표 범위에 대한 특별한 제약은 없다.

※ 굳이 편리를 위해 어떤 제한을 둔다면, **(0, 0, 0) 좌표가 지형 모델이 사용하지 않는 좌표가 되도록 위치를 조정하는 방안**을 생각해볼 수는 있을 것이다.
왜냐하면, 소스 코드에서 '유효하지 않은 좌표 값'인지 판별해야 하는 경우가 간혹 있는데, 이럴 경우에 (0, 0, 0)과 같은 특수한 지점의 좌표는 절대로 사용되지 않는다는 점을 보장할 수 있으면, 값을 검사할 때 편리하기 때문이다.

- **B-6-4-3.** 지형의 '바닥' 높이에 대한 특별한 제약은 없다.

- **B-6-4-4.** 지형의 '바닥'의 평평한 정도에 대한 특별한 제약은 없다.
: 자연스럽게 바닥으로 인식할 정도의 높이 편차나 경사는 허용한다. (물론, 그렇다고 해서 걸어 다닐 길에 대한 경사를 절벽처럼 급격하게 만들지는 않을 것이다.)

- **B-6-4-5.** 지형 모델에서도, 특정한 구역은 플레이어 캐릭터가 아예 진입해서는 안 되는 구역이 존재해야 할 수도 있다.

※ 예를 들자면, 맵 경계면에 있는 산이나 바다 같은 지형들은, 사용자들이 빈 공간을 보지 못하도록 가려주는 장치이다. 대개의 게임들은 이곳을 어떤 방법으로든 지나쳐 갈 수 없게 의도적으로 막아둔다.

※ 그런 장소들은 아예 그 근처부터, 겉으로는 보이지 않는 별도의 수직적인 충돌체들을 배치해서, 그러한 벽 충돌체가 플레이어 캐릭터의 충돌체를 감지하면 플레이어 캐릭터의 의 진입을 막는다.

◆ B-6-5. 위치 정보와 속성 정보의 표현 방식

- **B-6-5-1.** 각 지형마다 위치 정보가 특별히 필요한 경우에는, 필요한 위치 정보들을 알려주는 객체들의 목록을 보유할 수 있다.
: 예를 들자면, 인공지능으로 길을 탐색할 때, 반드시 거쳐가야 하는 위치에 대한 정보라든가,. 적 몬스터가 생성될 지점에 대한 정보 등이 있다.

- **B-6-5-2.** 지형과 관련이 있으나, 지형 메쉬를 직접 그리는 데 필요한 정보가 아닌 정보들을 가진 객체들은 지형 메쉬들과 다른 객체 그룹으로 묶어야 한다.
: **[B-2-2]** 항목의 스테이지 객체들의 계통 구조를 참조한다.

- **B-6-5-3.** 특별한 경우가 아니라면, 각 지형 맵의 정보를 표시하기 위해 맵을 세밀한 구획으로

쪼갠 타일 방식의 맵 정보를 구성하지는 않을 것이다.

- **B-6-5-4. 2차원, 혹은 3차원 좌표 값으로 (0, 0), (0, 0, 0)은 유효한 값이어야 한다.**
: 만약 유효하지 않은 값을 표현하고자 한다면, 2차원 좌표일 때는 (NaN, NaN), 혹은 (Infinity, Infinity), 3차원 좌표일 때는 (NaN, NaN, NaN) 혹은 (Infinity, Infinity Infinity)을 무효 값으로 사용해야 한다.

※ NaN(Not a Number)과 Infinity(무한대)는 ISO 표준에 의해 float 과 double 등의 부동소수점 실수 표기에서 사용하는 특수한 값이다.
좌표에서는 x, y, z 좌표가 모두 0 인 경우가 분명히 존재할 수 있고, 음수 좌표도 좌표가 될 수 있기 때문에, 무효한 값을 아무 생각 없이 (0, 0)이나 (0, 0, 0)으로 사용하면 곤란한 상황이 발생할 수 있다.

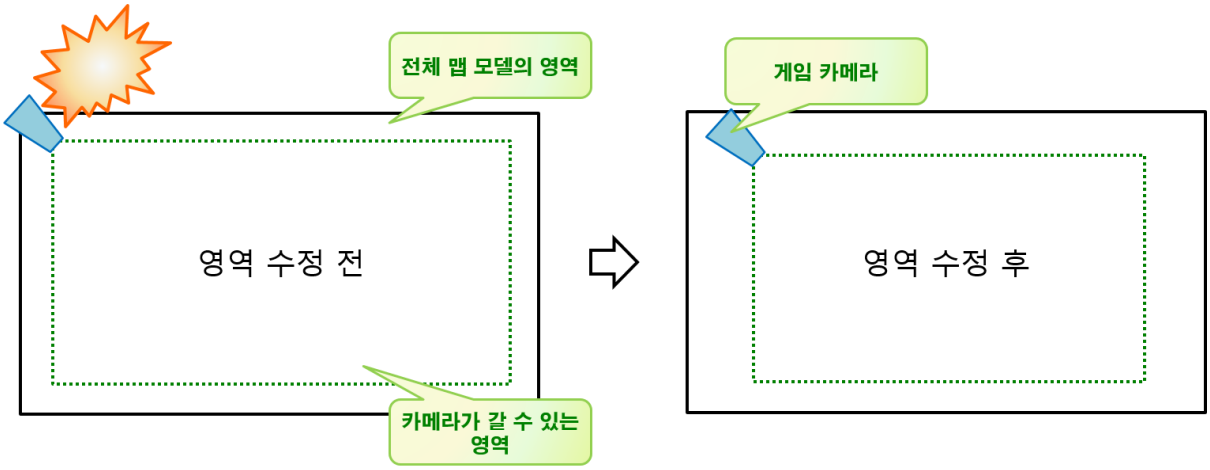
좀 다른 방식을 고려하고 있거나, 아예 모든 지형을 제작하고 배치할 때, 지형의 모든 범위가 (0, 0)이나 (0, 0, 0) 좌표를 포함하지 않게 하는 방식으로 규칙을 정하는 것도 한 방법이 될 수 있다. 이렇게 제작하는 경우에는 (0, 0)이나 (0, 0, 0) 좌표를 확실히 무효한 값으로 쓸 수 있다.
좌표를 불가피하게 정수형으로만 표기해야 한다면, 이 방법이 더 좋을 수도 있다.

◆ B-6-6. 경계 처리

- **B-6-6-1.** 각 미션 맵의 지형에서, 지형의 가장자리에 들어가면, 가장자리 좌표에 도달하기 전에, 카메라가 캐릭터를 따라다니는 이동을 중지한다. (이 상태에서도 회전 / 확대 등은 금지된다.)
정지하는 기준이 되는 좌표는, 카메라가 현재 보여주는 영역에서 맵 가장자리 너머가 보이지 않는 가로, 세로의 한계선상에 있는 좌표다.

- **B-6-6-2.** 카메라가 보여주는 영역이 맵 지형의 경계를 지나기 전에 카메라가 멈추면, 캐릭터는 맵 좌표의 가장자리 좌표까지만 이동할 수 있다. 그 너머로 이동하려고 하면, 충돌시켜서 맵 좌표 영역 바깥으로 이동하지 못하게 해야 한다.

- **B-6-6-3.** 카메라 경계 처리를 할 때, 기술적인 이유 때문에 실제 맵 모델이 차지하는 영역보다, 게임에서 카메라가 갈 수 있는 범위를 제한할 수 있다.
: 이러한 제한 범위는 **'사용자에게 지형 모델 너머의 공간이 안 보이는지'**에 따라 달라진다.



<맵 영역 너머 공간을 못 보도록, 카메라가 갈 수 있는 영역을 조정함>

◆ B-6-7. 카메라를 가리는 부분에 대한 처리

- **B-6-7-1.** 가장 좋은 방법은, 지형을 설계할 때, 카메라 시야를 가리는 부분이 생길만한 여지를 만들지 않는 것이다. (...)

- **B-6-7-2.** 만약, 카메라가 캐릭터를 바라보는 공간 사이에 반드시 지형 메쉬가 끼어들어야 할 수 밖에 없는 경우가 생긴다면, 그 지형 메쉬는 카메라의 앞을 가리게 될 시점에서, 보이지 않게 처리한다.

※ 이렇게 하려면, 사전에 카메라를 가릴 것으로 예상되는 메쉬들은, 지형 모델의 다른 메쉬들과 다른 그룹 단위로 묶여야 한다.
카메라 앞을 가리는 메쉬들의 텍스처는 특별히 투명도를 사용할 수 있게 해서, 카메라를 가릴 때 투명한 정도의 값을 올리는 방식도 있겠으나, 그러지 않고 그냥 단순히 카메라를 가리는 메쉬들만 '휙' 사라지는 것도 나쁘지는 않다. (후자가 연출은 별로일 수 있어도, 성능상으로는 더 이득이다.)

B-7. 애니메이션 시스템

◆ B-7-1. 애니메이션 객체

- B-7-1-1.

◆ B-7-2. 구성과 제한사항

- B-7-2-1. 독점적인 실행

- B-7-2-2.

◆ B-7-3.

- B-7-3-1.

B-8. 액션 시스템

◆ B-8-1. 액션 객체

- **B-8-1-1.** 액션은 **정지 과정(Stop), 재생 과정(Play), 일시 정지 과정(Pause)**을 가져야 한다.

- **B-8-1-2.** 액션은 제한적인 횟수만큼 재생하거나, 혹은 일부러 종료하지 않는 한, 무한히 반복해서 재생하는 경우 두 가지를 모두 표현해야 한다.

- **B-8-1-3.** 액션의 재생 속도를 조절할 수 있어야 한다.
: 예를 들자면 이동 속도가 증가하거나 감소해야 할 경우에 사용해야 하는데, 단정도 실수(float) 형으로 나타내면 될 것이다. (1.0 = 1 배속)

※ 현재는 Unity3D 내부의 애니메이션 재생 함수에서 지원하는 방식대로 사용해도 무리가 없다고 판단함.

- **B-8-1-4.** 각 액션은 다른 이벤트와 합성할 수 있어야 한다.
: 이에 해당하는 대표적인 것들은 이펙트 재생 이벤트, 사운드 재생 이벤트 등이다.

- **B-8-1-5. 한 번에 하나의 액션만 재생할 수 있다.**
: 만약, 여러 개의 동작이 동시에 수행되어야 하는 경우, 별도의 액션으로 분리해야 한다.

※ 이를 테면, 달리기 동작과 공격 동작이 동시에 이루어져야 하는 경우를 들 수 있다. (이걸 실제로 게임에 집어넣는지 여부와 상관 없이...)
이럴 때, 액션 시스템의 구조 상으로는, 달리기 동작과 공격 동작으로 동시에 Play() 할 수는 없다는 말이다. 이럴 때는 '달리면서 공격하기'라는 별도의 액션으로 전환해서 Play()를 해줘야 한다.

- **B-8-1-6.** 사용자가 입력한 회전 명령은 **'즉각적으로'** 적용하여야 한다.
: 플레이어의 입력이 즉각적으로 반영되어, 반응성이 좋다는 느낌을 주기 위해서다.

※ 실제 움직임에 유사하게 만들기 위해서, 혹은 부드러운 움직임을 보여줄 목적으로 회전에 대한 보간을 준 경우일지라도, **내 캐릭터에 대해서만큼은** 보간을 매우 빠른 속도로 주거나, 아니면 즉각적으로 회전하는 별도의 회전 방식을 사용한다.
그러나 선회 기능이 있는 것이, 즉각적인 회전에 비해 반드시 사용자 느낌이 좋다고 할 수는

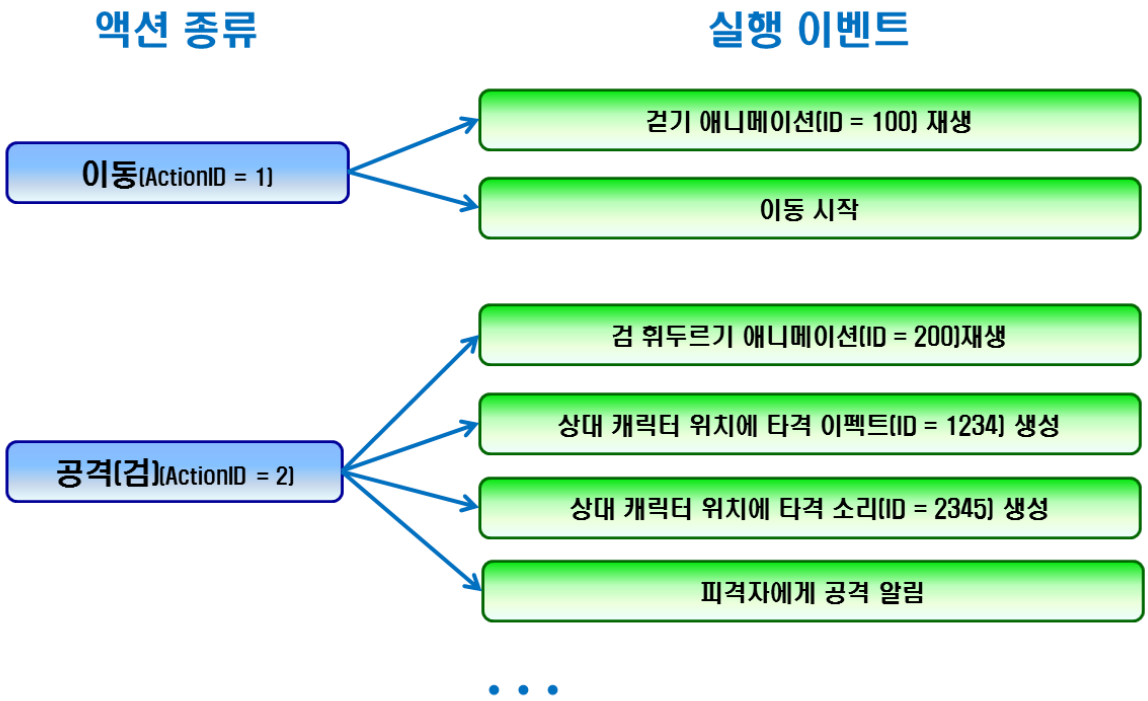
없다. 더 간편하면서도 사용자 입장에서 반응성이 좋다고 느낄 수 있는 구현으로 가는 것이 좋다고 본다.

- **B-8-1-7.** 모든 액션 내 이벤트들의 호출은, 액션 재생 시간 간격 이내에서 수행되어야 한다.
  : 만약 액션이 2초 짜리인 경우라면, 액션 내 이벤트들은 2초 안에 호출이 이루어져야 실행이 가능하다는 의미이다.

- **B-8-1-8.** 액션 재생 시간 간격의 범위를 벗어나서 호출을 요청하는 이벤트들의 실행은 무시할 수 있다.
  : **모든 액션 이벤트의 실행은, 액션의 수행 시간에 의존적**이다.

◆ B-8-2. 구성과 제한사항

- **B-8-2-1.** 액션 객체는 사실 렌더링이나 애니메이션 데이터와는 별 상관이 없는, **순수한 이벤트 함수들의 집합**이다.
  : 캐릭터가 게임에서 분류된 특정한 행동을 함으로써 발생하는 각종 이벤트들을 일정한 구조로 모아놓은 단위가 된다.



<액션은 캐릭터 별 실행 이벤트들의 집합이다.>

- **B-8-2-2.** 액션 객체의 각 이벤트 단위는, 관련된 리소스 데이터를 발견할 수 없으면, 그냥 다

음 이벤트로 넘어가는 식으로 설계한다.
: 리소스 데이터가 없는 경우에도 액션 자체는 진행하는 것이 가능해야, 각 구성 요소에 대한 의존성을 줄일 수 있어서 앞으로의 제작 및 테스트에 유리하다.

※ 실제로 제작하는 경우에는, 리소스 데이터의 정보에 의존해야 하는 경우도 많기 때문에, 이 내용이 반드시 모든 액션 객체의 코드에 대해 만족해야 하는 것은 아니다.

- **B-8-2-3.** 액션 이벤트의 수행 실패가 일어난 경우, 어떠한 실패가 발생했는지에 대한 로그를 남겨줄 수 있어야 한다.
: 그냥 다음 액션 이벤트로 넘어간다고 장땡이 아니라...

- **B-8-2-4.** 액션 객체는 소유한 액션 이벤트를 실행할지 말지에 대한 판단을 하지 않는다.
: 액션 객체는 '일반적으로' 액션 이벤트를 취사 선택해서 재생하지 않는다는 뜻이다.
소유한 액션 이벤트들은 특별한 경우가 아니라면, 조건 없이 수행한다.

※ 이는 객체와 관련 도구의 설계 / 구현을 좀 더 단순하게 하기 위한 장치다.
예를 들자면, 같은 '일반 공격' 액션이라도 들고 있는 무기에 따라서 애니메이션이나 시각 FX 이벤트들은 얼마든지 달라질 수 있다. (대부분의 게임에서 맨손 공격과 대검을 든 상태에서의 공격을 같은 애니메이션으로 처리하지는 않는다.)
둘 다 '일반 공격'이긴 하지만, 들고 있는 무기의 종류에 따라 다른 이벤트를 재생하도록 누군가 '판단'을 해야 한다.

이럴 때, 액션 객체가 스스로 '맨손 공격 액션'과 '대검 공격 액션'을 판단하게 만든다는 게, 액션 객체에게 있어 과도한 권한이라고 본다.
예시에서는 그저 '무기 종류'만을 들었지만, 실제로는 피격자의 갑옷 재질이라든가 지형 상태라든가, 기타 분류할 수 있을만한 상황이 수도 없이 존재할 수 있다. 이를 액션 객체가 '알아서 판단' 하기 위해서는 게임 구현 상의 수많은 분류 방식에 대한 타입들을 다 찾아볼 수 있어야 한다.

이런 상황은 객체들의 결합도 측면에서 볼 때, 결코 좋은 상황이라고 할 수 없다.
또한, 액션 이벤트들의 많은 부분을 외부에서 컨트롤할 수 있도록 구현해야 하는데, 각 액션 객체마다 이렇게 많은 타입에 대해 접근 가능해야 하고 설정해야 할 사항이 많다면 사용성을 크게 해친다.
액션 한 개 만들 때마다 수많은 옵션을 설정하고 이벤트 분기를 따진다고 생각해보자. 분명 기능은 강력할지 모르지만, 만든 사람 외에는 누구도 사용 방법을 제대로 파악하기 어려울 게 분명하다. 심지어, 그렇게 외부에서 컨트롤할 수 있도록 구현하기도 결코 쉬운 일이 아니다.
(그렇게 할 거면 차라리 별도의 툴을 만들고 말겠지...)

- **B-8-2-5.** 같은 의미를 가지는 액션이라도, 상황에 따라 다른 이벤트를 설정해야 한다면, 그 액

션들은 별개의 액션 종류로 분류해야 한다.

※ 예를 들자면, 같은 공격 스킬을 사용했을 때, 검을 들고 있는 경우와, 양손 창을 들고 있는 경우에 재생할 애니메이션과 FX 효과가 좀 달라야 할 수 있다.
좀 더 다른 예로는, 스킬의 부여효과를 적 대상에게 줄 때, 적이 작은 몬스터인지, 거대한 보스 몬스터인지에 따라 FX 이벤트를 좀 다르게 주고 싶은 경우도 있다.
이러한 분류를 한두 가지 기준으로 설정한 액션의 내부에서 기계적으로 분류해주기는 불가능에 가깝다. 그래서 차라리 의미적으로는 같은 액션이라도 각 상황 분류마다 액션 객체를 별도로 두는 게 낫겠다고 판단하였다.
즉, 이는 '휘두르기' 액션인 경우에, 실제로는 '검으로 휘두르기', '도끼로 휘두르기', '쌍수 휘두르기' 액션이 별도로 나뉘어서 존재한다는 뜻이 된다. (Action Code 값이 다르다.) 피격 액션일 경우에도, 공격 당할 수 있는 스킬마다 피격 액션이 다 따로 존재할 수 있다.

◆ B-8-3. 액션 이벤트의 처리 과정

- **B-8-3-1.** 각 액션들은 고유의 **액션 코드 번호(Action Code)**를 가진다.

- **B-8-3-2.** 액션 코드는 모든 액션의 분류 상황(캐릭터 종류, 직업 종류, 무기 종류, 재질 종류 등) 별로 고유해야 한다.

※ 같은 일반 공격 액션도 맨손 공격, 장검 공격, 양손 철퇴 공격은 다른 액션 코드를 가진 별개의 액션으로 처리해야 한다. (만일 전부 같은 동작으로 처리하는 것으로 처음부터 설계하지 않았다면 말이다.)

- **B-8-3-3.** 여러 개의 독립적인 액션 코드를 가지는 액션들을 성격 별로 분류할 수 있다.

※ 같은 동작이라도 상황에 따라 연출이 달라질 수 있다.
쉬운 예를 몇 개 들자면, 평화로운 상황에서의 대기 동작과 전투 상황에서의 대기 동작이 다를 수 있다. 공격 액션 / 방어 액션도 들고 있는 무기, 캐릭터의 종족, 직업 등에 따라 각각 달라질 수 있다.
그리고 그러한 분류 각각은 고유의 액션 코드를 가지는 별개의 액션 객체로 취급한다.

- **B-8-3-4.** 외부 데이터로 액션 데이터를 편집할 때는, 액션 데이터의 연출 부분만 편집할 수 있게 한다.
상황에 따라 액션을 선택하는 기능은 게임 프로그램 코드에 의존한다.

※ 독립적인 액션 객체들을 성격에 따라 묶는 방식은 아주 다양한 방법의 타입으로 접근할 수 있다.

같은 종족인 경우, 공격 액션인지 여부, 검을 사용하는지 등등... 이런 분류들을 전부 코드 외부의 데이터 조작만으로 설정한다는 것은, 가능하다 하더라도 지나치게 복잡한데다, 제대로 오류 없이 구현하기에도 매우 어렵다.
그래서, '어떤 상황에서 어떤 액션을 선택할 것인지' 여부는 코드 내부의 구현에만 의존하도록 제한한다. 이렇게 하는 편이 유연성은 좀 떨어질지 몰라도, 다루기에 훨씬 덜 복잡하고, 디버깅하기에도 훨씬 더 수월하다.

※ 좀 더 기술적인 부분으로 설명하자면, 외부에 노출되어 있는 (아마도 텍스트 스크립트나 Prefab으로 편집할) 데이터 형식에서는 오직 액션에서 어느 타이밍에 어떤 애니메이션을 재생하고, 이 타이밍에는 ○○○이라는 FX를 재생하고... 이런 내용들을 편집할 수 있다.

하지만 '공격' 액션을 재생해야 할 때, 현재 들고 있는 무기가 '장검'이기 때문에 '장검 공격' 액션으로 전환해야 한다면, 이를 판단하는 주체는 전술한 외부에 노출된 스크립트 데이터가 아니라, 공격 액션의 Play()를 호출하는 부분이다.
구체적으로는 구현하기에 따라, 인공지능 이벤트를 구현하는 객체이거나, 각 캐릭터 별 액션 관리자일 수도 있다.

- **B-8-3-5.** 액션이 전환되었더라도, 이전 액션에서 재생하던 시각 이펙트(보통 FX로 지칭함)와 사운드는 다음 액션으로 전환된 뒤에도 그대로 끝까지 재생하게 한다.

※ 이렇게 하기 위해서, 시각 이펙트와 사운드는 자체적으로 재생 시간을 관리하는 기능을 가지고 있어야 한다. (이펙트의 관리 구조에 대한 내용들을 참고할 것.)
이 경우에는 사실상 액션의 재생 시작 / 끝 시간은 불분명해질 수 있다. 또는 시작 시간은 있어도, 끝 시간은 불분명할 수 있다.

- **B-8-3-6.** 같은 액션 이벤트에서, 여러 개의 액션 이벤트가 동시에 수행하는 것은 가능하다.
: 캐릭터 객체가 현재 수행하는 액션 객체 그 자체는 한 번에 하나씩이지만, 그 액션 객체의 하위 이벤트들은 같은 프레임 시간에 여러 개가 수행될 수 있다.

※ 이는 종종 풀기 어려운 문제를 유발하곤 한다.
즉, 여러 개의 액션 이벤트가 같은 시간에 수행 가능하다는 점 때문에, 각 액션 이벤트들은 서로의 표현 영역을 '침범'하지 않아야 한다는 전제가 필요하기 때문이다.

- **B-8-3-7.** 캐릭터의 현재 동작 중인 액션 객체의 하위에 여러 개의 액션 이벤트들이 동시에 수행 중인데, 그러한 각 액션 이벤트가 수행하는 내용이 '독점적으로' 동작해야 한다고 치자.
이 때의 각 '독점적'인 동작이 어떤 순서로 실행될지는 액션 시스템에서 보장하지 않는다.

※ 여기에 있어 가장 흔한 예는 애니메이션 호출이다.
액션 객체 1개가 애니메이션 1개와 대응해야 하는 이유가 여기에 있다. 하나의 캐릭터 단위가

동시에 2개의 애니메이션을 재생해야 하는 경우는 종종 모순을 낳는다.
한 번에 하나의 애니메이션만 수행할 수 있다는 전제 하에서는, 동시에 2개 이상의 액션 이벤트가 각기 다른 애니메이션을 재생하도록 명령한다면 어떻게 될까? 아마도, 그냥 뒤에 호출된 액션 이벤트의 애니메이션 호출 명령이, 먼저 호출된 액션 이벤트의 애니메이션 호출 명령을 '덮어씌우고' 말 것이다.
여러 애니메이션들을 가중치 합산하는 경우라도, 동시에 2개 이상의 애니메이션 단위를 재생할 수야 있을 것이다. 그래도 원래 액션 이벤트를 디자인할 때 의도했던 바가 아니라는 점에서는 의미상 다를 바가 없다.

◆ B-8-4. 애니메이션 시스템과의 관계

- **B-8-4-1.** 모든 종류의 액션들은 각 액션 객체당 1개의 애니메이션을 소유할 수 있다.

※ 게임에서 하나의 액션이 여러 개의 애니메이션을 '보여주는' 대표적인 경우는 '연타 공격'이다.
그러나 사실, 연타 공격조차도 자세히 보면 **공격 이벤트만 한 액션 내에서 여러 번 들어갈 뿐**, 애니메이션 자체는 하나로 하는 편이 더 낫고, 실제로도 그렇게 제작한다.
만일 그렇지 않다면, 연타 공격을 처리할 때 '완전히 같은 공격 동작으로' 여러 번 애니메이션을 재생하는 방식이 되어야 하는데, 아마도 대개의 그래픽 아티스트들은 연타 애니메이션을 그렇게 '프로그래밍으로' 처리하는 데 대해 동의하지 않을 것이다. (1타, 2타, 3타가 완전히 똑 같은 모션이라면 정말 밋밋하게 느껴질 게 분명하다.)

- **B-8-4-2.** 액션이 소유하는 애니메이션의 재생 요청은 액션의 재생 시간 이내에서 이루어져야 한다.
: 액션 재생 시간 범위를 벗어난 시간에 요청하는 애니메이션 재생은 무시할 수 있다.

- **B-8-4-3.** 애니메이션이 종료되는 시점이 되기 전에 액션 시간이 먼저 종료되는 경우의 처리
: 그 때, 재생하는 애니메이션은 재생이 종료될 때까지 시간이 아직 남아 있더라도, 다음 액션의 애니메이션으로 강제로 재생이 전환될 수 있다.

※ 설계상, 모든 액션 이벤트들은 액션의 재생 시간 안에서 이루어지는 것을 권장한다. 만일 예외적인 연출로 인해 액션 종료 시간을 넘어서 액션 내 이벤트들이 수행되는 경우에는, 다음 액션의 액션 이벤트들에 의해 얼마든지 '잘리거나', 재생 도중에 전환되는 과정을 허용한다.

- **B-8-4-4.** 애니메이션 전환 방식
: 아래 전환 방식들을 같은 인터페이스로 수용할 수 있도록 설계한다.

명칭	설명

즉각적인 전환 (Stop & Play)	• 이전 애니메이션은 즉시 정지하고, 다음 애니메이션은 즉시 재생한다.
부드러운 전환 (Crossfade)	• 엔진 내부의 애니메이션 보간 / 전환 기능에 의해 이전 애니메이션에서 다음 애니메이션으로 전환한다.
전환 애니메이션 (Crossfade Animation)	• 전환 과정을 별도의 애니메이션으로 제작한다.. • 전환용 애니메이션을 액션에 별도로 등록해야 한다. • 액션은 전환 시점이 되었을 때, 전환 애니메이션을 먼저 출력하고 다음 액션으로 넘어가게 만든다.. • 전환 애니메이션이 액션마다 다른 경우를 대비해서, 전환 애니메이션을 직접 등록하지 않고, 일종의 전환 테이블의 키로써 등록하는 방식으로 설계한다.

※ 전환 애니메이션을 두는 방식은 워낙 특수한 동작이라서, 아무래도 액션 시스템에 대해 예외적인 구현을 많이 해야 할 수도 있다.
그래서 경우에 따라서는, 전환 애니메이션의 이벤트가 아닌, '전환 액션' 단계로 두는 방식을 고려해 봄직하다.

※ 별도의 전환용 애니메이션을 통해 애니메이션을 전환하는 기능은, 향후에 구현할 수 있는 여지를 주되, 가장 나중에 구현한다.
현재 프로젝트에서는 전환용 애니메이션을 별도로 두는 경우를 고려하고 있지 않기 때문이다.

- **B-8-4-5.** 애니메이션의 재생 속도와 액션의 재생 속도는 동일하지 않을 수 있다.
: 각 액션은 **'1 배속 액션'**으로 지정된 애니메이션의 속도를 가진다. 이 때 애니메이션 속도도 반드시 1 배속일 필요는 없다. 애니메이션은 2 배속인데 액션은 1 배속으로 칠 수도 있고, 애니메이션이 0.5 배속인데 액션은 1 배속으로 사용할 수도 있는 것이다.

※ 액션의 속도와 애니메이션의 재생 속도가 완전히 결합되어 있다면, **특정 액션의 속도에 대해 애니메이션 속도가 일부 맞지 않는 경우, 애니메이션 작업 자체를 다시 해야** 하는 부작용이 있다.
물론, 자연스러움을 위해 어느 정도는 기본적인 액션 속도(공격 속도, 이동 속도 등)에 맞춰서 애니메이션 키를 잡을 것이다. 그렇지만, 제작 과정에서의 액션 속도 수정이라든가, 게임 기본 기능인 각종 속도 관련 버프 / 디버프를 받게 되는 경우에 그 때마다 애니메이션 속도를 맞춰서 다시 잡는 것은 불합리할 것이다.

사실 이 문제는 애니메이션의 재생 속도뿐 아니라, **시각 / 청각 이펙트의 재생 속도 문제에 있어서도 마찬가지로 적용**하여야 한다. (물론, 일반적인 경우에 사운드를 1 배속이 아닌 다른 속도로 재생하는 경우는 거의 없지만 말이다....)

- **B-8-4-6.** 그래픽 저작 도구(3DS Max)에서 애니메이션 FPS(초당 프레임) 수

: **애니메이션의 FPS 는 Unity3D 엔진에서 기본 변환하는 30FPS 에 맞춘다.**

※ 더 세밀하게 FPS 를 잡는다고 해도, Unity 엔진에서는 최대 30FPS 로 자동으로 변환해버린다. 만일 더 적게 FPS 를 잡는다면, 30FPS 기준에 맞춰서 애니메이션의 키 프레임 숫자를 넣어줘서 추가해줘야 하므로, 매우 번거로운 작업이 되어버린다.

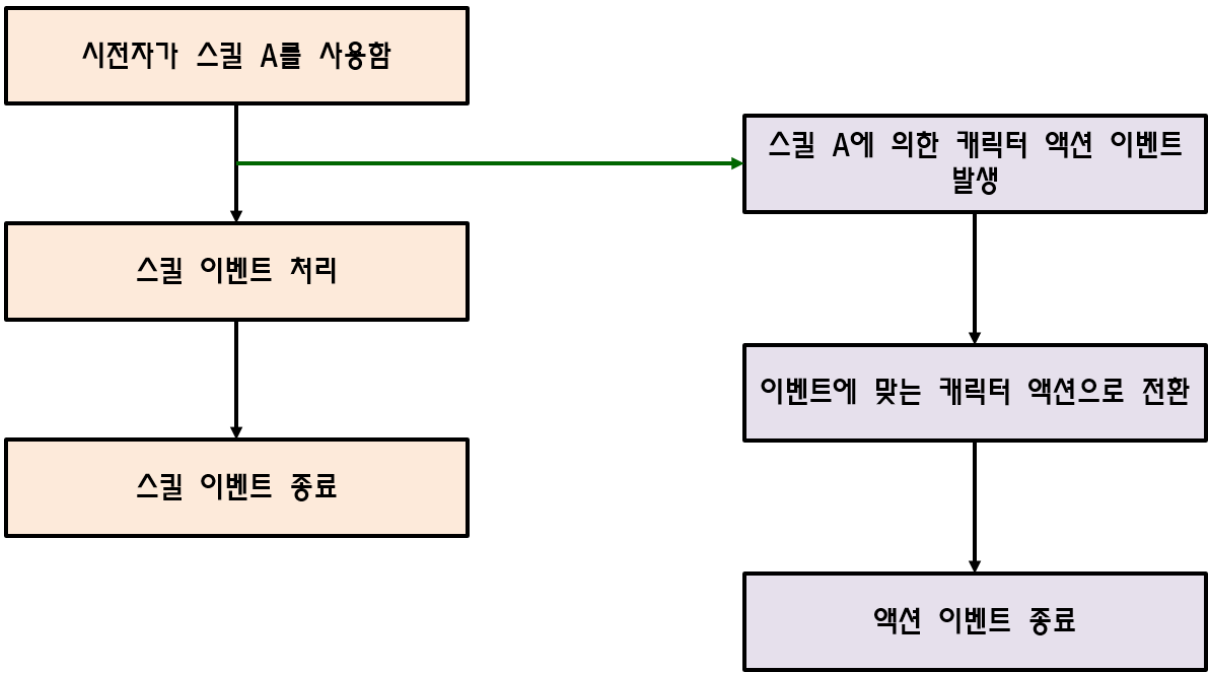
- **B-8-4-7.** 어느 한 애니메이션에서 다른 애니메이션으로 전환할 때, **전환할 다음 애니메이션의 종류에 따라 전환 방식을 다르게 지정할 수 있어야 한다.**
: 애니메이션마다 단 하나의 전환 방식만 소유할 수 있다면, 어떤 경우에는 다음 애니메이션과의 전환 과정이 매우 어색하게 보일 수가 있다.

◆ B-8-5. 캐릭터 스킬 시스템과의 관계

- **B-8-5-1.** 캐릭터 스킬들은 사용하는 과정이 대개 액션을 수행해야 하는 경우가 많으므로, 자주 엮이는 관계이다.

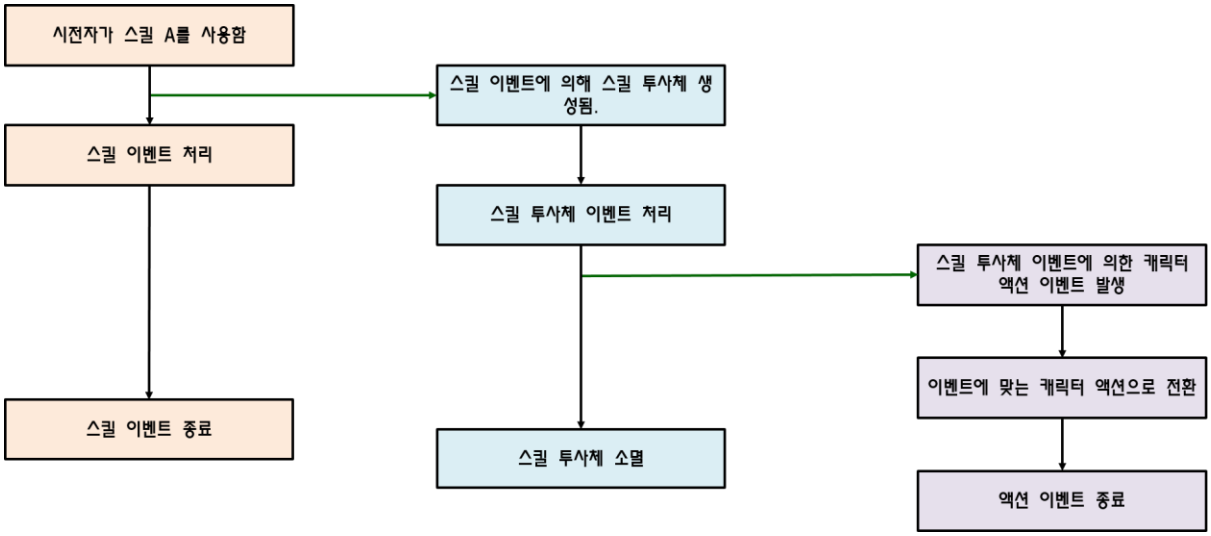
- **B-8-5-2.** 원칙적으로, 스킬의 이벤트 진행 과정과 액션의 이벤트 진행 과정은 관리 주체가 별도로 존재하고, **관리하는 단계와 수명주기도 독립적**이다.
: 액션은 액션 관리자가, 스킬 이벤트들은 각 스킬 인스턴스 객체가 관리한다.

- **B-8-5-3.** 스킬 이벤트가 액션 이벤트를 호출하는 과정



<스킬 이벤트가 액션 이벤트를 호출하는 과정>

- **B-8-5-4.** 스킬 투사체가 액션 이벤트를 호출하는 과정



<스킬 투사체가 액션 이벤트를 호출하는 과정>

※ 위 그림에서 스킬 종료 시점, 스킬 이벤트 종료 시점, 스킬 투사체의 소멸 시점, 액션 이벤트의 종료 시점이 모두 위치가 다르게 그려져 있는 점을 확인할 수 있다.
실제로는 스킬 종료 시점이나 스킬에 의해 발생한 액션의 종료 시점이 같은 시간 간격을 가지고 있을 수도 있고, 밀접하게 관련을 가져야 할 수도 있다.
그러나, 위 그림에서는 색상으로 구분되어 있는 각 이벤트 종류들이, 명백하게 구분된 타입의 객체들이며, 각 종류의 객체들의 수명주기 또한 서로 독립적으로 구동된다는 사실을 묘사하고 있다.

◆ B-8-6. (연구사항)가변적인 재생 시간을 가지는 액션

- **B-8-6-1.** 기본적으로, 액션은 시간 이벤트 기반으로 동작하고, 이러한 사항들을 외부에서 사전에 정의하는 방식이다.
그렇기 때문에, 액션의 길이가 가변적으로 변할 수 있어야 하는 경우가 있다면, 이를 어떻게 구현할지에 대한 이슈가 생기기 마련다.

※ RPG 에서 이에 대한 가장 손쉬운 예는 주문 시전 동작, 그리고 돌진 동작이다.
주문 시전 동작은 시전에 들어가는 시간이 얼마나 되는지에 따라 시전 동작의 반복 횟수가 달라지며, 돌진하는 애니메이션의 재생은 실제 타격하는 대상과의 거리에 따라 애니메이션의 재생 시간이 더 길거나, 짧아질 수 있어야 한다.

- **B-8-6-2.** 가변적인 수행 시간을 가지는 액션에서의 가장 큰 문제점은, 시간 이벤트 단위가 아

니기 때문에, 외부에서 이벤트의 발생 조건과 종료 조건을 통제하기가 대단히 어렵다는 점이다.
: 이런 액션들의 실제적인 수행 / 종료 시간은 그것이 실제 실행될 시점에나 알 수 있고, 시작도 하기 전에는 알 방법이 없다.

- **B-8-6-3.** 또한, 액션이 뭔가 가변적인 상황을 인식하게 만들어야 한다는 점도 문제이다.
: 경우에 따라서는 정확하게 어떤 조건에 의해서 수행 시간이나 발생 / 종료 조건이 변하는지를 판별해야 할 수가 있다. 이런 판별 기능을 만드는 것도 확실히 보통 일은 아니다.

- **B-8-6-4.** 개인적인 생각으로는, 액션에서 이렇게 뭔가를 판단하게 해야 하거나, 가변적인 상황을 다루게 하는 것 자체가 매우 무리하다고 본다.
: 그저, **액션 시스템에서 다룰 수 있는 요소들은 사전에 모든 조건을 통제할 수 있는 경우로 한정하는 게 좋겠다**고 본다.

※ 점프 동작은 뛰어오르려는 동작 -> 공중에 뜬 동작 -> 착지 동작으로 보통 이루어진다.
이 중에서 가변적인 요소는 공중에 뜬 동작이다. 낙하하는 거리가 길다면 공중에 뜬 동작이 재생되는 시간이 길어지고, 낙하 거리가 짧으면 그 시간은 짧아진다. 나머지 동작들은 스크립트로 쉽게 뺄 수 있는, 게임 수행이 시작되기도 전에 이미 모든 조건이 결정된 이벤트만을 가진다.
과연 'Jump' 라는 액션 객체는 공중으로 뛰어오르는 동작, 공중에 뜬 동작, 착지 동작 3 개를 한꺼번에 포괄해야만 할까?
이런 개념이 상당히 유용하긴 하지만, 결국 게임 수행 중에 판단할 수 밖에 없는 요소가 있다면, 이를 외부 XML 스크립트 등으로 빼내어 사전에 조건을 설정하게 만드는 건 아주 어렵고 번거롭다.
그러느니 차라리 이를 가장 잘 처리할 수 있는 곳, 즉 코드 내에 Jump 액션 객체를 구현하는 게 더 낫다. 게임 코드는 가변적인 상황 조건을 설정할 수 있는 모든 내용을 알고 있다.

액션이 꼭 1 개의 애니메이션만 가지도록 하는 제약은 이에서 비롯되었다.
애니메이션이 여러 개일 수 있고, 심지어 이들 중에 뭔가를 선택해야 하는 상황이라면, 이는 사실 액션 객체가 다룰만한 범위가 아니라는 뜻이다. 액션 객체가 다루는 상황의 범위는 외부 스크립트에서 다룰 수 있는 정도여야 하고, 그러려면 게임이 실행되기 이전에 설정할 수 있는 조건들을 가지고 있어야 한다. 즉, 훨씬 더 단순한 임무를 맡아야 한다.

여러 개의 애니메이션을 재생하는 액션은 사실 AI 의 한 상태에 더 가까우며, 같은 액션으로 분류했던 각 애니메이션들은 애니메이션마다 별도의 담당 액션 객체가 있는 게 더 맞다.
액션 객체는 애니메이션을 wrapping 하여, 사운드, FX 등의 시간 이벤트와 결합하려는 게 가장 큰 목적이기 때문이다. 뭔가의 '판정'을 코드 밖에서 하려는 시도는 확실히 무리가 많이 갈 것으로 보인다.

B-9. 전투 시스템

◆ B-9-1. 전투 피해의 판정 공정

- **B-9-1-1.** **일반 공격의 전투 피해 처리 순서**는 아래와 같다.

1. 공격자의 공격 가능 여부 판정
: 공격자가 공격이 불가능한 상태에 있다면, 전투 프로세스는 공격자의 공격 실패로 종료한다.

2. 공격자의 공격 조건의 유효성 판정
: 공격자가, 일반 공격 스킬에서 허용하는 유효한 조건에서 사용하는지 점검한다. 사용 조건들은 스킬마다 제각각일 수 있으나, 대표적인 경우들은 아래와 같다.
- 유효한 대상을 공격하는지 점검한다.
- 적대 대상이 아닌 경우에는 공격할 수 없다.
- 공격 사정 거리 이내에 대상이 들어오기 전에는 공격할 수 없다.
- 일반 공격일지라도, 소모 비용이 존재할 수 있다. (공격 한 번에 마력을 ○만큼 소모하거나...)

3. 방어자의 회피 여부 판정
: 여기서 회피가 성공하면, 혹은 방어자가 무조건 공격자의 공격을 회피하는 상태에 있다면, 전투 프로세스는 공격자의 공격 실패로 종료한다.
방어자가 회피에 실패하면, 공격자의 공격이 명중한 것으로 판정하며, 방어자는 막기 판정 과정으로 들어간다.

4. 공격 피해량 산출 및 방어도 / 저항력에 따른 피해 감소량 산출
: 공격자는 각 속성별 공격력 취합하고, 방어자는 각 속성별 방어력(이나 저항력)을 취합한다.
이게 가장 기본적인 사항이고, 공격자 / 방어자 간 서로 현재 걸려 있는 부여효과에 따라 특수한 경우를 처리하기도 한다.
예를 들면, 공격 횟수가 많아지면 피해량이 증가한다든가, 방어자가 해당 속성에 대해 피해 면역(Immune) 상태라거나 등등...

5. 공격 피해량에서 방어도 / 저항력에 따른 피해 감소량만큼 피해 수치를 감소
: 피해를 방어할 수 있는 수치가 공격 피해 수치보다 더 높은 경우, 이 시점에서 계산되는 피해량은 0 이 된다.

6. 방어자의 막기 여부 판정

~~: 이 단계에서는 공격자의 공격은 명중한 상태이다. 단지, 방어자가 막기에 성공하면 피해를 줄일 수 있다.~~

~~(여기서의 피해량은 방어도에 의해 감소된 상태의 피해량 임을 명심할 것)~~

~~- 막기 판정 성공 : 피해 수치를 50% 감소시킨다.~~

~~- 막기 판정 실패 : 피해 수치를 그대로 적용한다.~~

7. 기본 공격력에 대한 치명타 적용 여부 판정

: 막기 판정 과정까지 거친 이후의 남은 피해량에 대해 치명타 여부를 적용한다.

- 치명타 성공 : 피해 수치가 70% 증가한다.
- 치명타 실패 : 피해 수치를 그대로 적용한다.

8. 최종 피해량 산출

: 위 과정에서 최종적으로 산출된 피해량을 방어자에게 적용한다.

9. 최종 피해에 의한 사망 여부 판정

: 피해에 의해 생명력이 0 이하가 되면, 방어자는 사망한 것으로 처리한다.

10. 공격 후 부여효과 적용

: 공격에 의한 무력화, 지속 피해, 감속 효과, 생명력 / 마력 흡수 등이 대표적인 예시다.

공격 후 부여효과가 적용되는 대상은 방어자일 수도 있지만, 공격자일 수도 있다. (대표적으로 입은 피해를 공격자에게 되돌리는 유형의 부여효과)

- **B-9-1-2. 선택적 사용 스킬(일반적으로 스킬 버튼을 눌러서 쓰는 스킬)의 전투 피해의 처리 순서**는 아래와 같다.

: 현재 기획상으로는, 일반 공격의 판정 프로세스와 거의 다르지 않다.

1. 시전자(Caster)의 시전 가능 여부 판정

: 시전자가 스킬 시전이 불가능한 상태에 있다면, 전투 프로세스는 시전자의 스킬 시전 실패로 종료한다.

2. 스킬의 사용 조건 유효성 판정

: 시전자가, 스킬에서 허용하는 유효한 조건에서 사용하는지 점검한다. 사용 조건들은 스킬의 종류마다 제각각일 수 있으나, 대표적인 경우들은 아래와 같다.

- 적대 대상에게만 사용할 수 있는지 여부
- 지형 좌표를 선택해야 하는 스킬인지
- 소모 비용(대개 마력)이 충분히 남아 있는지
- 스킬을 사용할 수 있는 사정거리 내에 대상이 존재하는지

3. 대상자의 회피 여부 판정

: 일반 공격이 아닌 스킬도 회피가 가능하다.
(이 부분에 대해서는 좀 더 논의가 필요하다. 일단, 현재 기획으로는, 스킬마다 무조건 적중하는 스킬이 있고, 회피가 가능한 스킬이 있다고 함.)
- 대상자의 회피가 성공하면, 혹은 대상자가 무조건 시전자의 시전을 회피하는 상태에 있다면, 전투 프로세스는 시전자의 명중 실패로 종료한다.
- 대상자가 회피에 실패하면, 시전자의 스킬이 대상자에게 명중한 것으로 판정하며, 대상자는 효과 판정(피해를 비롯해) 과정으로 들어간다.

4. 스킬에 의한 피해량 산출 및 방어도 / 저항력에 따른 피해 감소량 산출
: 공격자는 각 속성별 공격력 취합하고, 방어자는 각 속성별 방어력(이나 저항력)을 취합한다.
이게 가장 기본적인 사항이고, 공격자 / 방어자 간 서로 현재 걸려 있는 부여효과에 따라 특수한 경우를 처리하기도 한다.
예를 들면, 공격 횟수가 많아지면 피해량이 증가한다든가, 방어자가 해당 속성에 대해 피해 면역(Immune) 상태라거나 등등...

5. 공격 피해량에서 방어도 / 저항력에 따른 피해 감소량만큼 피해 수치를 감소
: 스킬이 방어력이나 저항력을 무시한다면, 이 부분은 건너뛴다.
피해를 방어할 수 있는 수치가 스킬의 공격 피해 수치보다 더 높은 경우, 이 시점에서 계산되는 피해량은 0 이 된다.

~~**6. 방어자의 막기 여부 판정**~~
~~: 방어자가 막기 판정을 수 없는 스킬인 경우, 이 부분은 건너뛴다.~~
~~이 단계에서는 시전자의 스킬 공격은 명중한 상태이다. 단지, 방어자가 막기에 성공하면 피해를 줄일 수 있다.~~
~~(여기서의 피해량은 방어도에 의해 감소된 상태의 피해량 임을 명심할 것)~~
~~- 막기 판정 성공 : 피해 수치를 50% 감소시킨다.~~
~~- 막기 판정 실패 : 피해 수치를 그대로 적용한다.~~

7. 기본 공격력에 대한 치명타 적용 여부 판정
: 치명타를 적용하지 않는 스킬인 경우, 이 부분은 건너뛴다.
막기 판정 과정까지 거친 이후의 남은 피해량에 대해 치명타 여부를 적용한다.
- 치명타 성공 : 피해 수치가 70% 증가한다.
- 치명타 실패 : 피해 수치를 그대로 적용한다.

8. 스킬 효과 적용
: 이제까지 처리 결과에 의해 결정된 최종적인 스킬에 대한 효과와 판정값들을 실제로 대상에게 적용한다.
스킬에 대한 효과 적용은 스킬마다 정의한 내용이 각각 다르므로, 이 부분은 구현 과정에서 반드시 다형적으로 처리해야 한다.

9. 스킬의 효과 적용에 의한 사망 여부 판정
: 스킬 효과에 의해 어떤 이유로든 생명력이 0 이하가 되면, 방어자는 사망한 것으로 처리한다.

10. 스킬 적용 이후 처리
: 스킬 중에는 목표 대상에게 실제로 적용된 최종 결과값을 가지고 뭔가 다른 효과를 주는 경우가 존재한다. (생명력 흡수, 마력 흡수 등이 대표적이며, 그 외에도 기획에 따라 다양한 형태가 존재할 수 있다.)
스킬에 이런 기능이 필요 없는 경우에는 건너뛴다.

- **B-9-1-3.** 어떤 스킬이라도(일반 공격 스킬을 포함해서) 스킬의 기획에 따라, 스킬을 사용하는 캐릭터의 능력치에 의존한 판정 값을 가질 수 있다.
스킬 별로 판정 값을 어떻게 적용할지 여부는 관련된 기획서의 내용을 따른다.

※ 이는 RPG에서는 꽤 일반적으로 등장하는 방식이다.
예를 들자면, 일반 공격 '스킬'들은 대개 전사라면 힘, 궁수나 도적이라면 민첩, 마법사라면 지능에 의존해서 가할 수 있는 피해 값이 증가한다.
일반 공격 자체도 여러 가지 스킬의 한 종류임을 감안하면, 일반 공격이 아닌 다른 선택적 사용 스킬들 역시 마찬가지로 작동할 수 있어야 함을 알 수 있다.
그렇기 때문에, 스킬 판정을 구현하는 경우, **스킬의 판정 값이 고정된 상수만을 사용할지, 아니면 캐릭터의 특정 능력치에 의존할 수 있는지를 구분해서 처리해줄 수 있게 설계**해야 한다.

- **B-9-1-4.** 피해 판정 프로세스에서, 정상적으로 단계를 거치더라도, 현재 단계의 결과값이 실질적으로 다음 단계를 진행하는 데 있어 의미가 없는 경우에는 다음 단계를 생략할 수 있다.

※ 대표적인 경우는 공격자의 피해량을 방어력으로 감쇄했는데, 방어력으로 감쇄하는 수치가 더 높아서 실질적인 피해가 0인 상황이다. 또는, 이미 생명력이 최대치인 대상에게 생명력 회복 마법을 쓰려고 하는 경우도 들 수 있다.
위 상황들은 스킬의 결과 판정 공정의 전체를 거칠 필요가 없는 경우이다.
이미 방어력에 의해 피해가 0인 상황에서는, 막기 판정으로 피해량을 감소시킬 수 있는지, 치명타로 피해량을 늘일 수 있는지는 아무런 영향이 없다. 어차피 뭘 해도 다 0이다. 이럴 때는 불필요하게 (성능을 잡아먹어가며) 판정 프로세스를 진행할 필요 없이. 곧바로 결과를 돌려주도록 처리한다.

◆ B-9-2. 전투 목표를 설정하는 방식

- **B-9-2-1. 직접 선택한 목표 대상 공격**
: 미리 타격 목표를 설정해두고, 그 목표를 공격하면 **목표로 삼은 개체만 공격 대상**이 된다.

- **B-9-2-2.** 타격할 목표를 직접 선택하는 방식은, 사용자의 조작에 의해 선택하는 경우와, 게임이 진행되는 상황에서, 내부 인공지능에 의해 자동적으로 선택하는 경우를 모두 포함한다.

<대표적인(?) 직접 선택 방식>

- **B-9-2-3. 충돌한 목표 대상 공격**
: **각 스킬마다 판정하는 범위 혹은 궤적이 존재**하고, 이러한 **판정 범위 안에 있는 모든 개체들은 목표 대상**이 되는 방식이다.

- **B-9-2-4.** 충돌한 목표를 대상으로 스킬은 어떤 형태로든, 판정 범위에 대한 내용이 존재해야 한다.

※ 판정하는 형태는 원형일 수도, 사각형일 수도 혹은 그 외 특별하게 정의한 방식일 수 있다. 대개는 가장 연산 비용이 저렴한 원형(혹은 구체 형) 판정 방식을 사용한다.

※ 충돌 방식으로 목표를 정하는 경우에는, 전투할 때 '타겟을 잡는다.'는 특별한 행위는 없다. 그냥 목표를 향해 방향을 돌리고, 무기나 전투 스킬의 사거리 이내에 들어오도록 접근한 뒤, 공격하는 식이다. 회피할 때는 반대로, 적의 공격 방향이 아닌 방향으로 이동한다.

<판정 범위 안에 들어가면 공격 목표가 된다.>

◆ B-9-3. 명중 / 회피 판정 방식

- B-9-3-1. 즉시 명중 / 회피 판정

: 스킬을 사용하는 즉시, 명중 / 회피를 판정한다.
즉, 스킬 사용 시점과 판정 시점이 같다.

1. 전투 피해 처리 공정에서의 회피 단계를 성공하지 못했다면, **그 외 어떤 방법으로도 명중을 피할 수 없다.**

2. 일단 이 시점에서 명중 판정이 났다면, 피해를 즉시 주지 않더라도, **결국은 반드시 피해를 적용**한다.
: 이런 경우에 피해 적용이 시점이 늦어지는 이유는, 그냥 연출상의 자연스러움을 위한 장치에 불과하다.

3. 판정 범위에 대한 정보를 가질 필요가 없다.

4. 판정 비용이 가장 저렴함. (지속적으로 감시할 사항들이 없고, 모든 결과가 즉시 나오므로)

- B-9-3-2. 최종 위치 기반 명중 / 회피 판정

: 스킬의 효과를 적용해야 하는 시점에 스킬의 명중 / 회피 여부를 판정한다.
스킬의 사용 시점과 명중 / 회피를 판정하는 시점은 다르지만, 판정 자체는 1회만 한다.

1. 판정 범위에 대한 정보를 보유해야 한다.

2. 명중 / 회피 판정에 대한 범위는 스킬을 사용하는 시점에서 구체적으로 값이 정해진다.

3. 하지만, 명중 / 회피 판정은 **효과를 적용시켜야 할 시점에서 1회만 수행**한다.

4. 대상자는 비록, 스킬 사용 당시에는 판정 범위에 들어 있었던 상태였더라도, 판정 수행 시점에서 판정 범위 밖으로 벗어나 있으면 피해가 적용되지 않는다. (즉, '수동 조작으로' 회피한 셈이다.)

5. 만약, 반대로 스킬을 막 사용했을 시간에는 스킬의 명중 범위 밖에 있었더라도, 스킬의 명중 판정을 수행해야 하는 시점에 스킬의 명중 범위 안으로 들어온 상태인 경우라면 스킬에 명중한 것으로 간주한다.

6. 스킬의 사용 시점과 판정 시점 사이에 스킬의 경로로 '끼어들더라도' 스킬의 적중은 일어나

지 않는다.
: 왜냐하면, 이 방식은 **명중 판정이 일어나는 특정 시점에 딱 1번만 점검**하기 때문이다. 중간에 끼어들었을 때 적중이 일어날 수도 있게 하려면 **충돌 기반 명중 / 회피 판정 방식**을 사용하여야 한다.

※ 이런 방식은 즉시 명중 / 회피 판정을 할 때보다 뭔가 '사용자가 컨트롤할 수 있는' 여지를 더 많이 부여하는 것처럼 연출할 수 있다.
특히나 스킬의 사용 시점과, 명중 / 회피 판정이 필요한 시점의 시간 간격이 길면 길수록 더욱 명확하게 차이를 드러내줄 수 있다. 그러면서도, 후술할 충돌 기반 명중 / 회피 판정 방식보다 수행 비용을 훨씬 저렴하게 구현할 수 있다.

- **B-9-3-3. 충돌 기반 명중 / 회피 판정**
: 스킬을 사용한 시점부터, 스킬의 효과가 소멸할 때까지 매번 판정 주기마다 목표 대상의 스킬 명중 / 회피 여부를 검사한다.

1. 판정 범위에 대한 정보를 보유해야 한다.

2. 스킬 판정 객체는 조건에 의해 사라지기 전까지 지속적으로 판정 범위에 어떤 대상들이 명중하거나, 혹은 회피하였는지 여부를 감시한다.

3. 이 방식은 모든 명중 / 회피 판정 방식 중에 **가장 수행 비용이 비싸다.**
: 다른 방식들은 명중 판정을 '대상 1회' 또는 '가능한 대상 목록 점검 1회' 수준에서 끝내지만, 이 방식은 스킬 효과가 소멸할 때까지 가능한 대상 목록들을 계속해서 점검하기 때문에 판정 비용이 가장 비쌀 수 밖에 없다.

4. 판정을 수행하는 특정한 주기가 존재할 수 있다.
: 매 Frame마다 판정하는 경우, 판정의 정밀도는 가장 높지만 수행 비용은 극도로 비싸진다.
따라서, 수행 성능을 위해서 '초당 ○○번' 수행하는 식으로, 일부러 수행하는 횟수를 제한하여 판정할 수도 있다.

- **B-9-3-4.** 충돌 기반으로 피격 목표를 설정할 때, 피격 범위 내에 있는 대상들의 전부 혹은 일부만 피격으로 판정하는 경우도 가능하다.

※ 쉽게 말해, 넓은 범위를 공격하는 스킬인 경우라도, 피해 대상을 1 개체, 혹은 지정된 수만큼으로 제한할 수도 있다는 뜻이다

- **B-9-3-5.** 피격 대상들의 개수를 제한하는 조건은, 각 스킬의 세부 구현에 따라 다르게 설정할 수 있다.
각 스킬 별로 구체적인 내용 설정은 관련된 기획서의 내용을 따른다.

※ 어떤 스킬은 범위 내에 최초로 들어온 적만 피해를 받는 스킬일 수도 있고, 또 다른 스킬은 방사 피해이긴 하지만 최대 6 개체까지만 피해를 주는 개체일 수도 있다. 이러한 제한 조건들은 스킬을 구현하기 나름이다.

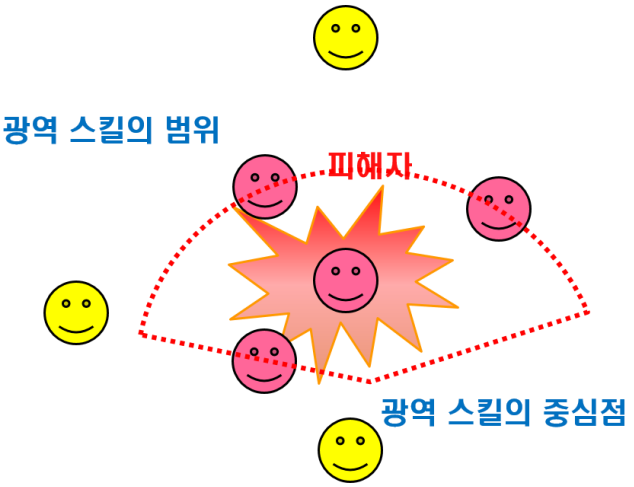
- **B-9-3-6.** 특정 각도에 대한 판정
: 단일, 혹은 여러 목표를 한 번에 판정하는 경우에, 특정한 각도 범위 안에 들어야 유효하다고 판정해야 할 경우에 사용한다.

※ 예를 들자면, 적이 내 캐릭터 전방에 위치해야만 적중할 수 있는 스킬을 구현해야 한다고 하자. 그리고 스킬은 방사형 피해를 줄 수 있다고 가정한다.
그렇다면, 그 대상이 각도상으로 봤을 때, 내 캐릭터의 '뒤에' 위치해 있다면, 명중 판정이 나서는 안 된다.

※ 원형(또는 구체 형태)으로 동작하는 방사 피해는 사실 특정 각도 범위 내에서 좌우 180도 이내에 목표가 위치할 때의 판정과 같다고 할 수 있다.
하지만 구체에 대한 판정을 할 때 굳이 각도를 계산할 이유가 없다. 각도 판정은 내적과 코사인 값을 구하는 계산이 반드시 필요한데, 그저 거리 비교하는 정도 계산에 비해 계산 비용이 많이 비싸다.
그러므로, 각도 판정에 관련된 계산은 반드시 필요한 경우에만 하도록 제한하는 설계를 한다.
(어떤 설계라도 이런 원칙은 마찬가지겠지만 말이다.)

- **B-9-3-7.** 판정 범위에 대해서 각도를 주는 기능은 목표를 직접 지정하는 경우와, 충돌 기반으로 목표를 탐지하는 경우에 대해 둘 다 사용 가능하다.

- **B-9-3-8.** 각 스킬 별 판정 범위에 대한 설정은 관련된 기획서의 내용을 따른다.



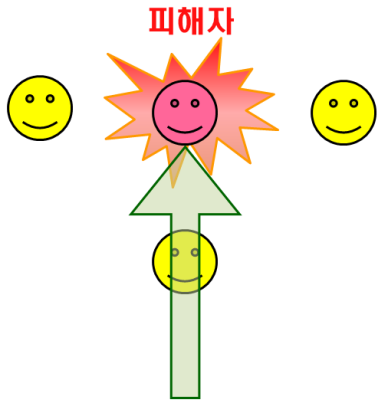
<범위가 충분해도, 각도에서 벗어나 있으면 명중으로 판정하지 않는다.>

◆ B-9-4. 피해(효과 적용) 범위 방식

※ 전투 시스템이기 때문에 이해하기 쉽게, 전투 스킬에 의해 피해를 받는 상황만을 가정했다.
사실은 이로운 스킬 / 해로운 스킬과 상관없이, 스킬의 효과가 적용되는 방식 전반에 대해 같은 원리를 적용할 수 있다.

- B-9-4-1. 선택 피해(선택 효과 적용)

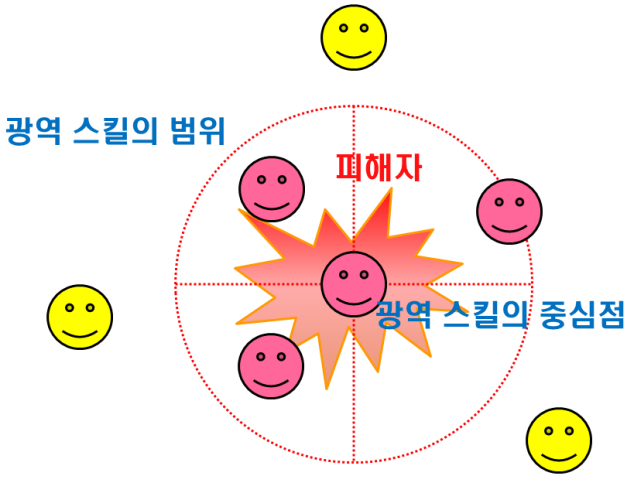
: 정확히 목표로 선택한 대상들만 피해를 입는다. (또는 효과를 적용 받는다.) 이때, 선택한 대상은 하나일 수도 있고, 여럿일 수도 있다.



<선택 피해>

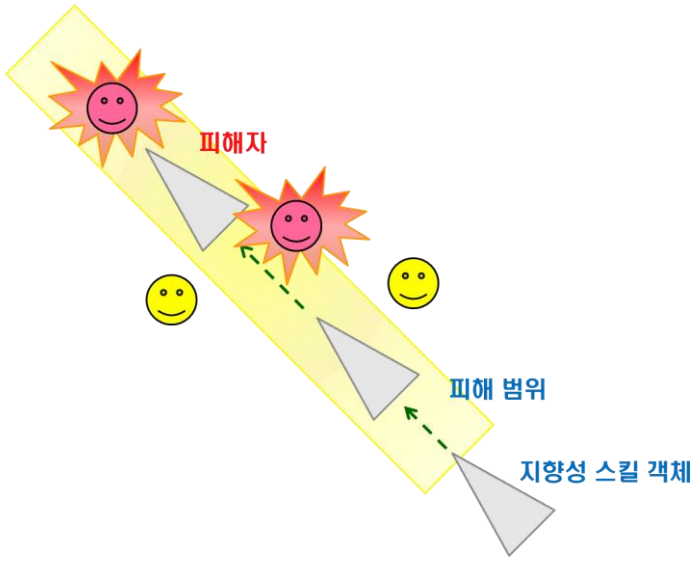
- B-9-4-2. 방사 피해(방사 효과 적용)

: 판정 범위 내에 있는 모든 대상 목표들이 피해를 입는다. (또는 효과를 적용 받는다.



<방사 피해의 예>

- **B-9-4-3.** **궤적 피해(궤적 효과 적용)**
: 판정 방식은 방사 피해의 판정 방식과 같다.
하지만 판정 객체가 지속적으로 움직이고 있기 때문에, 피해를 받을 수 있는(또는 효과를 적용 받는) 범위의 공간상 좌표는 매 순간 달라지고 있을 수 있다.
방사 피해의 변형된 형태라고 할 수 있다.



<궤적 피해의 예>

※ 궤적 피해(궤적 효과 적용)를 주는 객체가 공간 좌표 상에서 이동을 하지 않는다면, 효과를 적용 받는 범위의 공간 좌표는 변하지 않는다. 그러나, 궤적 피해를 주는 객체가 이동 중이라면 효과를 작용 받는 범위의 공간 좌표는 그 객체가 이동하는 궤적을 따라다니면서 변할 것이다.
즉, 궤적 피해의 효과 범위는 **그 피해를 주는 주체가 되는 객체의 현재 속도(속력 + 방향)와 지속 시간에 따라 결정된다**고 보면 된다.

◆ B-9-5. 전투 시의 제약 사항들

- **B-9-5-1.** 원칙적으로 전투에 사용하는 일반 공격 / 선택적 사용 스킬들의 액션이 수행되는 동안에는 사용자의 조작으로 움직이지 못한다.

※ (움직이다가) -> (서서) -> (스킬 사용) -> (다시 움직임) 의 개념으로 전투할 때의 액션들을 처리한다.

- **B-9-5-2.** 전투 액션 중일지라도, 스킬의 효과에 의해 캐릭터의 위치가 바뀌는 경우는 가능하다.

: 즉, 플레이어가 직접 의도한 지점으로 캐릭터를 움직이려고 하는 행위만 제한한다.

※ 대표적인 경우는 강한 공격에 의해 캐릭터가 강제로 뒤로 밀려나거나, 공중에 한 번 띄워졌다가 떨어지는 상황이다. 이는 플레이어가 이동하려는 의도와는 무관하고, 스킬의 효과 중 하나이기 때문에, 스킬 액션을 취하고 있는지 여부와 상관없이 이동을 적용한다. (사실 이럴 때는 이동이라고 하기보다는, 위치가 변했다고 하는 게 더 맞는 표현이다.)

◆ B-9-6. 피해 속성

- **B-9-6-1.** 피해 속성의 종류
: 일반, 물리, 마법, 화염, 빙결, 전격, 맹독

- **B-9-6-2.** 여러 개의 피해 속성이 있는 경우, 각 피해량 계산은 속성 별로 연산한다.
: 모든 속성별 피해량을 합친 뒤에 피해량 관련 계산을 하면 안된다.

- **B-9-6-3.** 캐릭터의 현재 공격력이나 방어력 수치를 이용해야 하는 연산인 경우, 속성별 공격력의 총합과 속성별 방어력의 총합을 이용한다.
: 이는 어떤 관점에서 보면 완전히 정확한 계산은 아닐지 모르지만, 기본적으로 피해 속성 타입이 있는 능력 종류 -> 피해 속성 타입이 없는 능력 종류로의 변환이기 때문에 이와 같은 방식을 허용한다.

※ 최종 피해량에 대한 후처리로써 이용해야 하는 경우는 대표적으로, 생명력이나 마력을 흡수하는 기능을 들 수 있다.
그런데 생명력이나 마력 둘 다 일반 피해니, 화염 피해니 하는 **피해 속성 타입과는 아무런 관련도 없는 능력 종류**다. 그러면 이럴 때, 최종 피해량의 상수 XXX만큼, 혹은 ○%만큼을 흡수해야 하는지는 어떻게 판단해야 할까? 이런 상황에서는 특정 피해 타입을 명시적으로 선택할만한 뚜렷한 근거가 없다.
결국 가장 공평한(?) 방법은 모든 피해 속성의 피해량을 다 더한 다음, 그 합산된 피해량을 가지고 뭔가 후처리를 하는 방법이다.

- **B-9-6-4.** 피해량을 1가지의 간단한 방식으로 표시해야 하는 경우, 모든 속성 별 피해량을 전부 더해서 표기한다.
: 속성별 피해량을 각각 표기하는 방식은 '정확하지만' 곧장 인지하기는 어렵다.

◆ B-9-7. 치명타에 대한 처리

- **B-9-7-1.** 치명타를 적용할 때는, 스킬마다 아래 내용들을 구분해서 조절할 수 있어야 한다.

1. 치명타 적용 여부
 : 스킬마다 치명타가 가능한 스킬과 가능하지 않은 스킬을 구분한다.

2. 치명타 적용 방식
 : 치명타를 계산하는 방식은 스킬마다 다를 수 있다. 고정적일 수도 있고, 능력치에 따라 다를 수도 있다.

3. 치명타 적용 계수
 : 치명타의 계수가 변화해야 할 가능성도 있다. (어떤 경우에는 치명타가 1.5배 피해를 주겠지만, 어떤 경우에는 3배의 피해를 줘야 하는 경우가 생길 수도 있다.)

4. 치명타 값에 대한 최소 / 최대 값
 : 필요한 경우, 치명타 값의 최소 / 최대 값 제한을 두고 그 범위를 넘지 못하도록 조절해야 할 수 있다.

※ 예를 들면 '치유' 스킬에 대해서는 증폭될 경우, 본래 100 의 생명력을 채워주는 것을 200 의 생명력을 채워주도록 구현할 수 있다.
 또 다른 예로는, '후려치기'라는 스킬을 구현할 때, 증폭될 경우, 본래 1 초 동안 상대를 무력화시키는 것을 2 초 동안 무력화하도록 구현하는 경우다.

 이처럼 스킬마다 증폭이 적용된 경우의 효과 증가에 대해서는 스킬의 구현 나름이기 때문에, 특히 이 부분에 대해서 유연하게 확장할 수 있도록 구현 코드를 정의해야 한다.

- **B-9-7-2.** 치명타를 적용할 때, (그게 일반 공격이든, 선택적 사용 스킬이든...) **치명타에 치명타를 중복해서 적용하지 않는다**.
 : 치명타 200%짜리 부여효과를 두른 상태에서, 일반 공격에 치명타가 나와서 (기본 공격력) × (치명타) × (치명타 200% 부여효과)가 되는 경우와 같은 사항을 인정하지 않는다는 뜻이다.

- **B-9-7-3.** 지속성 부여효과에는 치명타를 적용하지 않는다.
 : 일반 공격으로 인해 발생하는 부여효과인 경우에도 마찬가지로 치명타를 적용하지 않는다. (치명타를 적용하는 피해는 명중한 일반 공격 그 자체의 피해에 대해서만이다.)

※ 예를 들면, 맹독 속성의 공격은 '3 초당 12 씩 피해를 주는' 방식의 부여효과를 적용한다고 가정하자.
 이런 경우, 비록 일반 공격이기는 하지만, 부여효과를 주는 것이기 때문에, 3 초당 12 씩 주는 피해에 대해서는 치명타가 적용되지 않는다.

◆ B-9-8. 전투 기능에 의한 변환(Translation) 처리

- **B-9-8-1. 전진(Go-ahead)**
: 지형 상으로, 최종적인 위치가 걸어 다닐 수 있는 좌표인 경우에 한해서, 스킬을 사용하면 스킬 액션이 진행되는 동안 캐릭터의 위치가 이동하는 게 가능하다.

- **B-9-8-2. 밀쳐내기(Knockback)**
: 대상을 밀쳐낼 때, 대상도 지형 상으로, 최종적인 위치가 걸어 다닐 수 있는 좌표까지만 밀쳐낼 수 있다.

- **B-9-8-3. 끌어당기기(Pulling)**
: 끌어당겨진 대상 객체들이 여럿이라면, 그 대상 객체들은 끌어온 지점을 중심으로 겹쳐져야 한다. 서로 떨어져 있기 위해서 강제로 (물리 연산을 통해)밀어내지 않는다.

※ 물리 엔진 등을 사용해서 점유 공간을 자동으로 찾게 해버리면, 이런 경우에 예상치 못한 문제들이 발생할 수 있다.
그렇기 때문에, 캐릭터 개체들이 한 지점에 겹치지 않게 처리를 하려고 하더라도, 이런 상황에 대해서 효과적으로 처리할 수 있는 방향으로 설계를 해야 한다.
(이런 문제 때문에, 겹침 방지 처리를 Unity 엔진의 Rigidbody 컴포넌트를 이용해 해결하려는 방식은 그다지 좋아 보이지 않는다.)

- **B-9-8-4. 띄워 올리기(Airborne)**
: 공중에 대상을 띄우는 경우, 대상은 공중에 띄워진 동안 어떠한 이동에 관한 조작도 하지 못한다. (이것 자체가 '불능화' 기능의 일종이다.)

- **B-9-8-5. 회전(Rotation)**
: 회전에 대한 가장 큰 제약은 애니메이션 정보에 의해 움직이는 메쉬에 대한 회전을 시도하는 경우이다.
이 경우만 뺀다면, 어떤 객체를 어떤 방식으로 회전하는지 여부는 설계하기 나름이다.

- **B-9-8-6. 크기 변환(Scaling)**
: 대상에 대한 크기를 변환하는 경우, 크기가 변환된 대상에게 붙는 FX들의 크기도 같이 일괄적으로 변해야 한다.
혹은, 각 FX가 대상의 크기를 따를 것인지 여부를 컨트롤할 수 있어야 한다.

◆ B-9-9. 자동적인 전투 목표 탐색(Auto Targeting)

- **B-9-9-1.** 자동적 전투 목표 탐색은 플레이어가 직접 조종하는 캐릭터(PC로 통칭)와 직접 조종

하지 않고 인공지능에 의해 행동하는 캐릭터(NPC로 통칭)가 둘 다 사용한다.
: 사실, NPC의 경우에는 목표 탐색을 프로그램 쪽에서 해 줘야만 하기도 하다.

- **B-9-9-2.** 플레이어가 직접 조종하는 캐릭터들도, 일부 스킬(특히 공격 스킬)들은 사용 명령이 내려지면, 미리 목표 대상을 선택하지 않았더라도, 스스로 알맞은 목표 대상을 탐색하고 그 대상에게 스킬을 사용하게 만들 수 있다.

※ 정교한 수동 조작이 쉽지 않은 모바일 기기들의 특성 때문에, 이런 편의 기능에 대해 기획적으로도 신경을 많이 쓰게 된다.

- **B-9-9-3.** 어떤 스킬들이 자동적으로 전투 목표를 탐색할 수 있는지에 대한 사항은 관련된 기획서의 내용을 따른다.

- **B-9-9-4.** 플레이어가 직접 조종하는 캐릭터의 목표 탐색은 **직접적으로 사용해서 적을 공격하는 스킬에 대해서만 적용**하는 것으로 개발 범위를 제한한다.
: 대부분의 경우, 스킬의 주된 임무는 적대 대상을 공격하는 것이며, 가장 빈번한 임무에 특화된 기능만 개발해서 개발 부담을 줄이고자 한다.

※ 사실, RPG에서 캐릭터의 스킬들은 적을 직접 공격하는 것 외에, 간접적으로 공격을 하는 경우도 있고(함정 설치 같은 것), 아군에 도움을 주는 스킬이 있는 등 목표가 다양할 수 있다.

하지만 자동으로 목표를 탐색하는 주제에 이르게 된다면, 이런 다양한 목표들에 대해 반드시 자동으로 탐색해야 할 필요가 있다고 보기는 어렵다. 사실, 그렇게 일반화해서 구현하기도 어려운 일이라고 할 수 있다.
모든 종류의 스킬 목표에 대해 효과적인 목표 탐색 방식을 구현하는 건 결코 쉬운 일이 아니기 때문이다. 스킬의 목표에 따라 우선적으로 평가해야 하는 기준이 다를 수도 있고, 어떤 대상을 탐색할지에 대한 내용조차도 달라질 수 있다. 그러면서도 사용자가 직접 하는 판단과 최대한 유사한 결과를 내놓아야 한다.

무엇보다도 문제는, 그렇게 잘 만들어낼 수 있다고 해도, 적을 공격하는 경우를 제외하고는 그 멋진 기능들을 사용자들이 많이 활용하기는 어렵다는 점이다. 어차피 이 게임은 적을 공격하는 전투를 반복하는 과정이 거의 대부분이다.
자동차의 자동 주행 기능을 만들 것도 아닌데, 그렇게까지 비용을 들여서 고도로 지능화된 기능의 설계와 구현을 할 당위성은 없지 않을까 싶다.

- **B-9-9-5.** 자동적인 전투 목표 탐색은 **위협도(Threat, Agro)**에 근거해서 이루어진다.
: 위협도가 가장 높은 대상부터 탐색한다. 전투 상황에서 목표를 탐색하는 방식이기 때문에, 위협도에 기반한 탐색 방식이 가장 합리적이다.

◆ B-9-10. 위협도(Threat, Agro)

- **B-9-10-1.** 쉽게 말하자면, 내게 가장 위험한 대상을 찾을 때 쓰는 척도이다.

- **B-9-10-2.** 위협도를 적용하는 대상은 개체는 적대적인 대상이어야 한다.
 : 중립 혹은 우호적인 대상 개체는 위협도를 판단하지 않는다.

- **B-9-10-3.** 위협도에서 고려하는 요소들은 아래와 같다.

1. 거리
 : 거리가 가까운 순서대로 위협도가 높다.

2. 나에게 해로운 영향을 주는 정도
 : 이를 판단하는 근거는 적대 대상이 나에게 주는 피해량(나에게 검과 마법으로 공격할 때의 피해 값)과, 적대 대상이 내게 적대적인 다른 대상에게 이로운 효과를 줄 때(적 개체들끼리 치유 마법을 주고 받을 때)의 수치 값들을 모두 더한 결과 값이다.

- **B-9-10-4.** 스킬 사용 등의 행동으로 위협도 수치를 구체적으로 산출하는 공식은 관련된 기획서의 내용을 따른다.

- **B-9-10-5.** 거리에 의한 위협도는 별도의 위협도 수치를 제공하지는 않는다.
 : 대상 위치가 변경될 때마다 위협도 계산을 새로 하게 만드는 건 낭비라고 생각한다.

※ 게임에서 가장 빈번한 정보 수정이 위치 좌표가 바뀌는 상황이기 때문이다.
 그래서 개념적으로는 거리가 가까울수록 위협도가 높아지지만, 실제 구현 상으로는 **그저 거리에서 가까운 순서대로 공격할 대상들의 목록에 적재**하기 때문에, 가까운 순서대로 공격하러 가게 될 뿐이다.

- **B-9-10-6.** 위협도는 전투 상황에 돌입할 때부터 계산하기 시작하고, 전투 상황이 끝나면 보유한 위협도 정보는 폐기해야 한다.

※ 평상시에도 위협도 정보를 관리하는 것 자체가 낭비에 불과하다.
 또한, 위협도 정보가 어떤 상황에서도 폐기되지 않는다면, 전투 중에 도망쳐 버린 적의 위협도 정보를 계속 유지해야 하는 부담이 생길 수도 있고, 혹은 전술적인 이유 때문에 전투를 다시 버

- **B-9-10-7.** 위협도는 개체의 내부 인공지능과 상태효과에 의해 임의 시점에 초기화하거나 위협 순서를 무시할 수 있다.

: 가끔 위협도를 무시할 수 있는 상황은 게임 플레이 방식을 더 풍부하게 할 수 있다.
이에 대한 상세한 구현 내용은 관련된 기획서의 요구사항을 따른다.

- **B-9-10-8.** 플레이어가 직접 조종하는 상태의 캐릭터는, 거리 외의 다른 요소를 위협도에 반영하지 않는다.
: 그냥 가까운 순서대로 공격할 뿐, 아무런 내부적인 계산을 안 한다는 의미이다.

※ 다만, 플레이어가 직접 조종하고 있으면서도 자동적으로 전투가 계속되고 있는 상황의 캐릭터와, 플레이어가 명시적으로 자동으로 플레이하도록 지정한 상태의 캐릭터가 똑같이 먼저 공격대상에 등록한 순서대로만 적을 공격하는 게 맞는지는 결정이 필요하다.

◆ B-9-11. 처치 콤보(Kill Combo) 기능

- **B-9-11-1.** 연속적으로 적(몬스터, 또는 보스)을 죽이거나 피해를 입힐 때마다 숫자를 세는 기능이다.

- **B-9-11-2.** 콤보 숫자는 바뀔 때마다 플레이어에게 알리거나, 혹은 주기적으로 알릴 수 있다.
: 매 번 카운트가 증가할 때마다 '○○ Combo!!!' 하는 식으로 알릴 수도 있지만, 특정한 숫자(예를 들면 10, 100 등)가 돌아올 때마다 알리는 방식을 쓸 수도 있다.

- **B-9-11-3.** 콤보 기능에 대한 보상 방식은 기획에서 정의한 내용을 따른다.

※ 콤보의 보상이 실제로는 따로 없고, 그냥 '기분상' 보여주는 숫자에 불과할 수도 있다.
아니면 정말로 일정한 콤보 개수를 달성한 경우에 별도의 보너스 경험치, 화폐, 부여효과 등을 제공해줄 수도 있다. 이는 기획에서 정하는 바를 따른다.

- **B-9-11-4.** 처치 콤보의 카운트의 초기화는, 해당 캐릭터 개체가 전투 상태에서 벗어난 뒤, 일정한 시간이 지날 때 초기화한다.
: 아무리 마지막 적을 죽인 뒤 시간이 오래 지났더라도, 계속해서 전투 중이었던 경우에는 콤보 누적을 계속 감시해야 한다. (보스 몬스터와 싸우고 있었는지 누가 알아?)

- **B-9-11-5.** 내 캐릭터가 사망한 경우에도 처치 콤보 카운트는 초기화한다.

※ 그리 중요한 내용은 아니지만, 보통 사망 -> 부활 과정에 걸리는 시간이 있으니까 으레 알아서 콤보 카운트가 초기화 되겠거니 여기는 건, 그다지 좋은 설계는 아니다.
어떤 이유로 인해 콤보 카운트가 초기화되기도 전에 부활해서 콤보 카운트가 계속 이어질 수도 있는 법이다. 이런 경우가 발생하면 누구라도 이상하게 생각할 게 분명하다.

B-10. 트리거 시스템

◆ B-10-1. 트리거 객체의 특성과 제한 사항

- **B-10-1-1.** 트리거들은 조건에 맞는 상황인지를 감시하고 있다가, 조건에 도달한 경우에 사전에 정의한 이벤트를 수행한다.

※ 게임이 돌아가고 있을 때, 다른 프로그램 논리의 실행과 상관없이 지속적으로 조건을 감시해야 하는 경우에 트리거를 이용한다.

- **B-10-1-2.** 트리거 객체들의 조건 검사는 트리거 프로세스 자체의 실행이 허용되는 한, 트리거와 직접 관련이 없는 다른 게임 실행 코드들의 처리 상태와 관계없이 항상 수행한다.

- **B-10-1-3.** **한 번 적재한 트리거 객체는 게임 애플리케이션이 종료될 때까지 소멸하지 않는다.**
: 즉, 트리거를 일회성으로 사용할지, 반복 사용할지 여부는 그 트리거의 조건에 달렸다.

※ 메시지 객체와 트리거 객체의 용도와 성격에서 가장 큰 차이점이 이 부분이다.
메시지 객체는 한 번 생성되고 나서 사용하고 나면, 사라진다. (실제로는 수집기 객체가 수거해서 재활용하지만 어쨌든 개념상으로는 그렇다.) 휘발성이라는 뜻이다.
반면, 트리거 객체는 한 번 트리거 컨테이너에 들어가면 사라지지 않는다.
조건에 따라 트리거 객체도 단 1 회만 사용할 수 있고, 메시지 객체도 반복적인 호출을 통해 여러 번 재활용할 수 있지만, 근본적으로 객체 자체가 '한 번 사용하면 끝나는지'(메시지), '지속적으로 유지되다가 조건에 맞으면 실행하는지'(트리거)에 대한 근본적인 차이가 있다.

- **B-10-1-4.** 모든 트리거들은 게임이 실행되기 전에 배치와 동작 방식을 사전에 정의한 상태로 준비한다.
: 일단 게임이 실행되면, 트리거들은 필요한 시점에 트리거 관리자에 적재될 뿐이며, **이미 트리거 업데이트가 수행되는 도중에 새로운 트리거가 중간에 생성되어서 끼어들 수 없다.**

- **B-10-1-5.** 트리거의 실행은 스위치처럼 켜고 끌 수 있다.
: 트리거 객체는 본래 계속해서 자신이 가지고 있는 트리거의 수행 조건들을 검사하지만, 트리거 실행을 비활성화하면 아예 수행 조건들의 검사 자체를 안 한다.

- **B-10-1-6.** 트리거 객체는 자기 자신, 혹은 다른 트리거의 실행 여부를 결정할 수 있다.

※ 자기 자신을 켜고 끌 때는, 최소한 1번은 실행이 이루어지는 셈이다.

- **B-10-1-7.** 트리거 객체는 특정한 횟수만큼만 실행하거나, 혹은 횟수 제한 없이 지속적으로 실행할 수 있다.
: 만약, 트리거의 실행 횟수가 제한되어 있다면, 제한된 횟수만큼 실행한 트리거는 그 다음부터는 수행 조건들의 검사 자체를 수행하지 않게 된다.

- **B-10-1-8.** 트리거 객체들의 조건을 판단할 때, **트리거 관리자에 적재된 순서를 조건으로 삼아 판단할 수 없다.**
: 즉, 특정 트리거가 먼저 실행되어야만 작동하는 등의 조건을 설정할 수는 없다.

※ 각 트리거 객체들의 적재 / 실행 순서가 다른 트리거 객체의 실행에 영향을 미치지 않는다는 말이다.
트리거 객체들의 실행은 순서에 있어 자유롭다.

- **B-10-1-9.** 프로그램 소스 코드를 수정하지 않으면서도 트리거 기능들의 적재와 편집이 가능해야 한다.
: 트리거의 목적은, 자주 사용하게 되는 일정한 패턴의 코드 논리들에게 구체적인 매개 변수 값들을 전달해야 할 때, 그 작업을 소스 코드 밖에서 편집할 수 있게 만들기 위함이다.

※ 소스 코드에서 모든 트리거 내용들의 구체적인 매개변수 값(literal)들을 매번 편집해야만 한다면, 많은 코드 중복이 일어날 것이고, 값을 수정할 때마다 실행파일도 새로 빌드 해야만 한다.

- **B-10-1-10.** 트리거 데이터의 편집 방식은 최소한 텍스트 기반의 데이터 스크립트를 수정하는 방식이어야 한다.
: 텍스트 편집 기반으로 만드는 게 개발하기는 제일 쉽다.

- **B-10-1-11.** 향후 더 발달한 트리거 저작 도구를 개발하거나 조합하는 방식에 대한 제한은 없다.
그런 도구들을 개발할지 여부는 프로젝트 전체 비용에서 가능한 정도를 평가한 후에 결정한다.

◆ B-10-2. 내부 구조

- **B-10-2-1.** **트리거 ID**
: 트리거의 ID 는 트리거들을 적재하는 컨테이너에서의 인덱스를 ID 로 한다.

※ O(1) 접근 속도를 보장하기 위해, 트리거 객체들의 인스턴스를 관리하는 컨테이너는 일반 배열을 사용한다.
트리거 객체 자체는 수정 / 삭제를 할 이유가 없는 객체이기 때문에, 순회하는 속도가 가장 빠른 자료구조가 좋다.

- **B-10-2-2. 조건부(Condition) 함수 객체들의 컨테이너**
: 트리거의 실행부를 수행해도 좋은지에 대한 조건을 검사하는 임무를 담당하는 함수 객체들의 목록을 소유하고 있다.

- **B-10-2-3. 실행부(Operation) 함수 객체들의 컨테이너**
: 트리거가 담당하는 행동이 구현되어 있는 함수 객체들의 목록을 소유하고 있다.

- **B-10-2-4. 활성화 여부**
: 트리거 객체는 활성화된 상태에서만 작동한다. 활성화되지 않은 트리거 객체는 조건 검사조차도 하지 않고 넘어간다.

- **B-10-2-5. 현재 수행 횟수 / 최대 수행 횟수**
: 수행 횟수가 지정되면, 트리거는 그 수행 횟수만큼만 수행하고 나서, 트리거를 비활성화한다.

- **B-10-2-6.** 수행 횟수가 0이면, 수행 횟수의 제한이 없다고 간주한다.
: 트리거를 0번 수행한다는 게 아무런 의미가 없기 때문에 이렇게 정한다.

◆ B-10-3. 함수 객체

- **B-10-3-1.** 트리거를 구성하는 각 조건부와 실행부는, 소스 코드에서 하나의 함수를 이루는 구성 요소들을 객체의 형태(클래스, 혹은 구조체)로 표현한다.

※ 트리거 함수 객체의 의사 코드로 표현하면 대략 다음과 같다. (조건부 함수 객체로 표현함)

```
// 조건부의 기본 클래스
class TriggerConditionBase
{
    // 함수
    bool abstract Evaluate();

    // 멤버 변수
    ConditionType type;
    int index;
```

```
}

// 기본 클래스의 내용을 포함하는 세부 구현 클래스
class TriggerConditionExample : TriggerConditionBase
{
    // 함수
    bool override Evaluate();

    // 매개변수
    int argument_1;
    float argument_2;
    Something.Type argument_3;
    ...
}
```

- **B-10-3-2.** 조건부의 함수 객체들은 성공 / 실패 여부를 반환하는 공통적인 함수를 가진다.
  : 이 함수의 결과에 따라 해당 조건이 성공했는지, 실패했는지 결정한다.

```
※ 의사 코드로는 대략 아래와 같은 형태가 된다.
  bool ConditionMethod(object[] arguments)
```

- **B-10-3-3.** 실행부의 함수 객체의 수행 함수는 반환값이 없다.

```
※ 의사 코드로는 대략 아래와 같은 형태가 된다.
  void OperationMethod(object[] arguments)
```

- **B-10-3-4.** 인덱스(Index)
  : 조건 객체가 조건 목록에서 몇 번째 조건인지 저장한다.

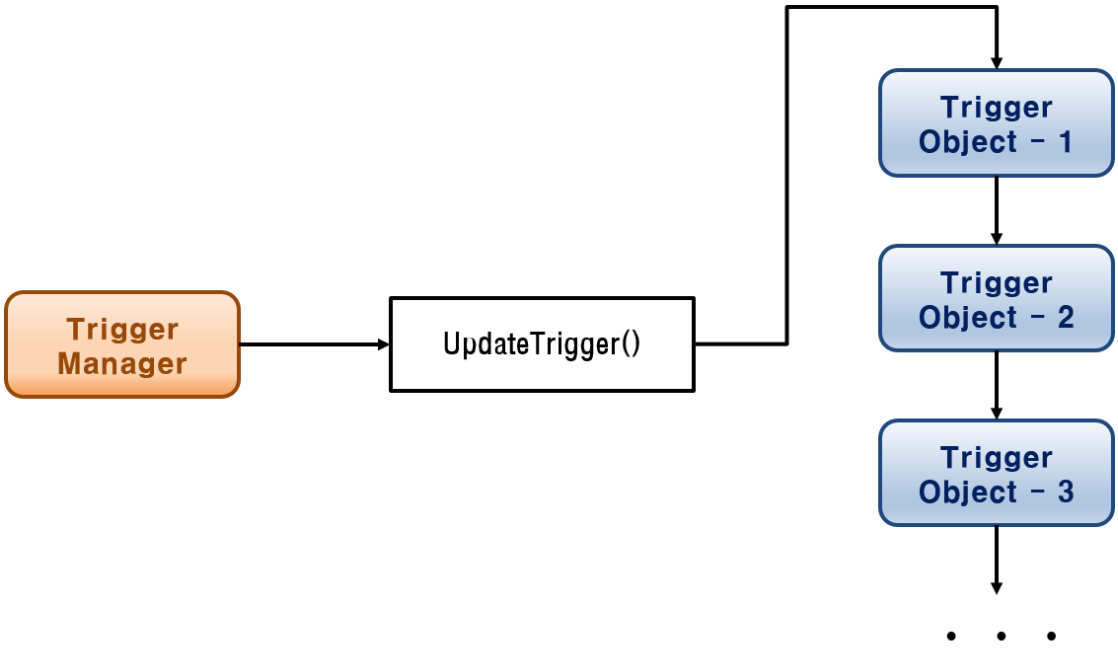
```
※ 조건 목록을 나타내는 배열의 인덱스이다.
```

- **B-10-3-5.** 매개변수 목록
  : 각 함수객체마다 요구하는 매개변수들의 타입과 종류는 트리거의 세부 구현에 따라 달라질 수 있다.

◆ B-10-4. 작동 프로세스

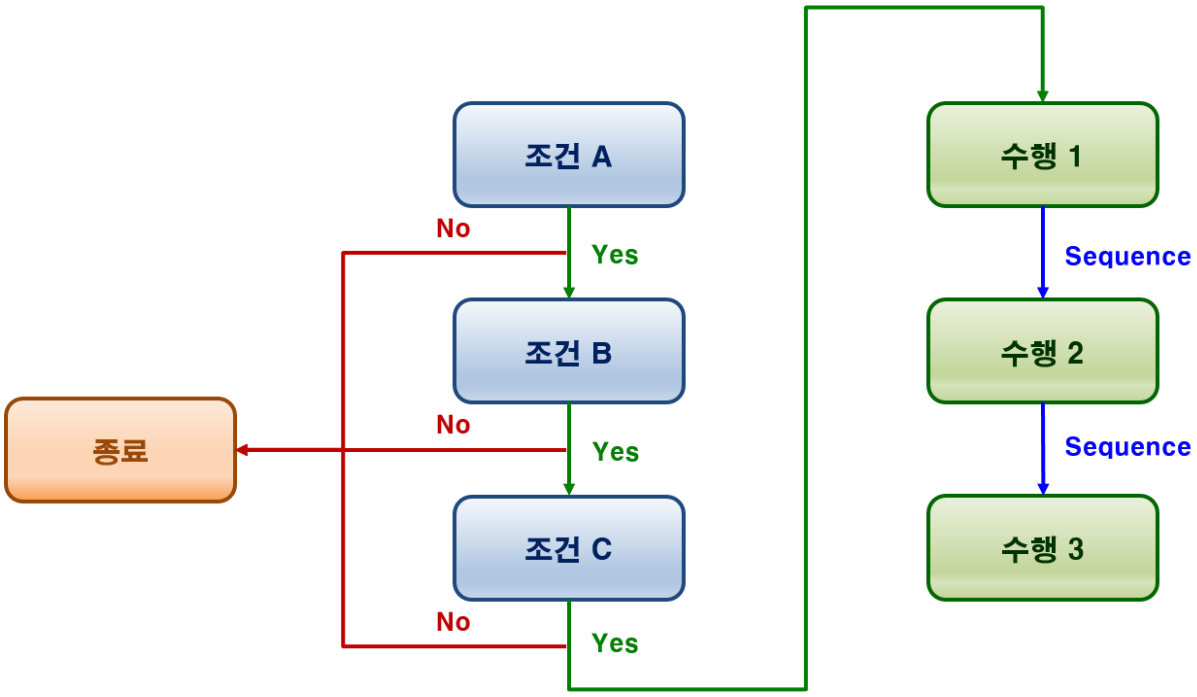
- **B-10-4-1.** 트리거 관리자는, 트리거들의 업데이트 1회 당, 트리거 컨테이너에 들어 있는 트리

거 객체들을 1회씩 순차적으로 내부 절차(조건검사 -> 맞으면 실행하는 과정)를 수행한다.
: **한 번의 트리거 업데이트 단계에서 동일한 트리거 객체를 2회 이상 업데이트할 수 없다.**



<트리거 업데이트 프로세스>

- **B-10-4-2.** 트리거 내부의 조건들을 순차적으로 검사하고, 모든 조건이 맞을 때만 실행부를 순차적으로 수행한다.
: 조건이 하나라도 맞지 않으면, 트리거의 실행부는 전혀 수행하지 않는다.



<트리거 내부 객체의 실행 흐름>

- **B-10-4-3.** 트리거의 업데이트 주기는 게임 애플리케이션의 최대 초 당 프레임 수(Frame Per Second, FPS) 이하에서 원하는 대로 설정할 수 있어야 한다.

※ 1초당 60 프레임을 화면에 그릴 수 있는 상황이라도, 트리거의 업데이트는 1초 당 15번, 30번 하는 식으로 구성할 수도 있다.

- **B-10-4-4.** **각 트리거 별로 업데이트 주기를 다르게 지정할 수 없다.**
: 트리거들의 업데이트 주기는 공통적으로 정한 하나의 기준을 따라야 한다.

◆ B-10-5. 트리거에 사용하는 매개 변수

- **B-10-5-1.** 트리거에 사용하는 매개 변수들은 소스 코드 외부에서 편집한 값으로 전달받아야 하기 때문에, 값 형식과 성격에 한계를 가진다.

- **B-10-5-2.** **참조 타입(reference type), 메모리 주소 값은 트리거의 매개 변수로 사용할 수 없다.**
: C / C++에서의 포인터처럼 메모리 주소 값을 직접 매개변수로 사용하려고 하거나, 참조자, C#에서의 클래스 인스턴스를 직접 매개 변수로 전달하는 행위 등이 불가능하다.

※ 그도 그럴 것이, 트리거를 편집하는 쪽은 소스 코드 외부이며, 이런 값들을 직접 알아낼 방법이 없다. 그리고 그러한 기능을 구현할 생각도 없다. 그걸 구현할 정도로 노력을 할 거면 차라리 소스 코드에 직접 트리거 인스턴스들을 구현하는 게 낫지...

- **B-10-5-3.** 트리거가 호출되는 시점에 가서야 알 수 있는 정보들을 트리거의 매개 변수로 사용해야 할 때는, **사전에 요구하는 사항에 맞는 매크로를 정의해서, 그 매크로를 정의하는 값을 사용**한다.

※ 트리거 업데이트가 호출되는 시점이 되어야 하는 정보들은, 예를 들자면 대개, '내 캐릭터가 현재 선택한 대상', '내 편에 속한 개체들', '개체들 중에서 몬스터들' 같은 식으로 표현할 수 있다.
이러한 매개변수들은 당연히 게임을 실행도 해보기 전에는 알 수 없는 정보들인데다, 심지어 정보를 알아내기 위해서는 또 다른 매개 변수를 전달해야 하는 경우도 있다.

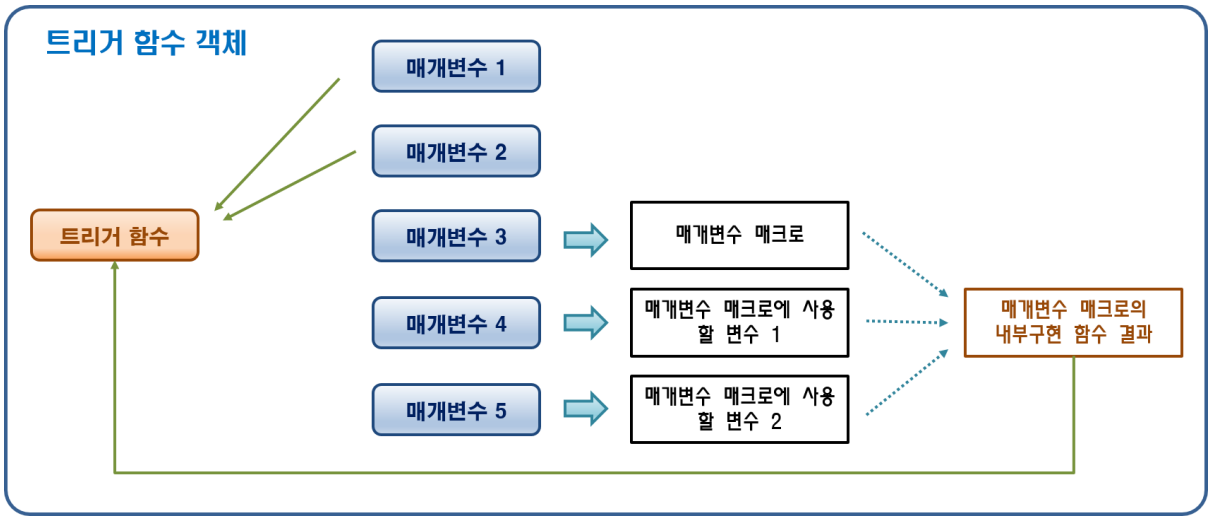
- **B-10-5-4.** 트리거 매개변수 매크로가 실행 코드 내부에서 동작하기 위해 요구하는 매개변수들도, 매개변수 매크로를 이용하는 트리거 함수 객체가 포함해야 한다.

※ 트리거 매개변수 매크로도 프로그램 내부로 따지자면, 트리거 함수 객체가 표현하는 것과 마

찬가지로, 한 개의 함수를 표현한다.
프로그램 코드에서의 함수는, 매개변수가 필요 없을 수도 있지만, 여러 개 필요할 수도 있다.
그러므로, 함수를 표현하는 트리거 매개변수의 매크로도, 자신이 표현하는 함수에 전달할 매개변수들을 관리하는 구조를 가져야 할 필요가 있다. 바로 여기에서 트리거 함수 객체와 매개변수 매크로의 차이가 발생한다.

트리거 함수 객체는 자신이 표현하는 함수의 매개변수들을 직접 관리한다.
하지만, 트리거 매개변수 매크로는 자신이 관리해야 하는 매개변수들의 목록을 매크로 자신이 관리하는 게 아니라, 매크로 자신을 사용하는 입장에 있는 트리거 함수 객체가 같이 관리하게 한다.

세부적인 구현에 대한 관점으로 보자면, 트리거 함수 객체는 클래스나 구조체의 형태로 표현하므로, 매개변수를 멤버로 가질 수가 있다.
그러나 트리거 함수에 전달하는 매개변수로 사용하는 매크로는, 실질적으로는 함수를 호출하게 만드는 기능을 하는 어떤 원시 타입primitive type (int일 수도 있고, enum일 수도 있지만 말이다.)에 불과하다. 클래스나 구조체가 아니기 때문에 매크로를 위한 매개변수들은 매크로의 입장에서는 보관할 방법이 없는 것이다.
다행히도, 매개변수 매크로를 사용하는 트리거 함수 객체는 클래스 혹은 구조체의 형식을 가지고 있다.
또한, 트리거 함수 객체의 목적상, 클래스(혹은 구조체) 하나 당 트리거 함수 딱 한 개만을 표현한다., 이는 트리거 함수 객체가 내부적으로 가지고 있는, 트리거 함수들의 매개변수들을 어떤 식으로 사용할지를 미리 결정할 수 있게 해 준다.
즉, 매개변수 매크로가 필요로 하는 변수들도 같이 멤버로 보관해준다고 해서, 어떤 게 트리거 함수에 직접 전달해야 하는 매개변수이고, 어떤 게 매개변수 매크로를 위해 신 보관해 준, 매개변수 매크로에 사용할 매개변수인지 혼동하지 않을 수 있다는 뜻이다. (만약, 트리거 객체가 더 추상적이어서, 하나의 객체로 여러 개의 트리거 함수를 표현할 수 있어야 했다면 상황이 달랐을 것이다.)



<내부적으로 매개변수를 요구하는 매크로가 있는 경우의 트리거 함수 동작 구조>

※ 개념적으로 정확하게 표현한다고 해서 장황하게 썼는데, 사실 그림으로 보면 그렇게 어려운 개념이 아니다.

위 그림을 보면 알겠지만, 그저 트리거 객체에 멤버 변수로 포함된, 트리거 함수에 전달할 매개변수들의 일부는 트리거 함수가 직접 이용하는 게 아니고, 매크로를 이용해 호출하는 별도의 함수에 전달할 목적을 가진 매개변수일 수도 있다는 뜻이다.

위 그림의 예를 들어보면, 트리거 함수 객체가 표현하는 트리거 함수는 3개의 매개변수를 요구한다고 할 수 있다. 하지만 매개변수는 5개가 들어가 있는데, 이 중에 매개변수 3번은 함수를 호출하기 위한 이름을 정의한 값(매크로)이고, 매개변수 4번과 5번은 매개변수 3번이 호출하는 함수가 요구하는 2개의 매개변수이다. 그 결과로, 매개변수 3번에 의해 호출된 함수가 먼저 실행되고 나서, 그 함수의 결과 값이 트리거 함수 객체가 표현하는 함수에 전달된다.

◆ B-10-6. 트리거의 영역

- **B-10-6-1.** 모든 트리거들은, 그 트리거가 발동되기 위한 트리거 영역을 가지고 있다.

- **B-10-6-2.** 트리거를 발동하기 위해서는, 기본적으로 조건 대상이 트리거 영역에 위치해야 한다.
  : 트리거의 모델을 단순화하기 위한 제한 사항이다.

※ 영역을 가지지 않는 방식의 전역 트리거는 사실, 역설적으로 플레이 가능한 영역 전체를 트리거 영역으로 가지는 트리거라고 볼 수 있다.
또한, 그러한 트리거의 기능은 게임 메시지 전달 시스템으로도 구현할 수 있는 사항이므로, 트리거 시스템에는 트리거만 할 수 있는 고유한 기능에 집중하자는 의미도 있다. (전제 조건이 있으면, 구현은 좀 더 단순할 수 있다.)

- **B-10-6-3.** 트리거 영역은 다음의 상태에 반응해서 각각 이벤트를 발생시켜야 한다.

1, 진입(Enter)
: 객체가 트리거 영역 밖에 있다가 트리거 영역 안으로 들어오는 순간 1 회에 한해 이벤트가 발생한다.

2. 머무름(Stay, Update)
: 객체가 트리거 영역 안에 머무르고 있는 도중에 이벤트가 발생한다.
이벤트의 발생 주기는 매 프레임이 아니라, 특정한 시간 주기일 수 있다.

3. 탈출(Exit)
: 객체가 트리거 영역 안에 있다가 트리거 영역 밖으로 나가는 순간 1 회에 한대 이벤트가 발생한다.

- **B-10-6-4.** 트리거의 영역 모양에 대한 특별한 제약 사항은 없다.
: 단지 트리거 영역 안에 있는지 계산할 수 있는 모양이라면 어떤 모양을 가져도 관계가 없다.

※ 내장된 물리 엔진에서 지원하는 충돌 객체(Collider)의 크기와 형태 그대로 가져다 사용하는 게 가장 쉽고 효과적이다.

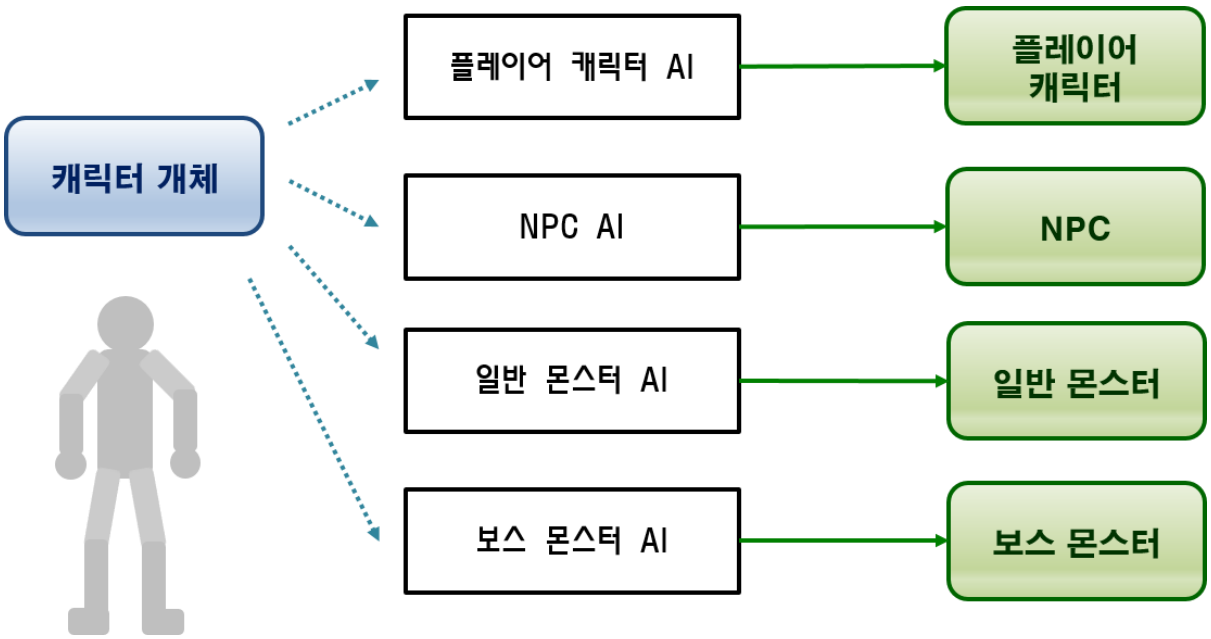
- **B-10-6-5.** 하나의 트리거 영역은 여러 개의 트리거 인스턴스 객체가 이용할 수 있다.
: 트리거 영역은 트리거 객체 그 자체에 종속되지 않는다. 단지, 영역 표시일 뿐이기 때문이다.

B-11. 인공지능 시스템

◆ B-11-1. 구조와 제한사항

- **B-11-1-1.** 각 캐릭터 개체는 인공지능을 관장하는 컴포넌트를 소유하고 있고, 인공지능을 담당하는 컴포넌트에 의해 인공지능의 세부적인 행동들을 수행한다.

- **B-11-1-2.** AI 컴포넌트의 종류는 여러 개로 구성할 수 있다.
  각 AI 컴포넌트의 종류마다 공통적으로 동작하는 부분이 있지만, 표현해야 하는 캐릭터 개체의 성격에 따라 AI 컴포넌트의 세부적인 내용이 달라진다.



<캐릭터 개체의 종류에 따라 달라지는 인공지능 컴포넌트>

- **B-11-1-3.** 인공지능 컴포넌트의 임무는 **캐릭터 혹은 캐릭터 군집의 행동에 대한 판단 뿐이다.**
  : 행동의 시각적 / 청각적 연출 과정을 직접 담당하지 않는다.

※ 캐릭터 행동의 연출은 액션 시스템이 담당한다.
즉, **[AI 에 의해 어떤 행동을 할 것인지 판단하면] -> [액션 시스템이 세부적인 연출을 지정하고] -> [애니메이션, FX 등의 연출 담당 모듈들은, 각 담당 분야 별로 실제 연출을 만들어낸다]**

오해하지 말아야 할 점은, 인공지능 컴포넌트가 상태기계를 통해 행동의 시각적 / 청각적 연출을 담당하지 않는다고 해서, 액션 연출에 대한 선택 권한이 없다고 여기면 안 된다.
인공지능 상태기계는 단지 구체적으로 어떤 애니메이션 키를 재생하고, 어떤 시간에 어떤 FX 를 재생할지 등을 내부에서 직접 호출하지 않는다는 의미일 뿐이다. 인공지능 상태기계 내에서 벌어지는 판단 요소들은, 현재 캐릭터의 액션을 선택하는 근거로 작용한다.

- **B-11-1-4.** **하나의 개체에 대해 한 개의 인공지능 컴포넌트만 작동할 수 있다.**

※ 기능적으로 겹치는 인공지능에 대해서는, '기본 인공지능'과 '확장 인공지능' 방식으로 설계해야 한다.
예를 들어, 일반적인 몬스터들(대개 '잡몹'이라고 부르는)은 상대적으로 단조롭고 평범한 방식으로 플레이어 캐릭터들을 상대한다. 그저, 감시 범위 내에 플레이어가 있으면 쫓아가서 공격하는 정도다. 그렇지만 보스 몬스터의 경우에는 일반 몬스터들의 행동 패턴은 기본으로 가져가면서도 특정한 조건마다(대개 생명력이 어느 정도 소진될 때) 새로운 전투 방식을 들고 나온다. (보통 전투 페이즈가 바뀐다고 표현한다.)
이러한 패턴들은, 보스 몬스터의 AI 컴포넌트가 일반 몬스터의 AI 컴포넌트의 인공지능 기능 구현들을 포함하면서 보스 몬스터 고유의 인공지능 기능을 확장했다고 표현하면 적절하다.

- **B-11-1-5.** 인공지능 기능은, **개체가 생성된 이후부터 언제나 업데이트해야 한다.**
: 화면에 보이는 경우는 물론이고, 화면에 보이지 않더라도 게임에서 뭔가 역할을 수행해야 하는 상태에서는 인공지능 기능이 항상 동작하고 있어야 한다.

※ 인공지능 기능을 '켜고 끄는' 동작은 인공지능 컴포넌트 자체를 활성 / 비활성화하는 게 아니라, 초당 업데이트 주기를 건드리는 방식을 사용할 것.

◆ B-11-2. 유한 상태 기계(FSM : Finite State Machine)

- **B-11-2-1.** 플레이어가 직접 조종하는지 여부와 관계없이, 모든 캐릭터 개체들은 각자 자신의 인공지능에 대한 유한 상태 기계 및 인공지능의 상태 객체들을 소유한다.

※ 플레이어가 직접 조종하는 캐릭터인 경우라도, 캐릭터 행동의 모든 사항을 직접 플레이어가 결정하지는 않는다.
예를 들어, 지점 A -> B로 이동하는 경우에도, 두 지점 간에 거리가 멀다면, 캐릭터 스스로 장애물을 피해서 길을 찾아가는 인공지능이 필요하다. 이는 마치, 사람이 밥을 먹을 때 숟가락을 몇 도 각도로 움직여서, 힘을 얼만큼 줘서 밥을 떠서 입에 넣을지를 직접 계산해가면서 밥을 먹지 않는 것과 같다.

- **B-11-2-2.** 각 상태 관리자 및 상태 객체들은 개별 캐릭터 개체에 소속된 것으로, 캐릭터 개체

와 운명을 같이 한다.

- **B-11-2-3.** 유한 상태 기계의 각 상태와 액션은 분리되어 있는 개념으로 만든다.
: 유한 상태 기계의 각 상태가 취할 행동 중의 일부가 액션 재생이라는 식이다. 또는 재생할 액션을 고른다.

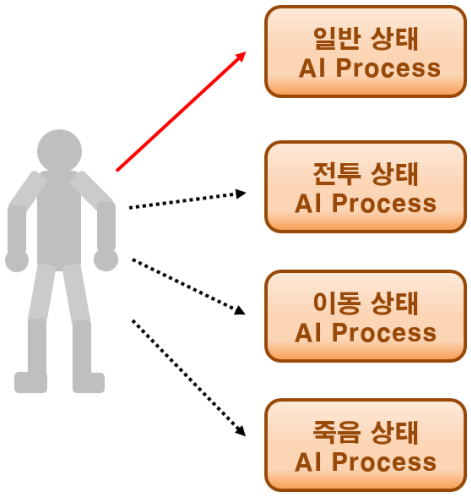
- **B-11-2-4. 상태 기계의 업데이트 주기를 정할 수 있어야 한다.**
: 상태 기계의 타입과, 상태 기계를 소유하는 캐릭터 개체의 타입에 따라 다른 업데이트 주기를 설정할 수 있어야 한다.

※ 내 캐릭터의 경우, 다른 캐릭터 개체들보다 중요하기 때문에 좀 더 자주 AI를 업데이트해야 할 것이다. (그래야 빠릿빠릿하게 움직이지...)
또는, 평화로운 상태와 전투 상태에서의 인공지능 업데이트 주기가 다를 수 있다. 전투 상황에 훨씬 급박하게 돌아가기 때문에 더 자주 AI를 업데이트할 수 있다.
플레이어의 조작이 한참 동안 없는 상태라면, AI의 업데이트 주기를 매우 드물게 할 수도 있다.
플레이어와 멀리 떨어져서 화면에 보이지 않는 NPC들은 AI를 '꺼 놓을' 수도 있다.

- **B-11-2-5.** 캐릭터 개체는 여러 개의 상태 기계를 소유할 수 있다.

- **B-11-2-6.** 캐릭터 개체는 동시에 하나의 상태 기계만을 작동하게 할 수 있다.

※ 각 캐릭터 개체들은 상태 기계 자체를 여러 개 가지고 있을 수 있으나, 현재 상황에 따라 작동시킬 수 있는 상태 기계는 그 중 하나뿐이다.
이렇게 만드는 이유는 구조적인 복잡성을 줄이기 위함이다. 여러 개의 상태 기계를 동시에 업데이트한다면, 상태 기계의 업데이트 순서에 의한 의존성이 생길 수가 있다. AI의 구현 자체가 상당히 구체적이면서도 절차적인 요소가 많은 점도, 복잡하고 병렬적인 구조를 만들면 골치 아픈 이유 중 하나다.



<AI 상태 기계는 여러 개 보유하고 있지만, 한 번에 하나의 상태만 프로세스 한다.>

- **B-11-2-7.** 한 캐릭터 개체가 소유한 AI 상태 기계들은, 단독으로 초기화할 수 없으며, **AI 상태 기계에 대한 초기화를 요청할 경우, 반드시 해당 캐릭터 개체가 소유하는 모든 AI 상태 기계가 초기화**한다.

- **B-11-2-8.** 한 캐릭터 개체가 소유한 AI 상태 기계들끼리는, 필요한 경우 서로의 내부 멤버 변수들을 손쉽게 참조할 수 있는 인터페이스를 제공해야 한다.

※ 일견, 객체 지향 원리에 어긋나는 것처럼 보이는 위 요구사항은, 인공지능 상태 기계의 구현 특성 때문이다.
이는 마치, AI 상태 기계 클래스가 원래 하나의 클래스인데, 목적에 따라 여러 개로 클래스를 갈라서 쓰는 방식처럼 사용하게 해달라는 뜻이다. C#에서의 partial class와도 의미가 통하는 부분이 있다.
AI의 상태 기계를 나누는 기준은 행동의 어떤 부분을 기준으로 하느냐에 따라 매우 달라지기 때문에 정확한 정답이 없다. 이는, 그만큼 공통으로 사용해야 하는 상태들이 많다는 뜻이기도 하다.

예를 들자면, 전투 중에 어딘가로 이동한다고 했을 때, 이전에 공격하던 적이 누구였는지에 대한 정보, 또는 거리에 대한 정보는 전투 상태나 이동 상태나 둘 다 필요할 가능성이 있다. 심지어, 지금이 전투에 들어갈 때인지 판단하기 위해서 평화로운 상태의 상태 기계에서도 참조를 요구할 수 있다.
이런 상황은 생각보다 아주 빈번하게 발생하기 때문에, 각 AI 상태 기계들은 서로의 상태 기계가 존재한다는 보장과 함께, 각 상태 기계들 안에서 사용하는 정보들에 대한 접근 권한이 필요하다.

물론, '객체 지향적인' 구현의 입장에서 본다면, 이런 공통적인 멤버들은 상태 기계들을 소유하는 AI 컴포넌트의 직접 소유로 구현하는 것도 방법이다. 다만, 이 경우에는 아주 많은 변수들을 AI 컴포넌트가 소유할 수 밖에 없는 구조이기 때문에, AI 컴포넌트가 개수대(kitchen sink) 클래스처럼 보일 수도 있다.
구현하기에 적절하다고 생각하는 설계를 선택해서 적용할 것. 중요한 점은 **각 AI 상태 기계들은 공동 운명체**라는 점이다.

- **B-11-2-9.** AI 상태 기계를 전환하는 경우, 과거의 상태 기계 데이터와 논리 흐름은 유지하지 않는다. 항상 새로운 상태 기계의 처음부터 다시 논리를 수행한다.

※ 예를 들자면, 전투 상태에서 이동 상태로 전환한 뒤, 다시 전투 상태로 전환한다고 해도, 이전에 쓰던 전투 상태에서의 정보들이, 현재 전환된 전투 상태에서도 유효하다는 보장은 전혀 없다. 이전에 쓰던 전투 상태에서 원래 공격하고던 있었던 적이 지금 막 다시 전투 상태로 전환된 뒤에는 이미 죽어버렸을 수도 있고, 공격자가 중간에 지정한 목표가 바뀌었을 수도 있다.

그러므로, 상태 기계가 전환되는 경우, 이전 상태 기계에서 사용했던 조건 값이 어떻든 간에 모두 처음부터 다시 평가해서 진행한다.

◆ B-11-3. 길 찾기

- **B-11-3-1.** 길 찾기 기능은 클라이언트에서 수행한다.

※ 게임 설계 상으로, 서버가 개별적인 게임 방 인스턴스에 대한 정보를 보유할 의무가 없다. 따라서, 길 찾기 정보의 계산과 검증은 클라이언트에서 한다.

- **B-11-3-2.** 각 지형 모델마다 길 찾기 메쉬(Navigation Mesh)에 대한 정보를 두고, 이 정보를 기반으로 길을 찾게 한다.

※ Unity 엔진에는 Navigation Mesh 생성 기능이 내장되어 있다. 그러므로, 특별한 이유가 없다면 Navigation Mesh 는 Unity 엔진에서 정의하고 있는 기능들의 요구사항을 따른다.

- **B-11-3-3.** 길 찾기 메쉬 모델의 정밀도는 굳이 조절 가능할 필요가 없다.

※ Unity 엔진 플러그인 중에 ' A* Pathfinding Pro '는 이 기능을 보유하고 있다.
Unity 에 내장된 Navigation Mesh 생성기에는 이러한 기능이 없다.

- **B-11-3-4.** 반드시 거쳐야 하는 목표물이나 목표 지점을 정의할 수 있는 기능을 제공한다.

- **B-11-3-5.** 반드시 거쳐야 하는 목표물 혹은 목표 지점에 대한 정보는 스테이지 정보에 포함하는 방식으로 관리한다.
: 대개, 지형 맵이 달라지면 스테이지도 달라지긴 하지만, 경우에 따라 같은 지형 맵을 여러 스테이지가 사용할 경우도 생각해야 한다. 이 때는 미묘하게 필수적으로 거쳐가야 하는 구간도 달라질 가능성이 있다.

※ A* 알고리즘이나 길 찾기 메쉬를 이용하는 방식 모두 '최단 거리의' 올바른 길을 찾는 결과를 보장한다. 하지만, 가장 빠른 길을 찾는 것만이 게임에서 가장 좋은 연출은 아니다. 게임 플레이의 목적에 따라 반드시 거쳐 가야만 하는 중요한 지점이 있을 수 있는데, 수학적인 길 찾기 알고리즘은 이러한 부분을 전혀 고려하지 않기 때문이다.
따라서, 게임 플레이에서의 중요한 중간 목표 지점을 별도로 지정할 수 있는 기능이 꼭 필요하다. 각 중간 목표 지점 간의 '세부적인' 길 찾기는 수학적인 길 찾기 알고리즘이 담당하더라도, 게임 플레이 상 중요한 지점들을 거쳐 가는 경로를 만들어줄 수 있어야 한다.

- **B-11-3-6.** 중간 목표 지점들은 여러 개를 둘 수 있으며, 거쳐야 하는 순서를 지정할 수 있어

야 한다.

- **B-11-3-7.** 각 캐릭터 개체들마다, 행동 목적과 현재 상황에 따라 중간 목표 지점을 거쳐가는 순서를 다르게 할 수 있어야 한다.

※ 캐릭터 개체가 게임 플레이에서 어떤 임무를 가지고 있는지에 따라, 목표 지점을 달리 해야 할 수 있다.
물론, 이건 원칙적으로 그렇게 설계하는 편이 확장하기 손쉽기 때문이고, 실제 이 프로젝트의 목적은 대개 일방통행 식의 던전 돌파이기 때문에, 복잡하게 중간 목표의 경로들을 전환하는 과정은 특별히 없을 것으로 예상한다.

◆ B-11-4. 캐릭터 개체들의 점유 공간 찾기

- **B-11-4-1.** 캐릭터 개체들이 점유 공간을 가진다는 의미는, 캐릭터 개체들의 모델이 밀도를 가진 물체들의 상호작용처럼, 서로 겹치지 않고 간격을 두면서 존재한다는 뜻이다.

- **B-11-4-2.** 다만, 게임에서 캐릭터 개체들의 점유 공간을 찾는 목적은, 정확하게 현실과 동일한 묘사를 하기 위한 게 아니라, **연출적으로 그럴 듯 하게 보이기 위함**이다.

※ 캐릭터들끼리 같은 위치에 똘똘 뭉치는 현상이 없는 게임이라고 할지라도, 캐릭터의 모델 메쉬들 중 일부는 서로 겹치는 현상을 허용한다. 특히 팔의 하박 부위나 손 등의 말단 부위가 대개 그러하다.
게임에서 캐릭터 개체들끼리 겹치지 않는다고 하면, '대체적으로' 겹치지 않는다는 의미이지, 실제 세계처럼 아주 정확하게 질량과 밀도 관계를 적용한다는 뜻이 아니다.

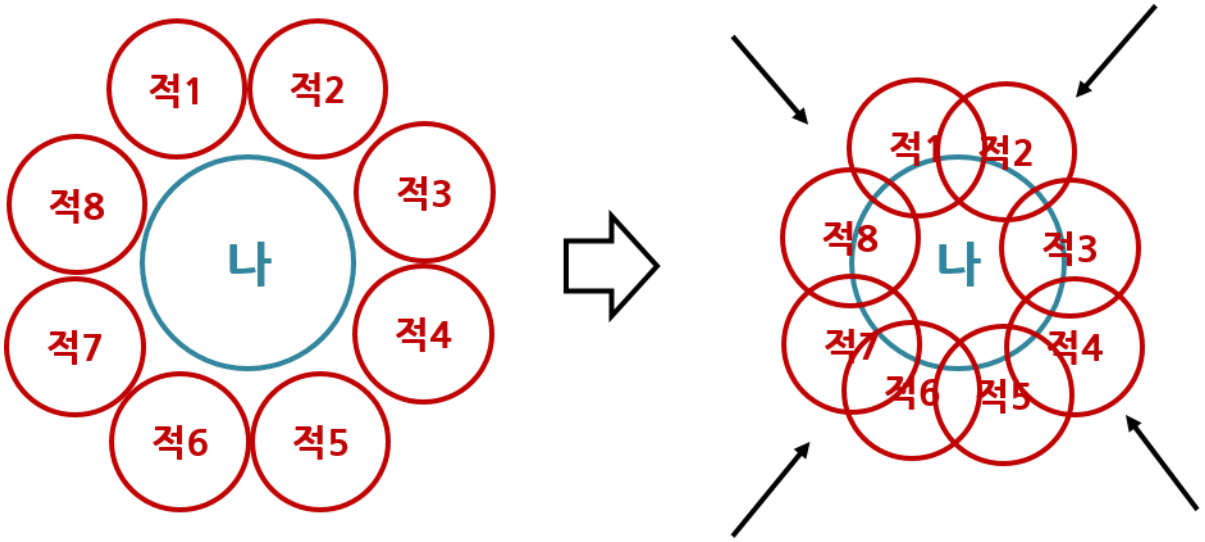
※ 그런 이유 때문에, 점유 공간을 찾는 기능은 구현 우선 순위가 낮다. 이 기능이 없다고 해서 게임의 핵심적인 기능이 작동하지 않는 경우는 없기 때문이다.

- **B-11-4-3.** 점유 공간을 찾는 계산은 **게임 플레이 상 필요한 경우에만 한 번씩 수행**한다.
: 매 업데이트 때마다 계속해서 점유 공간을 감시하면서 다른 개체들이 자신의 점유 공간 안에 들어오지 못하게 하는 계산을 하지 않는다.

※ 게임의 성격상, 다수의 몬스터들이 우르르 죽어나가는 일이 많을 것이고, 짧은 전투를 지속적으로 반복할 것으로 예상한다. 따라서, 캐릭터 개체들 간에 겹치지 않는 간격을 정밀하게 유지해야 할 의무가 없다.

- **B-11-4-4.** 점유 공간을 찾는 기능을 항상 유지해야 할 의무가 없다.
: 어떤 이유로 인해 캐릭터들끼리 서로 가까워져서, 서로 간에 모델이 겹치는 상황이 오게 되더

라도, 강제로 자신들의 모델 영역 바깥으로 서로를 밀어내지 않는다.

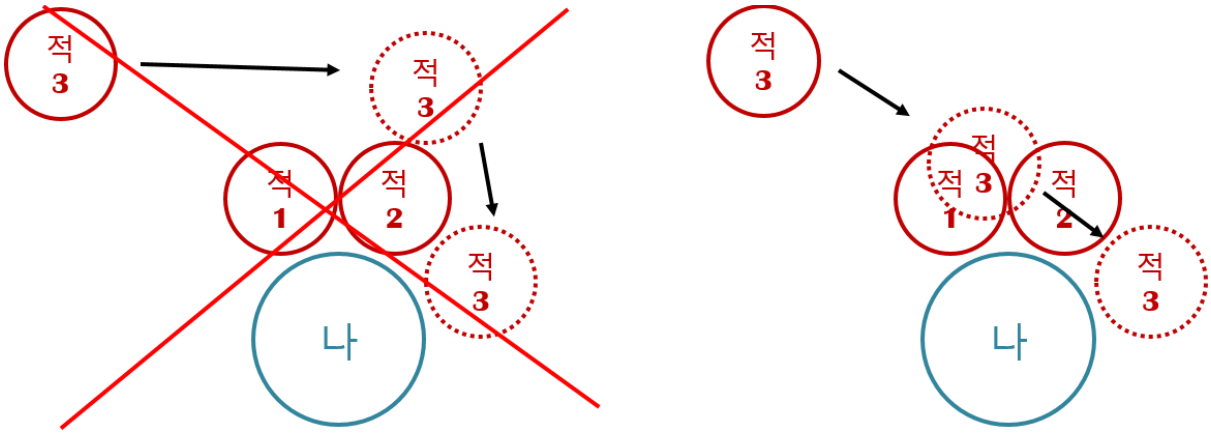


<서로 겹치게 되더라도, 그냥 그 상태를 유지한다.>

※ 대상을 강제로 밀어내거나, 끌어오는 스킬에 다수의 대상이 적중하는 경우, 위 그림처럼 대상 개체들끼리 서로의 점유 영역이 겹칠 수 있다. 만약 이러한 상황이 되더라도, 위 그림의 왼쪽 상황으로 되돌리기 위해 강제로 서로 빈 공간으로 밀어내지는 않는다.

- **B-11-4-5.** 길 찾기를 하며 이동하는 도중에는 점유 공간에 대한 감시를 하지 않는다. 오직, **도달해야 하는 목적지에 대해서만 점유 공간을 계산**한다.

※ 따라서, 누군가를 포위하러 여러 몬스터가 달려들어가는 경우에도, 달려들어가는 그 과정에는 몬스터들끼리 겹침이 있더라도 허용하겠다는 뜻이다.
마지막에 몬스터들이 대상 개체를 둘러싸려고 하는 순간에만 점유 공간에 대한 계산을 한다.



<점유 공간은 필요한 경우, 즉 마지막 목적지 위치를 파악할 때만 계산한다.>

◆ B-11-5. 지능적인 자동화된 플레이

- **B-11-5-1.** 모바일 기기에서 조작해야 하는 게임이기 때문에, 최소한의 조작으로 사용자의 의도를 반영하기 위해, 지능적으로 동작하는 자동화된 플레이 기능을 기본 혹은 선택할 수 있는 방식으로 구현한다.

- **B-11-5-2.** 자동으로 다음 공격 목표 대상을 설정하는 기능
  : 처음에 공격 목록에 등록된 순서대로 다음 목표 대상으로 설정한다. 매번 공격 목록의 등록 순서를 거리 순으로 정렬한다거나 하는 연산은 하지 않는다. (쓸데없는 성능 낭비다.)

- **B-11-5-3.** 임무 완료 조건을 달성하기 위한 탐색
  : 임무 조건을 위해 필요한 목표를 인식하고 그 장소를 향해 다가가는 기능이다. 자동으로 다음 공격 목표를 설정하는 것보다 더 높은 상위 단계에서의 자동화 플레이이다.

※ 대부분의 게임 플레이에서 기본적으로 '던전 돌파' 임무가 주어진 상태라고 보면 된다.
플레이어의 캐릭터가 시작 이후부터 자동 플레이 상태에서는 죽 스테이지의 마지막 지점까지 길을 찾아가는 행위가 바로 임무 완료 조건을 달성하려는 탐색 과정의 한 구현이라고 보면 된다.

◆ B-11-6. 전투 인공지능

- **B-11-6-1.** 일반적인 몬스터들과 준 보스급 몬스터들은 복잡한 전투 패턴을 사용하지 않는다.
  : **플레이어와 AI 간의 전술 싸움을 하는 게 목적이 아니라, '화끈한' 전투를 하는 게 목적**이기 때문에 불필요하게 고수준의 전투 구현을 만들 필요가 없다.

- **B-11-6-2.** 가장 많은 수가 등장하는 하위 몬스터들은 체력 수치를 이유로 해서 후퇴하는 움직임을 보이지 않는다.
  : 화끈하게 쓸어버리는 손맛을 방해하기 때문이다... 그리고 금방금방 물리치는 컨셉인 일반 몬스터가, 조건에 따라 정교한 전술적인 움직임을 보이려고 하는 것도 어색하다.

- **B-11-6-3.** AI가 조종하는 캐릭터가 선택적으로 사용하는 스킬을 보유한 경우, 스킬의 재사용 대기시간이 지나면, 일정한 무작위한 시간 이내에 다시 스킬을 사용하는 방식을 쓴다.

※ 이 방식이 구현하기가 가장 간단하다.
선택적 사용 스킬의 특성마다 적절한 조건을 구현해주는 방식도 나쁜 건 아니지만, **AI로 조종하는 캐릭터가 사람이 직접 컨트롤하는 경우 이상으로 효율적으로 전투해야 하는 기능에 대한 요구사항은 없다.**

- **B-11-6-4.** 플레이어의 캐릭터는 AI에 의해 자동으로 플레이하는 상태라면, 전투 상황에서도 일반 공격만 할 뿐, 선택적 사용 스킬은 쓰지 않는다.
: 단, 길드 전장(R vs R 컨텐츠)에서는 예외이다.

※ 길드 전장의 명세 부분에 더 상세히 설명하겠지만, 길드 전장은 그 어떤 캐릭터도 조종할 수 없고, 오직 자동으로 진행되는 전투의 장면만을 관전할 수 있다. 이때는 플레이어의 캐릭터라도 그저 순수하게 플레이어의 개입 없이 AI에 의해 조종되는 NPC처럼 취급한다.

- **B-11-6-5.** 일반적인 전투 패턴으로 분류

1. 근접형 몬스터
: 공격 사정거리가 짧고 체력이 강한 몬스터들은 이런 타입으로 분류한다.
전투에서의 특징은, 최대한 적과 거리를 좁히려고 하며, 후퇴 없이 공격만 하려고 하는 점이다. 일반 몬스터이기 때문에, 특별히 후퇴하는 법이 없이 전형적인 자살 돌격하는 인공지능으로 구현하면 된다.

2. 원거리형 몬스터
: 공격 사정거리가 길고 체력이 약한 몬스터들은 이런 타입으로 분류한다.
전투에서의 특징은, 가끔씩 현재 적과의 거리를 점검하고, 최대 사정거리만큼 거리를 벌리는 움직임을 보여야 한다는 점이다.
다만, 생명력이 낮다고 해서 후퇴하지는 않는다.

3. 마법형 몬스터
: 마법형 몬스터들은 근접이거나, 원거리 형일 수도 있지만, 플레이어가 조종하는 캐릭터나 보스 몬스터들처럼, 선택적으로 사용하는 스킬을 가지고 있다.
그들은 스킬의 재사용 대기 시간이 지나면, 무작위 한 오차 범위 내에서 스킬을 사용한다.
(스킬 자체는 적을 공격하는 것일 수도 있고, 아군을 도와주는 것일 수도 있다.)
이들도 생명력이 낮다고 후퇴하지는 않는다.

4. 준 보스형 몬스터
: 준 보스형 몬스터의 제작 컨셉에 따라 위 3 가지 일반 몬스터들의 전투 패턴 중 하나를 가지며, 기본적으로 준 보스 몬스터들부터는 선택적으로 사용하는 스킬을 기본적으로 가진다.
3 가지 일반 몬스터들의 전투 패턴에, 스킬 사용 기느

5. AI 가 조종하는 플레이어
: 큰 범위에서는 마법형 몬스터의 인공지능 패턴과 기본적으로 유사하지만, 각 플레이어 캐릭터의 직업마다 고유한 인공지능 패턴으로 구현할 수 있다. 왜냐하면 플레이어들의 스킬 조합이 다양할 수 있기 때문이다.
이 부분에 대한 상세한 구현은 관련된 기획서의 요구사항을 따른다.

- **B-11-6-6.** 각 보스 몬스터들의 특별한 전투 상태 구현
  : 각 보스 몬스터들마다 적용해야 하는 특별한 전투 인공지능은, 각 보스 몬스터 별로 별도의 소스 코드를 적용하는 방식으로 제작한다.

※ 보스 몬스터의 전투 페이즈에 대해서는 일반화할 부분이 별로 없다. 전투 페이즈의 방식에 대해 어떤 패턴을 설정할 수는 있겠지만, 결국 고유한 전투 이벤트가 존재하는 한, 각 보스 몬스터 별로 고유한 '하드 코딩'은 피하기 어렵다.

B-12. 물리 시스템

◆ B-12-1. 제한사항

- **B-12-1-1.** 충돌체끼리의 충돌 여부 검출과 Raycast를 제외하고, 엔진에 내장된 물리 관련 기능들을 사용하지 않는다.

※ Character Controller, Rigidbody, NavMeshAgent 등 Unity 에 내장된 물리 관련 기능과 지원 컴포넌트들을 사용하면 훨씬 손쉽게 기능들을 구현하는 데 도움을 주는 건 사실이다. 하지만, 실제로 모바일 기기에 본격적으로 게임을 구동하는 시점에서는 여전히 성능에 부담을 주는 것도 현실이다. 비록, 내장된 물리 엔진에 모바일 환경에 최적화했다고 해도 그 사실이 변하지는 않는다.

더구나, 모바일 환경에서의 부족한 성능 때문에, 물리 기능을 판정하기 위한 감지 업데이트가 원활하지 않아서 발생하는 예상치 못한 버그가 발목을 잡을 때가 많다. 특히 강체(Rigidbody)를 사용할 때 그런 문제가 두드러진다.
대표적인 예가 경사진 지형에서 의도와 다르게 캐릭터가 미끄러져 버리거나, 본래 장애물로 설정되어 있어야 하는 지점을, 계속해서 이동을 시도하면 어느 순간 타고 올라가 버리거나(...), 큰 충돌체로 막아 놓은 지점을 계속해서 이동을 시도하면 어느 순간 뚫고 들어간다든가... (심지어 이 때 중력까지 적용해놓으면 한 없이 나락으로 떨어지는 캐릭터를 볼 수 있다.)

- **B-12-1-2.** 연출을 위한 용도 외에는, 창발적으로 발생하는 물리 기능의 연산 결과를 게임 내 이벤트 판정에 이용하지 않는다.
: 전투 판정이라든가, 미션에 대한 판정 같은 것을 말한다.

※ 물리 엔진의 내부에서 일어나는 계산과 반응의 결과를 게임 이벤트에 판정하기 위해서는, 그러한 판정에 영향을 받는 모든 객체들이 물리 관련 연산을 감시하고 있어야 할 것이고, 이는 기기의 성능을 크게 잡아먹을 것이다. 또한, 최적화도 쉽지 않을 것으로 판단한다.

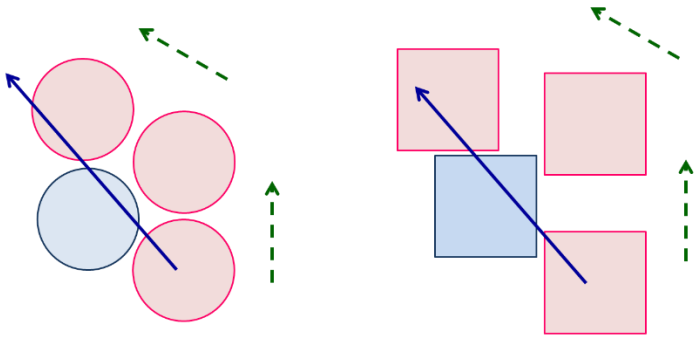
- **B-12-1-3.** 움직이는 모델의 충돌 검사는 반드시 **구형(Sphere), 상자(Box), 캡슐(Capsule)** 모양으로만 검사한다.

※ 이렇게 해야 가장 저렴한 비용으로 충돌 검사를 할 수 있다.

◆ B-12-2. 캐릭터 객체와의 충돌

- **B-12-2-1.** 캐릭터들은 **캡슐(Capsule) 형태**의 충돌체를 가진다.
: 인간형 캐릭터와 가장 비슷하게 생겼다.

※ 사실 캡슐 모양의 충돌체가 가장 성능 비용이 저렴하지는 않다.
더 저렴하기는 구(Sphere)나 상자(Cube)가 더 낫지만, 구의 경우에는 사람 모양과 비슷하지가 않고, 상자의 경우에는 단면이 사각형이기 때문에 충돌했을 때 우회하도록 만들기가 쉽지 않다.



<단면이 사각형인 개체는 아무래도 어색해 보인다.>

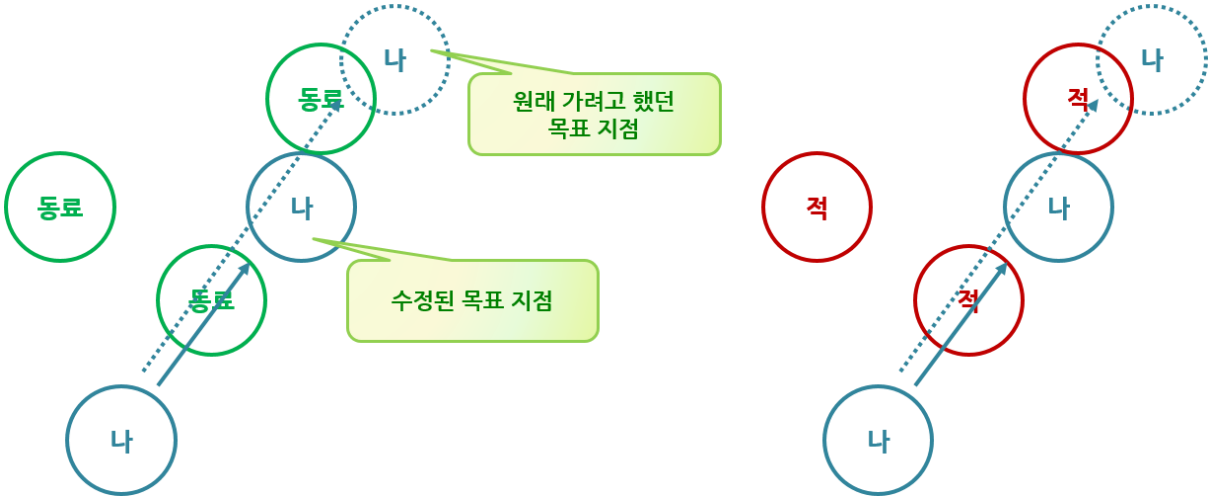
- **B-12-2-2.** 캐릭터 개체들은 게임 월드에서의 **중력을 적용할 수 있어야 한다.**
: 그러니까... 게임에서 개체가 공중의 높은 위치에 있으면, 아래에 있는 바닥 지형의 충돌체가 있는 지점까지 낙하하는 기능이 가능해야 한다는 뜻이다.

※ 게임 플레이 도중에 돌아다니는 캐릭터 개체들이 이런 가상의 중력을 어느 시점이든 항상 적용해야 한다는 의미는 아니다.
성능상 부담이 있는 등의 이유가 있다면, 필요할 때마다 해당 컴포넌트를 붙여서 잠시 작동시키는 식으로 사용해도 무방하다.
중요한 것은, 게임 내 캐릭터들이 실제처럼 낙하를 하거나, 투사체가 포물선 운동을 할 수도 있어야 한다는 것이다.

- **B-12-2-3.** 움직이는 캐릭터끼리는 서로의 캐릭터 별로 점유하는 영역을 차지하기 위해, 서로 충돌 점검을 하는 계산을 하지 않는다.
: 즉, 이동 중에 캐릭터끼리 모델이 겹치는 모습이 보이더라도 이를 허용한다.

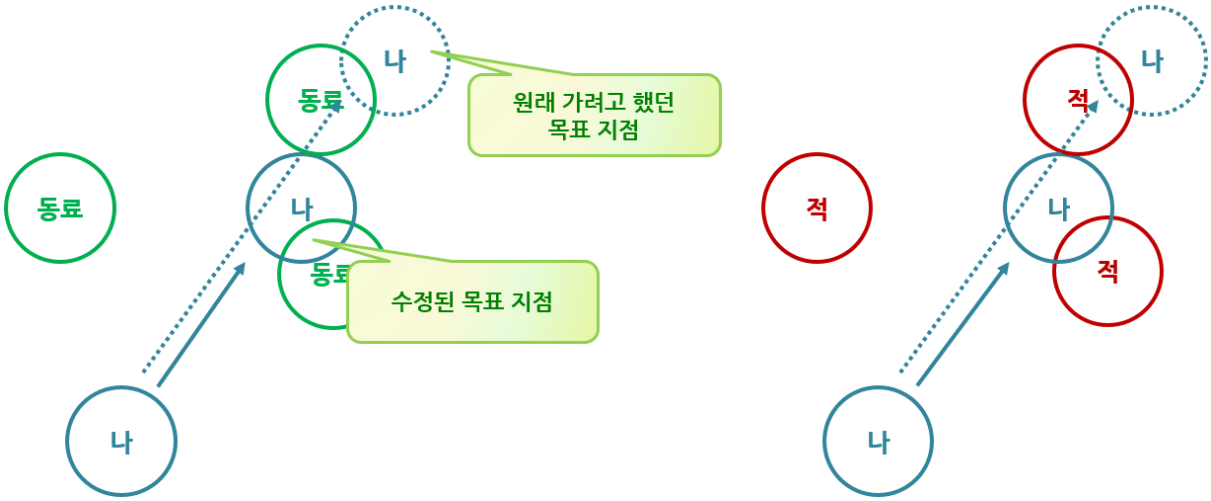
※ 여기서 말하는 기능은 소위 비비면서 빠져 나가기인데... 이 기능이 반드시 플레이어가 조종하는 캐릭터에 있어야 할 이유는 없다고 본다. 사용자의 조종을 통해 막힌 곳을 우회해서 빠져나오면 되기 때문이다.
눈에 그다지 띄지 않으면서 구현 비용이 클 것이라고 판단해서 이렇게 결정하였다.

- **B-12-2-4.** **캐릭터 개체들의 공간 점유 시도는, 캐릭터 개체가 가려고 하는 목표 지점에 대해서만 계산**한다.
  : 즉, 목표 지점까지 가려고 할 때 중간에 다른 캐릭터들과 겹치는 부분은 상관하지 않는다는 의미이다. 이는 적대적인 캐릭터일지라도 마찬가지이다.



<목표 지점에 대해서만 빈 공간을 찾아서 이동하는 원리는 같다.>

- **B-12-2-5.** 최초 목표 지점에서 다른 캐릭터와 겹치는 경우만을 검출하고, 나머지는 경우는 무시한다.
  : 수정한 위치에서 다시 또 다른 캐릭터와 겹치더라도, 그럴 때의 겹침 상태를 인정한다.



<수정된 목표 지점부터는 겹치는 영역이 있어도 그대로 간다.>

※ 내가 목표한 지점에 겹치는 캐릭터들에게 겹침 정보를 중계하고, 중계 받은 캐릭터가 밀려나고 -> 그 캐릭터가 겹침 검사를 한 뒤에, 겹침이 발생하면 다른 겹쳐진 캐릭터에게 중계하고... -> (무한 반복) 방식으로 하는 방법도 생각을 해봤다.
이 방식이 '올바르기는' 할 텐데, 이걸 모바일 장치에서 매번 연산하기에는 계산량이 과도하지 않을까 하는 걱정이 우선 들었다.

그것도 그렇고, 올바르게 잘 구현하려면 시간 비용도 크겠다는 생각이 들어서, 겹치는 영역에 대한 검사를 최소한으로 쓰는 방향으로 잡았다.

P.S : 그러고 보면, 겹침 검사 결과를 다른 캐릭터에게 재귀적으로 전달을 반복해서 수정하는 방식을 사용한다면, 어째 '스타크래프트2'의 유닛들의 이동 알고리즘과 유사하다는 생각이 든다.

- **B-12-2-6.** 2014. 09. 17 회의에 의해, 각 캐릭터 개체들이 항상 공간 점유를 검사하는 방향으로 결정했다.
  이에 따라, B-11-2의 내용을 전면 개정해야 한다.(흑흑...)

◆ B-12-3. 지형 객체와의 충돌

- **B-12-3-1.** 정적 지형의 경우, 캐릭터 개체들은 이동 가능한 구역으로 설정된 영역 안에서만 돌아다닐 수 있다.

※ Unity 엔진에서는, Navigation Mesh로 설정된 영역에 해당한다고 보면 된다.

- **B-12-3-2.** 게임 플레이에서의 이벤트 결과로 인해 캐릭터의 위치가 변하는 경우에도, 지형 중에서 이동 가능한 영역 내에서만 위치 변화가 가능하다.

※ 제 아무리 밀쳐내기, 띄우기, 순간 이동 등을 하더라도, 이동 가능한 영역으로 설정되어 있는 영역 내에서만 해야 한다는 뜻이며, 이 영역을 뚫고 지나가거나 할 수는 없다.

- **B-12-3-3.** 캐릭터가 아닌 객체들은, 경우에 따라 이동이 불가능한 영역으로 넘어가는 것도 가능하다.
  : **연출에 관한 이유 때문에 이동 불가능한 지형의 영역 너머로 객체를 보내는 경우를 허용**한다는 의미이다. 특히, 사용자가 조종하지 못하며, 일회성으로 생겨났다가 없어지는 객체들이 그러하다.

※ 대표적인 예가 총알, 원거리 스킬 등의 투사체와 그것들의 이펙트 연출 객체들이다.
  이런 건 일반 발사하면 목표 지점까지 끝까지 날아가는 게 보여야 한다. 중간에 이동할 수 없는 지형이라고 해서 투사체가 날아가다가 사라지거나 하면 연출이 어색해 보일 수 밖에 없다.
  그러므로 이런 객체들은 지형에서 이동이 가능한 위치에 있는지를 감시하지 않는다.

◆ B-12-4. 이펙트 객체와의 충돌

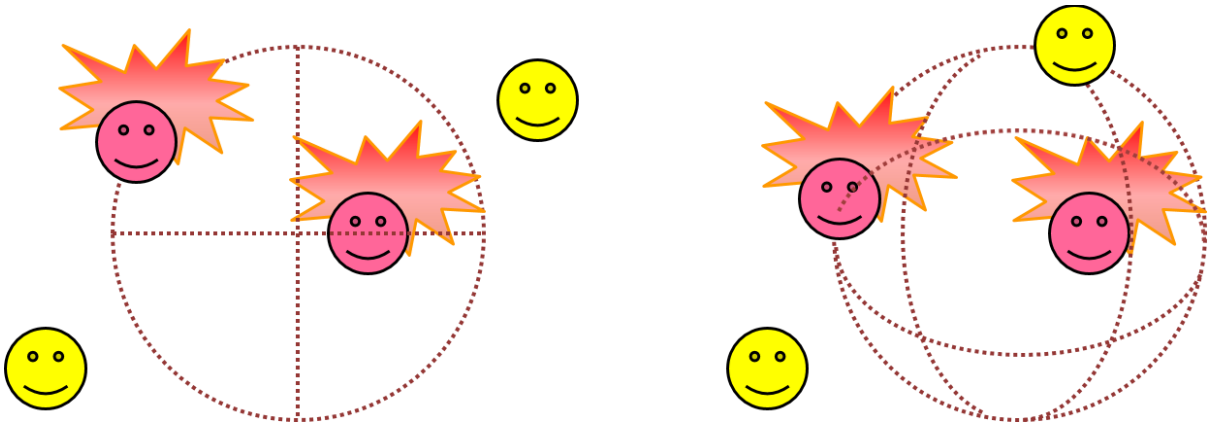
- **B-12-4-1.** 대부분의 용도는 스킬을 사용할 때, 스킬이 정의하는 효과 영역에 대상이 명중했는

지 판정하는 용도이다.

- **B-12-4-2.** 이펙트 객체의 충돌 방식에서, 메쉬의 모양으로 하는 충돌 검사를 수행하지 않는다.
: 최대한 연산 능력을 소모하지 않는 가장 간단한 방식으로 충돌 검사를 수행한다.

※ 특히, 동적으로 충돌 검출을 하는 충돌자(Collider)끼리 메쉬 모양으로 충돌 검사를 하면 극심하게 성능을 소모한다.
스킬 이펙트들은 워낙 자주 사용하는 만큼, 이들의 연산이 간단해질수록 성능 향상에 크게 기여할 수 있다.

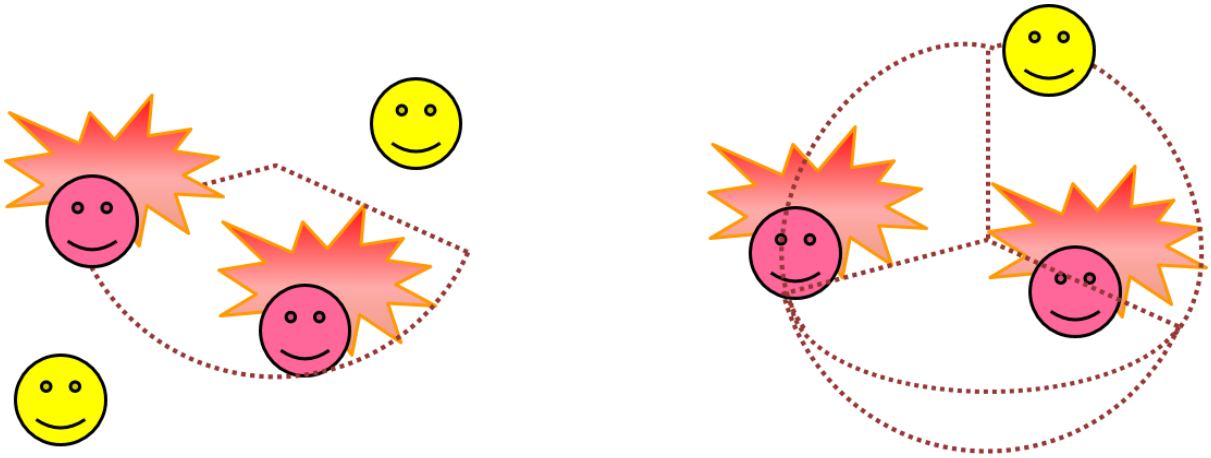
- **B-12-4-3.** **원형 충돌과 구체 충돌**
: 중심점으로부터 일정한 거리 이내에 (일부라도) 영역이 겹치는 객체들은 충돌한다.
원형과 구체 충돌의 차이는 **높이 값을 충돌 판정에서 고려하는지 여부**이다. 원형 충돌은 높이 값을 고려하지 않으나, 구체 충돌은 높이 값도 충돌 판정에 고려한다.



<원형과 구체 충돌>

- **B-12-4-4.** 원형 충돌과 구체 충돌은 충돌 범위의 각도를 지정할 수 있어야 한다.
: 단, 명중을 판정하는 각도는 시초선(각도가 0도가 되는 기준선) 기준으로 편향적일 수 없다.
항상 시초선으로부터 좌우 대칭인 각도 단위로만 판정한다.

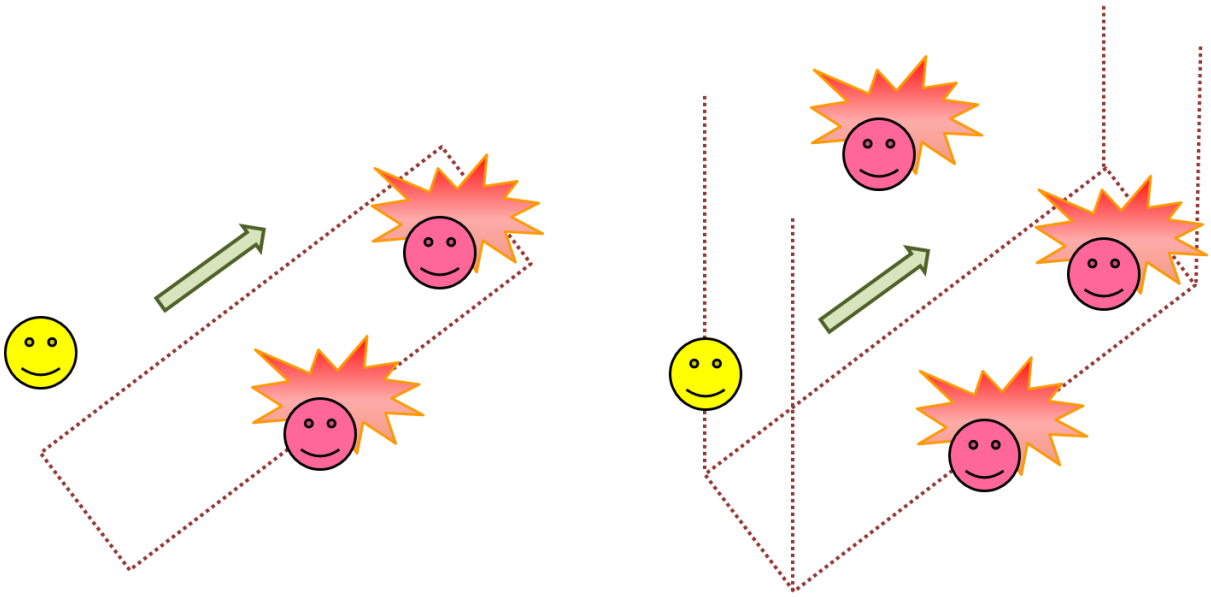
※ 예를 들어, 90도 각도로 판정한다고 하면, 판정할 기준이 되는 선으로부터 좌우 각 45도 안에 들어 있는 객체들을 충돌로 판정한다는 의미이다. 오른쪽으로만 90도, 혹은 왼쪽 20도에 오른쪽 70도 하는 식으로는 판정할 수 없다.
만약, 굳이 편향적인 각도로 판정을 해야만 한다면, **판정할 시초선의 방향을 편향적으로 설정하고 판정하면 동일한 효과**를 구현할 수 있다.



<원형과 구체의 충돌을 특정 각도 범위 이내에서 할 때>

- B-12-4-5. 사각형 충돌

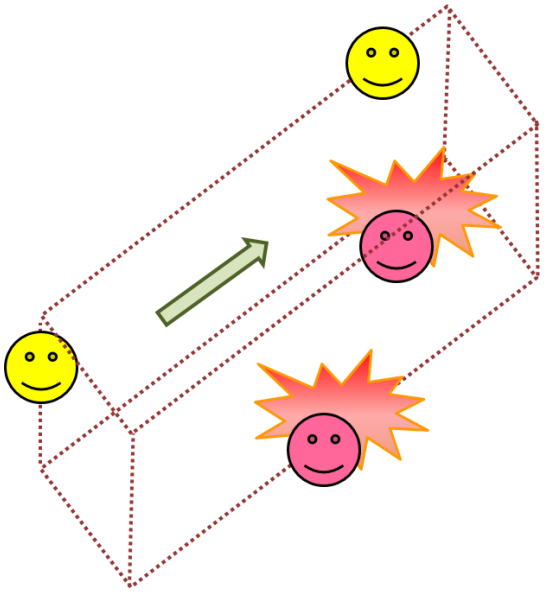
: 사각형 영역 안에 개체의 영역이 포함되면 충돌로 판정한다. 높이 값은 고려하지 않는다.



<사각형 충돌>

- B-12-4-6. 상자 충돌(육면체 충돌)

: 사각형 충돌과 거의 같지만, 높이 값을 고려해서 판정한다.



<상자 방식의 충돌>

◆ B-12-5. 궤적

- **B-12-5-1.** **직선**
  : 판정 객체는 투사 각도를 고려하여 현재 위치를 계산한다. 중력에 의한 하강은 전혀 고려하지 않는다.

- **B-12-5-2.** **포물선**
  : 판정 객체는 투사 각도와 중력에 의한 하강을 고려하여 현재 위치를 계산한다.

※ 직선 궤적은 포물선 궤적의 특수한 형태이지만, 실제 구현은 분리해야 할 수 있다.
일반적으로 포물선 궤적 계산이 훨씬 복잡한 과정을 거쳐야 하기 때문이다.
구현하기에 따라서는, 직선 궤적 계산은 포물선 궤적 계산의 일부 기능만을 호출하는 방식으로 만들 수도 있을 것이다.

- **B-12-5-3.** **목표 대상까지 유도**
  : 목표가 된 대상의 현재 위치를 지속적으로 추적해서 따라다닌다. 단, 추적하는 정확성이 모든 경우에 반드시 보장되어야만 할 필요는 없다. (어느 경우에는 정확하게 추적하고, 어느 경우에는 대충 추적하게 해도 된다는 뜻이다.)

- **B-12-5-4.** 중간에 궤적 방식을 다르게 변경하는 기능은 허용하지 않는다.
  : 이런 기능이 필요할 수도 있을지는 모르겠지만, 실효성도 의문이고 시스템도 지나치게 복잡할 것 같다. (솔직히 직선으로 나가던 총알을 특정 시점부터 유도 미사일 방식으로 바꾸는 정신 나간 스킬이 얼마나 될까..?)

- **B-12-5-5.** 궤적을 따르는 판정 객체들은 최소 / 최대 진행 거리를 가질 수 있다.
: 캐릭터의 최소 / 최대 사거리와 유사한 개념이라고 보면 된다. 다만, 직접적인 관련은 없다. 이는 단지 '연출'에 관련한 사항일 뿐이기 때문이다.

- **B-12-5-6.** 궤적을 따르는 판정 객체들은 최소 / 최대 진행 시간을 가질 수 있다.
: 역시 마찬가지로, 캐릭터의 능력이나, 판정 객체를 생성하게 만든 스킬의 능력 등과 직접 의존 관계를 가지지는 않는다. 이것 역시 연출과 관련해 지정할 수 있는 제한사항이다.

◆ B-12-6. 중력 관련

- **B-12-6-1.** 중력 / 반중력 구현할 때는 실제 지구상 중력의 효과와 동일하게 구현하지 않아도 상관 없다.
: 점프나 부양(Airborne) 상태에서 지면으로 떨어지는 기능을 구현할 때, 떨어지는 속도를 등속도 운동으로 구현해도 상관 없다.

※ 현실에서의 지구 중력은 약 9.8m/s의 등가속도 운동이다. 계산상의 편의를 위해 10m/s로 간주하기도 한다.

B-13. 이펙트 시스템

◆ B-13-1. 인스턴스 이펙트의 템플릿 요구 사항

- **B-13-1-1.** 인스턴스 이펙트는 3D 월드 공간에서의 특정한 좌표 위치에 생성하는 이펙트이다.
: 일반적으로, 이벤트의 대상이 되는 개체들의 위치를 따라 다니거나, 대상 개체의 좌표 값이 이펙트의 위치에 영향을 주는 경우는 거의 다 인스턴스 이펙트로 보면 된다.

- **B-13-1-2. 2D 이펙트 개체**
: 2차원 평면 Mesh, 혹은 파티클 기능을 이용해서 제작한다.

※ 주의할 점은, 인스턴스 이펙트가 '2D'로 표현한다고 해서, 2차원적인 내용을 표현하기 위함이 아니라는 점이다.
2D 인스턴스 이펙트는, 굳이 3D 메쉬로 모델링한 리소스가 필요하지 않고, 2차원 텍스처 데이터만으로도 이펙트를 표현하더라도, 게임 상에서 다른 3차원 모델들과 위화감이 느껴지지 않기 때문에 사용한다.
즉, 2D 인스턴스 이펙트는 어디까지나 **표현할 때의 성능 효율에 대한 문제**이지, 2D 이펙트가 2차원적인 내용을 표현하기 위해서 쓰는 게 아니다.

- **B-13-1-3. 3D 이펙트 개체**
: 3D 모델이 필요한 이펙트도 존재한다. 이러한 경우, 일반적인 캐릭터 개체 모델의 재생과 거의 유사하지만, 스스로 움직이지는 못하고, 특정 개체나 위치에 붙어 다니거나, 지정된 움직임만을 수행한다는 점이 다르다.

※ 이펙트 객체도 3D 모델 메쉬가 존재하고, 애니메이션을 하기도, 하며, 위치 정보를 가진다는 점에서 캐릭터 개체와 본질적으로 다른 부분은 없다.
그러니 이펙트 객체는 인공지능 컴포넌트가 없어서 스스로 뭔가를 '판단'해서 행동하지 않는다.
또한, 소유자가 명백히 정해져 있고, 생성 / 제거 역시 소유자로부터 발생하는 이벤트에 의존한다.

- **B-13-1-4.** 이펙트 개체에 대한 조절 옵션
: 각 시각 이펙트 개체들은 다음의 조절 기능을 필요로 한다.

1. 크기 조절
2. 색상 조절

3. 회전 각도 및 속도에 조절
4. 재생 속도 조절
5. 무작위적인 개수의 생성 / 삭제 관리
6. 내부 텍스처의 UV 좌표 적용 값, 타일 만들기(Tiling) 여부 조정
7. 위치 변환에 관한 애니메이션 적용 가능
8. 텍스처 애니메이션 적용 가능
9. 파티클 적용 가능
10. 잔상 기능(Mirroring, Trailing) 적용 가능

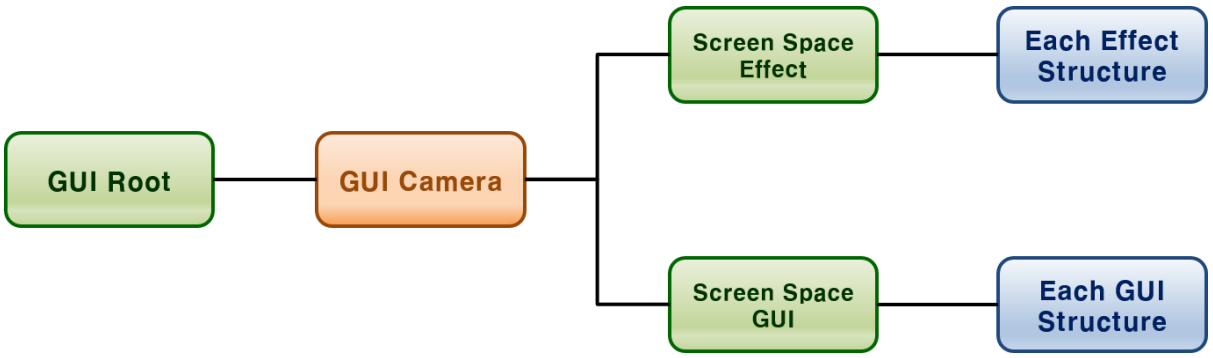
※ 위 기능들은 Unity 엔진에서 내부적으로 제공하는 컴포넌트들이 있으므로, 이를 조합해서 손쉽게 활용이 가능하다.
Unity 엔진의 내장 기능, 혹은 관련된 3th Party 제품 이용, 직접 제작에 대한 선택은 구현 비용과 성능 비용을 따져서 결정한다.

◆ B-13-2. 스크린 영역 이펙트

- **B-13-2-1.** 스크린 영역 이펙트는 스크린 영역에 그리는 GUI와 같은 원리로 구현한다.
: 이런 이펙트가 GUI의 Label, Sprite의 좀 특수한 부분이라고 볼 수 있다. 둘 다 스크린 영역에 그리는 방식이니까 말이다.

※ 생명력이 아주 낮을 때, 위험을 알리기 위해서 화면 가장자리가 빨갛게 깜빡이는 이펙트, 비나 눈이 내리는 날씨 이펙트, 공격 콤보를 보여주는 이펙트 등이 모두 스크린 영역 이펙트의 예가 된다.

- **B-13-2-2.** 단, GUI가 아니기 때문에, GUI를 끄는 옵션이 있다면, 스크린 영역 이펙트들은 제외하고 꺼야 한다.
: 구현 내부에서는 사실상 GUI와 같은 방식으로 구현한다 해도, 사용자 입력을 위한 GUI 계통과 스크린 영역 이펙트를 위한 계통을 분리하는 방식으로 설계를 해야 한다.



<화면 영역에 그리는 Effect와 GUI들의 내부 구현에서의 대략적인 계통 구조>

◆ B-13-3. 이펙트 개체의 관리

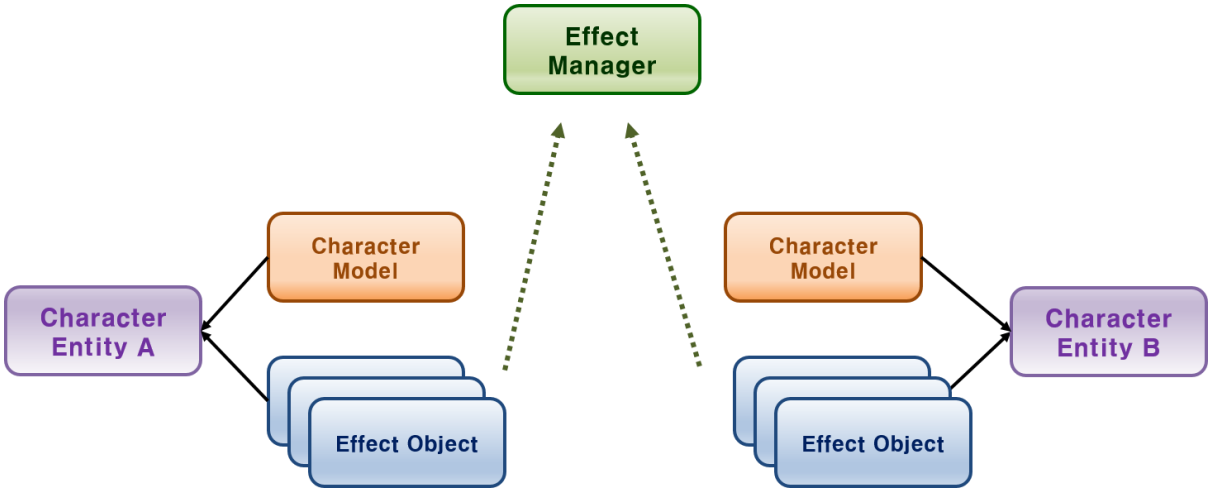
- **B-13-3-1.** **모든 이펙트 개체들은 이펙트 관리자에게 참조를 등록**한다.
  : 이펙트 관리자가 이펙트를 일괄적으로 관리할 수 있게 하기 위해서다.

- **B-13-3-2.** 이펙트 관리자는 이펙트를 전체적으로 관리하지만, 모든 이펙트 객체 인스턴스들의 수명주기를 도맡아서 추적 관리하지 않는다.

- **B-13-3-3.** 이펙트 관리자에게 직접 소유권을 위임한 경우를 제외하고, 이펙트의 수명을 결정하고 적절한 시점에 생성 / 제거하는 임무는 **이펙트를 불러와서 사용하는 객체에게 그 책임**이 있다.
  : 앞으로 이펙트를 호출해서 이용하는 객체를 **이펙트의 소유자 객체**로 부른다.

- **B-13-3-4.** 이펙트 관리자의 임무는 다음과 같다.

1. 이펙트 객체의 캐싱
2. 사용하지 않는 이펙트 객체의 수집 및 재활용 프로세스 담당
3. 이펙트 관리자에게 수명주기가 맡겨진 이펙트들의 수명주기 관리



<이펙트 객체와 이펙트 소유자 객체, 이펙트 관리자의 관계>

- **B-13-3-5.** 이펙트 소유자는 대개, 이펙트를 호출해야 하는 캐릭터 개체이지만, 전역적인 이벤트에 의해 호출하는 이펙트는 전역 이펙트 관리자에게 소유권을 넘길 수 있다

- **B-13-3-6.** 전역 이펙트 관리자는, 관리하는 이펙트들의 원래 소유자를 거치지 않고, 필요에 따라 일괄적으로 이펙트들을 관리할 수 있다.

: 한꺼번에 이펙트를 켜고 끈다거나, 일정 개수 이상의 이펙트를 화면에 출력하지 않도록 개수를 조절한다든가 하는 일이 가능하다.

- **B-13-3-7.** 이펙트 관리자는, 이펙트 개체 인스턴스들 자체를 생성하는 개수와, 화면에서 동시에 보이는 이펙트들의 개수를 둘 다 제한할 수 있다.

※ 이펙트 개체를 생성하더라도, 화면 바깥에 있으면 전혀 재생해야 할 필요가 없다. 물론, 단순히 보이고 말고의 경우는, 대개 엔진 내부의 3D API 레벨에서 카메라 절두체 객체 제외(Camera Frustum Culling) 등으로 해결하므로 별도의 구현 과정이 필요하지는 않다.
이펙트 개체들의 생성 여부를 통제하는 부분은 그래픽스 API가 해결하는 부분이 아니므로 이펙트 관리자 내부에서 관리 기능을 구현해야 한다.

- **B-13-3-8.** 동시에 보이는 이펙트 개체들의 개수가 제한된 상태에서는, 가장 오래된 이펙트 개체부터 차례로 보이지 않게 처리한다.

※ 이펙트 개체의 생성 개수 제한에 먼저 걸렸다면, 아예 이펙트 개체가 생성되지 않았을 것이므로, 이펙트 개체를 보일지 여부를 결정할 필요가 없다.
항상 [이펙트 개체의 생성 개수] -> [이펙트 개체가 동시에 보일 개수]의 순서로 판정한다.

- **B-13-3-9.** 화면 밖에 있는 좌표에서 벌어진 이벤트에 대한 이펙트들은 아예 처음부터 만들지 않는다.
: 즉, 이펙트를 연출해야 하는 이벤트가 발생한 좌표가 화면 안에서 발생한 경우에만, 그 이벤트의 이펙트를 재생한다.

※ 사실, 굳이 고려하자면, 이펙트 재생 시간이 아주 길어서, 최초 생성 시도 시점에서는 화면 밖이었지만, 재생이 진행되는 중에는 캐릭터가 이펙트가 발생했어야 할 좌표 지점으로 이동하는 경우도 있을 수는 있다.
그러나 대부분의 이펙트들은 단발성이고, 유지 시간도 매우 짧다. 투명도 값을 많이 쓸 수 밖에 없는 특성 때문에, 가급적 재생 시간을 짧게 유지해야 하기 때문이다. 정말로 대부분의 경우에는 화면 밖에서 발생한 이펙트들은 캐릭터가 다가가기 전에 이펙트 자체의 재생 시간이 종료된다.
몇몇 예외적인 사항들 때문에 이펙트 객체 인스턴스를 유지해야 하는 건 낭비라고 생각한다.

- **B-13-3-10.** 단, 어느 위치 좌표에 있는 캐릭터든지 봐야 하는 전역 이펙트들은, 화면 밖에서 벌어진 이벤트일지라도 내 화면에도 보여야 한다.
: 이런 이펙트들은 전역 이펙트라고 할 수 있다.

※ League Of Legend의 녹턴이나 카서스 궁극기 스킬에 따른 이펙트들이 대표적인 전역 이펙트라고 볼 수 있다.

- **B-13-3-11.** 이펙트 인스턴스의 개수 제한을 무시하는 '필수적인 이펙트'들을 지정할 수 있어야 한다.
: 이펙트 관리자는 이러한 필수 이펙트와, 개수 관리가 가능한 이펙트들을 구분해야 한다.

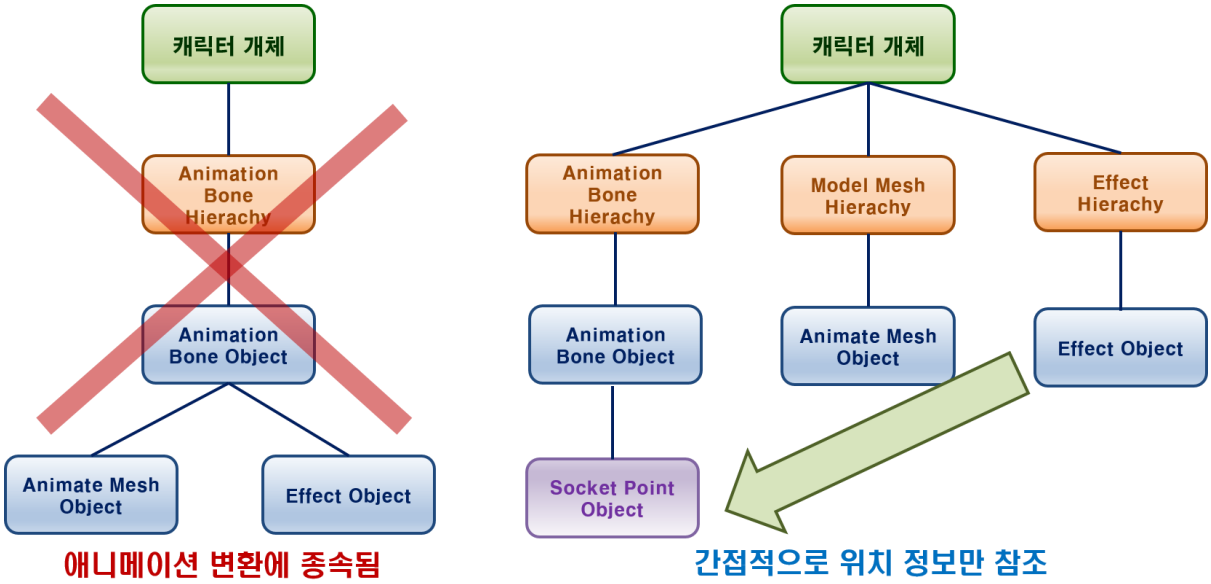
◆ B-13-4. 캐릭터 소유의 이펙트

- **B-13-4-1.** 이펙트 개체들이 어떤 캐릭터에게 붙는다고 할 때, 그 캐릭터의 애니메이션 노드의 하위 노드로 들어가지 않는다.
그 캐릭터가 소유하는 이펙트라고 해도, 캐릭터의 애니메이션 노드들과는 다른 계통으로 관리한다.

- **B-13-4-2.** 이펙트는 캐릭터 모델의 애니메이션에 의한 위치 변환을 따르기 위해, 특별하게 이를 위해 사전에 설정된 '위치 정보 소켓 객체'의 변환 정보를 참고하여 현재의 위치 좌표, 회전각, 크기 배율 조절을 한다.

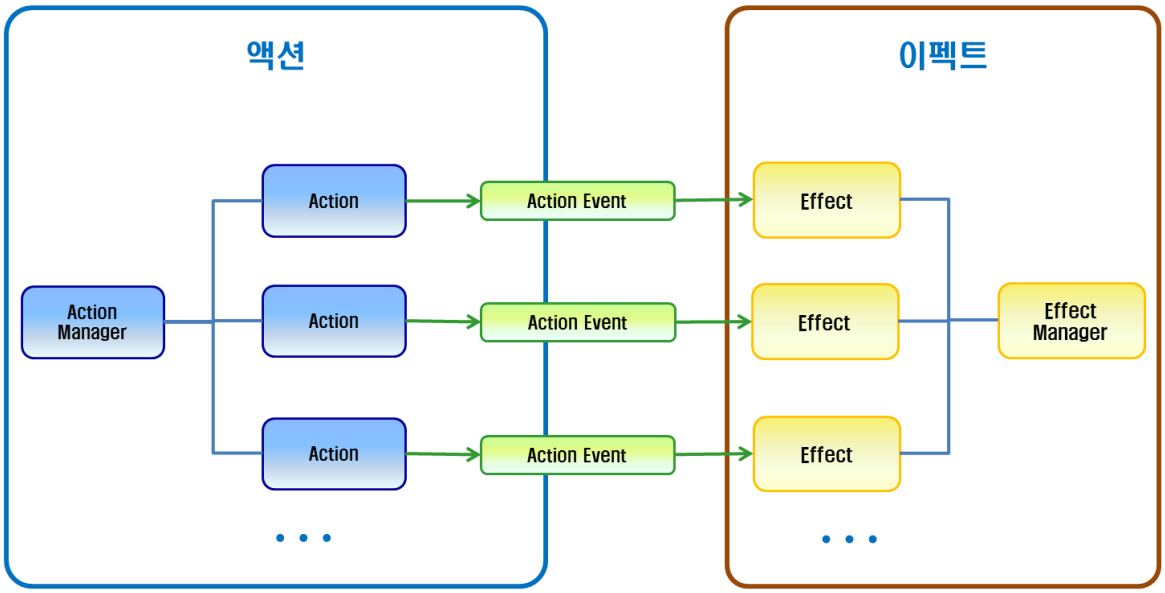
※ 이런 식으로 이펙트를 관리하는 이유는, 이펙트 중에서는 소유자 개체의 현재 회전 등에 관계없이 항상 소유자로부터 특정 지점을 유지해야 하는 경우가 있기 때문이다.
그런 이펙트 개체를 소유자 개체의 애니메이션 모델의 하위 계통 노드로 두면, 소유자의 회전에 따라 이펙트 개체도 따라서 회전이 되기 때문에 원하던 연출이 안 나올 수도 있다. (태양을 공전하는 지구를 떠올리면 된다.)

그래서, 소유자 개체의 위치를 따라다니기만 하는 이펙트 객체들은 소유자 개체와는 별개의 변환Transform 계통에서 변환이 관리되며, 소유자 객체에게는 자신의 참조자를 줌으로써 소유 관계를 연결시키도록 구현하는 게 좋다.
이렇게 되면, 이펙트 객체는 소유자의 현재 위치와 크기 변화를 '참고'하여 자신의 위치와 크기 변화를 별도로 업데이트하는 것이며, 상위인 소유자의 변환을 그대로 따라가는 것이 아니게 된다.



<캐릭터 소유의 이펙트라도, 직접 캐릭터 모델에 붙지 않고 간접적으로 위치만 참조한다.>

- **B-13-4-3.** 캐릭터 개체가 소유하는 이펙트들은 액션 이벤트에 의해 호출한다.
: 호출하는 경로를 일원화하기 위해서이다.



<액션이 액션 이벤트를 통해 이펙트에게 시켜먹는 관계이다.>

- **B-13-4-4.** 이펙트 객체들은 한 번 생성하면, 사용하지 않을 때, 실제로 제거하지 않고, 다음에 다시 사용할 일이 생길 때 재활용한다.
: 이러한 재활용 프로세스는 이펙트 관리자가 담당한다.

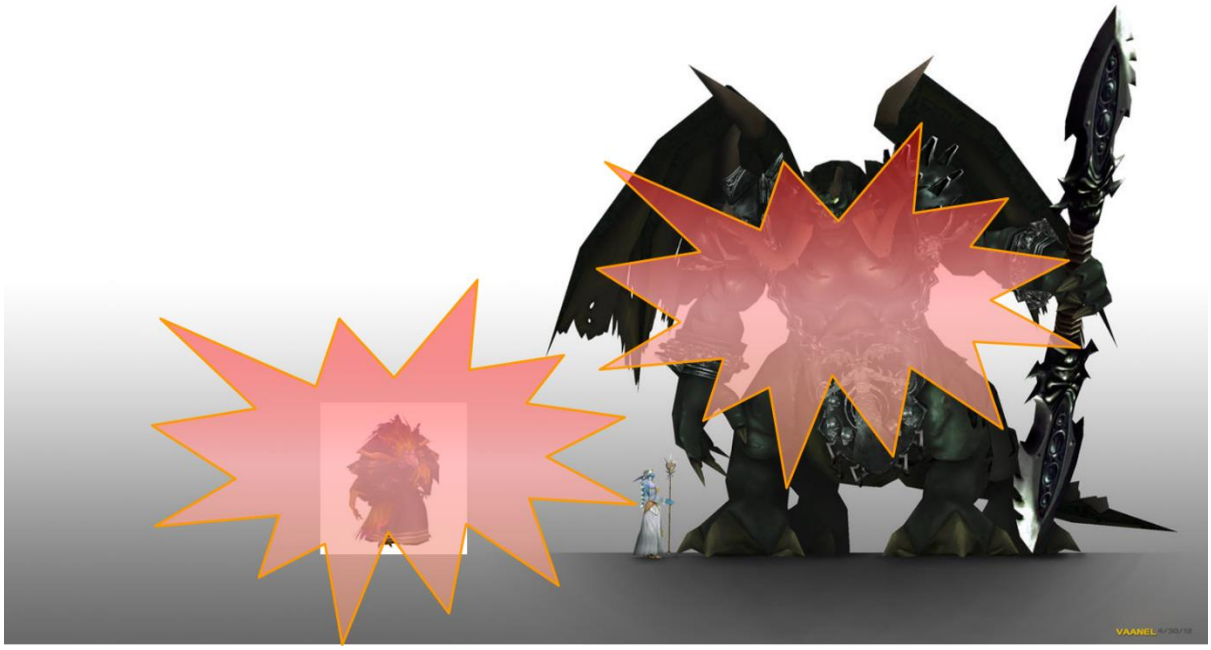
※ 게임 프로젝트에서의 객체들이 대체적으로 재활용 하는 방식을 기준으로 삼고 있다. 특히 이펙트의 경우에는 매우 자주 '생성 / 삭제'를 반복해야 하기 때문에, 실제로 메모리 할당 / 삭 제

를 하는 것보다, 객체의 활성 / 비활성 상태를 스위칭해서 재활용하는 편이 성능을 쾌적하게 유지하는 데 도움이 된다.

◆ B-13-5. 이펙트의 변환 규칙

- B-13-5-1. 이펙트의 크기는, 이펙트를 소유한 개체의 크기에 따라 달라질 수 있어야 한다.
: 같은 효과와 이벤트에 의해 생성되는 같은 성격의 이펙트라도 마찬가지다.

※ 공격에 적중 당했을 때, 피격되었음을 알리기 위해 뭔가 팍 터지듯 연출해주는 이펙트가 있다고 치자.
이런 이펙트가 고정된 크기만을 가진다면, 고양이 크기만한 몬스터와 용 크기의 몬스터가 같은 크기의 이펙트를 연출해야 한다는 의미이다. 이렇게 극단적으로 캐릭터 개체의 크기가 차이가 나는 경우는 RPG 에서는 얼마든지 볼 수 있다.
만약 캐릭터의 크기에 따라 이펙트의 크기를 조절하지 못한다면, 도대체 어느 기준으로 크기를 맞춰야 할까?



<이펙트가 항상 고정된 크기라면, 어느 쪽이라도 결국 어색한 상황이 나올 수 밖에 없다.>

- **B-13-5-2. 이펙트의 크기는 이펙트를 재생하는 액션 이벤트에, 재생할 이펙트의 크기를 매개변수로 넘겨서 정한다.**
: 즉, 이펙트의 크기를 결정하는 시점은 이펙트를 호출하는 액션 이벤트에 의해 이펙트가 재생되는 시점이다.

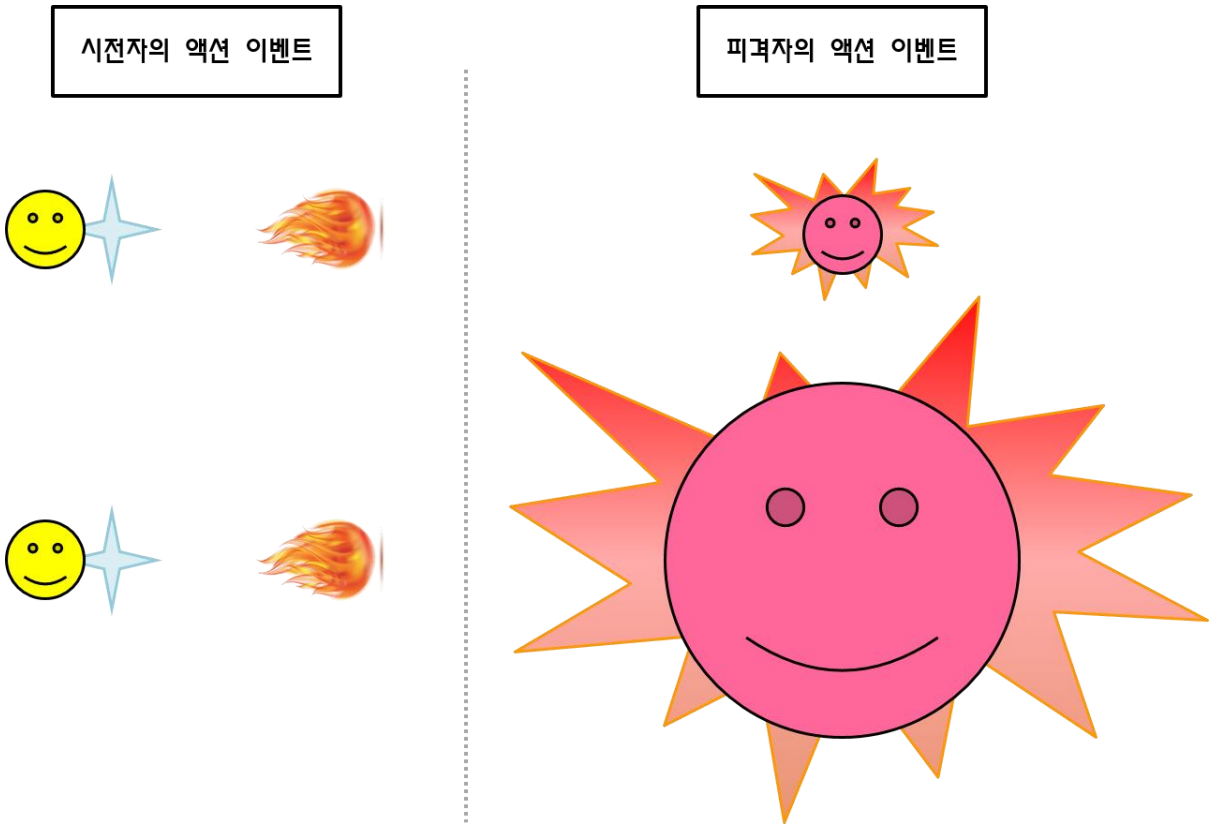
※ 위 방법과 더불어서, 특정한 기준이 될 캐릭터 크기(아마도, 플레이어 캐릭터들의 모델의 크기라고 봐야 한다.)에 맞는 이펙트들은 '표준 크기'로 놓고, 각 캐릭터들마다 자기 덩치에 맞는 이펙트의 크기 배율 값을 가지고 있게 하는 방법도 생각해봤다.
이 경우에는, 캐릭터 모델의 크기가 변하더라도 이펙트를 키우는 값을 일괄적으로 조절만 하면 되기 때문에, 한 번에 수정하기에는 쉽다.

그러나, 이펙트들을 특정한 표준 캐릭터 크기에 항상 맞춰서 제작을 해야 한다는 점은 아트 작업자들에게 있어 어색하게 느껴질 수 있다. 아주 웅장한 느낌으로 만든 이펙트인데, 인위적으로 특정한 크기 기준에 맞추다 보면, 제작할 때 제대로 느낌이 살지 않을 수도 있지 않을까?

이런 고민들을 하다가, 캐릭터들의 기본 크기가 변할 일이 그다지 많지 않을 것으로 생각해서, 액션 이벤트마다 관련된 이펙트들의 크기 배율을 지정하도록 하는 방식을 선택했다.
(캐릭터들의 기본 크기가 변한다면, 애니메이션 데이터들도 모조리 바뀌어야 해서 더 받아들이기 어렵다. 이런 경우에는 같은 모델을 크기 배율만 바꿔서 쓰는 게 더 옳은 선택이다.)

- **B-13-5-3.** 이펙트를 연출할 때, 연출할 대상의 크기에 적절하게 맞추기 위해서, 하나의 이벤트 흐름을 소유자에 따라 여러 개의 이벤트로 쪼개서, 적절한 시점마다 연속적으로 호출하는 방식

으로 구현할 수 있어야 한다.



<소유자의 이벤트에 따라 다른 이펙트 크기가 연출된다.>

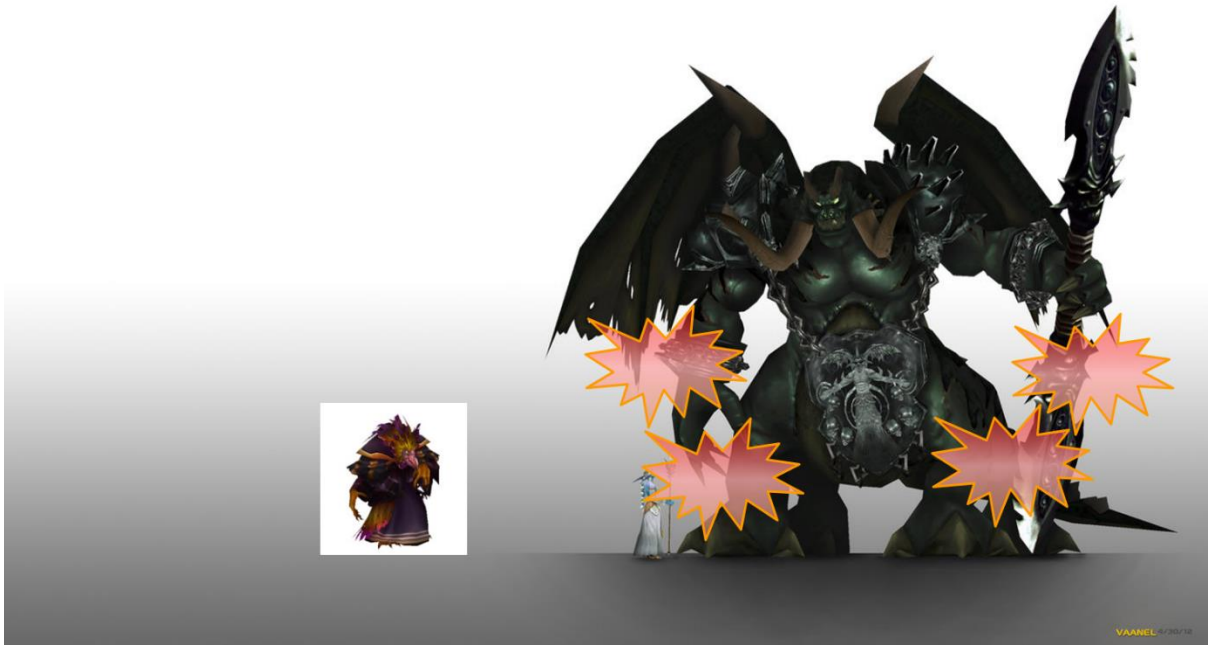
※ 위 그림에서처럼, 불덩어리 스킬을 적에게 발사하는 이벤트가 존재한다고 하자.
시전자가 불덩어리를 발사할 때, 불덩어리가 처음 생성되는 지점에 반짝이는 이펙트가 존재해야 한다. 그리고 불덩어리에 피격된 상대는 피격 이펙트를 연출해야 한다.
만약 이게 시전자 측의 하나의 이벤트 흐름으로 구현되어 있다고 하면 문제가 발생한다.
시전자가 발사한 화염구의 크기와 시전 이펙트는 시전자가 소유자이기 때문에 적절한 크기를 맞출 수 있다. 그런데 게임 플레이 중에 어떤 크기의 적이 피격될지는 누가 알까?

사실 이건 아무도 모르는 사항이다. 그때그때 다를 수 밖에 없는 문제다. 시전 -> 피격까지 하나의 이벤트 흐름으로 구성하는 게 문제가 되는 부분이 이러한 상황들이다. 심지어, 이걸 피격자 측의 이벤트로 바꿔도 문제는 마찬가지다. 피격자 측에서는 시전자가 누가 될지 모르기 때문에, 적절한 이펙트 크기를 정해줄 수가 없다.

그러므로 해결책은 위 그림의 회색 선이 분리한 것처럼, 불덩어리 시전 -> 피격에 대한 이벤트 흐름을 각각 '불덩어리 시전' 부분과 '피격' 부분으로 분리하는 것이다. 물론, 이렇게 구성하려고 하면, 처음에 일체형으로 제작하던 방식보다는 좀 더 '복잡해지긴 한다.'
하지만, 적어도 이펙트의 소유자는 명백하기 때문에, 불확실하거나 유동적인 매개변수로 인해 복잡한 추론 코드를 추가해야 하는 상황을 막아줄 수 있다.

- **B-13-5-4.** 이펙트의 **크기는, 반드시 이펙트 소유자의 모델 크기와 배율을 따르지 않아도 된다.**
: 반드시 모든 이펙트들이 모든 상황에서 이펙트가 붙을 대상의 크기에 맞춰야만 하는 건 아니다.
이펙트의 크기는 이펙트가 발생하는 액션이 정의하는 상황에 따라 유연하게 설정할 수 있어야 한다.

※ 이를 구현 단계에서 지나치게 복잡하게 생각할 필요는 없다.
이펙트를 재생하는 이벤트는 특별한 경우가 아닌 한, 100% 액션 이벤트를 통해야 하는데, 액션이 정의하는 사항이 달라지는 경우에는, 같은 동작을 표현하더라도 Action Code 가 다른 별개의 액션 데이터를 만들기 때문이다.
즉, '회오리베기' 스킬이 있다고 할 때, 일반 몬스터의 '회오리 베기 피격'과 보스 몬스터의 '회오리 베기' 피격의 이펙트 재생 방식이 달라야 한다고 하면, 이들은 '일반 몬스터 회오리베기 피격' 액션과 '보스 몬스터 회오리베기 피격" 액션으로 나뉘는 방식으로 구성한다.



<액션의 분리를 통해, 이펙트 이벤트도 경우에 맞게 조절할 수 있다.>

- **B-13-5-5.** 캐릭터의 신체 일부에 붙어서 따라 다니는 이펙트의 위치 변환은, 캐릭터의 현재 위치만을 따르는 경우와, 캐릭터의 현재 회전 값을 따르는 경우로 분리해야 한다.

※ 예를 들어, 캐릭터의 파츠 일부에 붙어 있게 보여야 하는 이펙트들은, 캐릭터가 회전하면 그 회전을 같이 따라가줘야 한다. 이런 경우는 이펙트가 캐릭터의 현재 회전 값까지 같이 따르는 경우이다. (기술적으로 말하자면, 캐릭터가 회전했을 때의 최종 위치 좌표를 따라간다.)

반면, 캐릭터의 발 밑에서 플레이어에게 어디로 가야 하는지 알려주는 방향 지시자의 경우를 보자. 진행 방향 지시자는 캐릭터가 방향을 바꾼다고 해서 자기도 같이 방향을 바꾸면 안 된다.(...) 진행 방향 지시자답게, 캐릭터의 발 밑을 따라다니되, 방향 자체는 진행 방향을 그대로 유지해야 한다.

◆ B-13-6. 문자열로 표현해야 하는 이펙트

- **B-13-6-1.** 이펙트를 문자열로 표현해야 하는 경우, 가급적 **숫자와 영어 문자, 자주 쓰는 특수 기호들(. , ? ! : " ' 정도)만을 사용하도록 디자인**한다.
: 최악의 경우에도, 표준 ASCII 코드에서 지정하는 문자열의 범위를 넘지 않게 한다.

※ 만약, 각 언어권마다 고유한 문자들을 이펙트에 쓰게 되면, 나중에 소프트웨어를 지역화할 때 큰 골칫거리를 만들 수가 있다.
왜냐하면, 문자열로 표현하는 이펙트들에 쓰이는 '문자'들은, 폰트 파일에 의해 그리는 문자열로 취급하기보다는, 문자의 모양을 가진 그림으로 보는 편이 실제 구현 방식에 더 가깝기 때문이다.

사실 대부분의 게임에서는, 이펙트에 쓰이는 몇 개 글자 때문에 그걸 표현해주려고 아예 전용 폰트를 만드는 경우는 거의 없다. (Blizzard 사의 World Of Warcraft 라든가, Diablo 시리즈의 경우가 있기는 하지만, 극소수이니 제외...)
이런 상황에서, 전세계적으로 공통으로 쓰는 문자가 아닌, 특정 언어권에서만 쓰는 문자열을 이펙트로 쓴다면, 언어권이 다른 지역에 서비스할 때마다 그 언어권에 맞는 새로운 이펙트 리소스를 제작해야 한다는 의미이다.
경우에 따라서는 그런 정책이 옳을 수도 있겠으나, 가급적 과도한 노력이 들어가지 않도록 하기 위해, 문자열로 표현하는 이펙트의 경우에는 전세계적으로 공통으로 사용하는 문자열만을 대상으로 한다.

B-14. 사운드

◆ B-14-1. 사운드 객체의 구조

- **B-14-1-1.** 객체 관리에 의한 타입 분류는 인스턴트 타입 사운드(Instant Type Sound)와 순차 타입 사운드(Sequence Type Sound)로 나눈다.

- **B-14-1-2.** **인스턴트 타입 사운드(Instant Type Sound)**
: **같은 사운드 객체를 여러 개 생성하여 동시에 재생할 수 있다.**
물론, 딱 1개만 생성하는 경우에는 순차 타입 사운드와 다를 게 없다.

- **B-14-1-3.** **순차 타입 사운드(Sequence Type Sound)**
: 사운드 객체를 한 번에 하나씩 재생한다. Fade in / Out 할 때는 잠시 2 종류의 사운드가 들릴 수 있으나, 원칙적으로 **한 번에 하나의 사운드만 재생**한다.

- **B-14-1-4.** 사운드의 재생 위치에 의한 타입 분류는 위치 기반(Coordination Type)의 사운드와, 전역 기반 (Global Type)의 사운드로 나눈다.

- **B-14-1-5.** **위치 방식 사운드(Coordination Type Sound)**
: 사운드 객체는 게임 월드에서의 특정한 좌표 위치에 생성되면, 사운드가 들리는지, 들리지 않는지는, 사운드 수신기 객체가 그 사운드의 좌표 위치에 얼마나 가까이 있는지에 달려 있다.
즉, 사운드가 생성되어 있는 위치에서 멀어지면, 그 사운드는 들리지 않는다.

※ 멀어짐을 판정할 때는 사운드의 진원지로부터의 거리를 기준으로 계산할 수도 있고, 화면의 시야 밖으로 사라졌는지 여부를 기준으로 할 수도 있다.

- **B-14-1-6.** **전역 방식 사운드(Global Type Sound)**
: 게임 월드의 어느 위치에 있든지 관계없이, 조건만 맞으면 사운드를 들을 수 있다.

※ 구현할 때는 위치 값에 상관없이 항상 사운드가 들리는 방식이거나, 아니면 위치 값을 가지는 사운드가 주 화면을 보여주는 카메라를 계속해서 따라다니는 것과 같다.

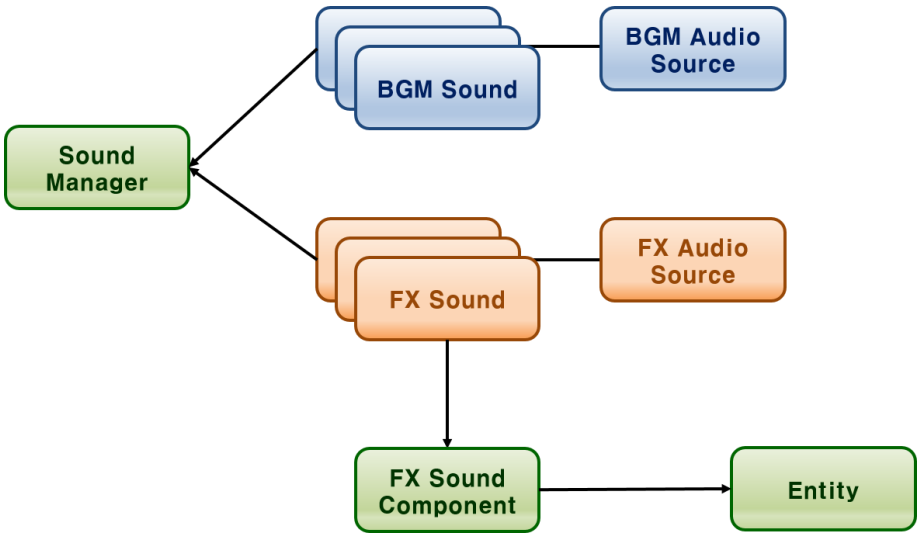
- **B-14-1-7.** 객체 관리에 의한 사운드 타입과 재생 위치에 따른 사운드 타입은 서로 혼용이 가능하다.

※ 모든 캐릭터들은 게임 세계에서 좌표 위치를 가지고 있기 때문에, 액션에 의한 FX 사운드들은 모두 인스턴트 타입이자 위치 방식 사운드로 구성하는 게 원칙이 된다.
하지만 내 캐릭터의 경우에는 조금 다를 수도 있다.
예를 들자면, 어떤 카메라 이벤트나 다른 인터페이스를 통해서 내 캐릭터가 잠시 멀어져 있거나, 혹은 화면에서 안 보이는 상황이라고 가정해보자. 그렇다고 할지라도, 내 캐릭터를 플레이어가 안 보고 있는 사이에 공격을 당하고 있다거나 할 때, 이를 알리기 위해서 거리와 상관없이 플레이어가 내 캐릭터에게서 발생하는 액션에 의한 소리를 듣게 만들고 싶을 수도 있다.
이런 경우에는 [인스턴트 사운드 + 위치 방식 기반 사운드]로 만드는 게 아니라, [인스턴트 사운드 + 전역 방식 사운드]로 만들어야 의도에 맞게 동작할 것이다.

◆ B-14-2. 사운드 처리 방식

- **B-14-2-1.** 각종 전투 관련, 스킬 사용, 사운드 이펙트, 이벤트에 따른 환경음 재생 등은 모두 인스턴트 사운드로 취급한다.

- **B-14-2-2.** 캐릭터 개체의 위치를 따라다녀야 하는 FX 사운드 객체들은, 그 캐릭터 개체의 FX 사운드 객체의 관리 컴포넌트가 소유한다.
: 캐릭터 개체들은 다수의 컴포넌트를 소유한다. 즉, 캐릭터 개체를 따라다니는 FX 사운드 객체들은 캐릭터 개체가 소유하고 그 수명을 관리한다.



<사운드 종류에 따라 소유자와 관리 방식에 차이가 있다.>

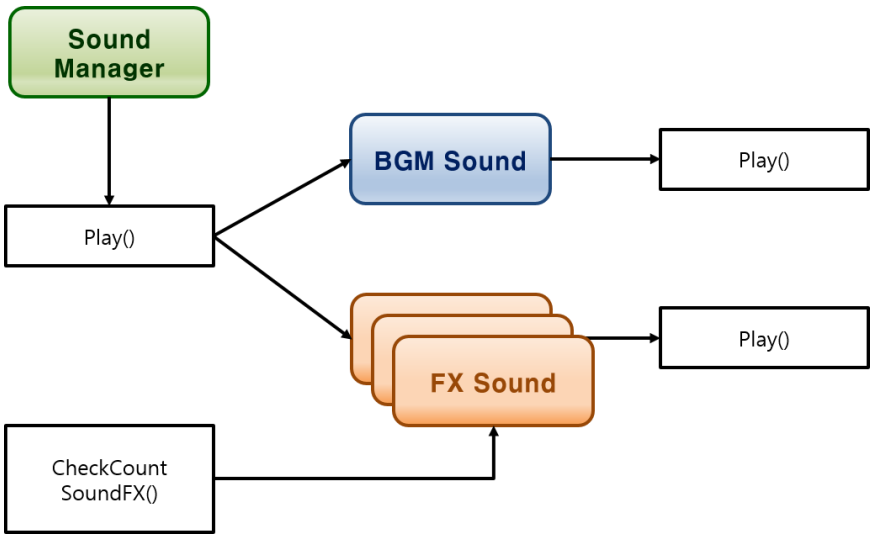
- **B-14-2-3.** 인스턴트 사운드들의 재생 여부를 선택할 수 있어야 한다.
: 이 재생 여부에 따라 모든 인스턴트 사운드가 일괄적으로 재생되거나, 혹은 재생되지 않는다.

- **B-14-2-4.** 배경 음악은 이벤트에 따라 교체할 수 있어야 한다.

- **B-14-2-5.** 배경 음악은 반드시 [순차 타입 사운드 + 전역 타입 사운드]로 취급해야 한다.

- **B-14-2-6.** 배경 음악의 재생 여부를 선택할 수 있어야 한다.

- **B-14-2-7.** 배경 음악의 재생 여부는 다른 인스턴트 사운드들의 재생 여부를 간섭하지 않고, 간섭 받지도 않는다.
: 순차 타입의 사운드들이 모두 인스턴트 사운드의 재생 여부와 관계 없이 독립적으로 재생하지 않는다는 점을 주의할 것. 오직 배경 음악에 대해서만 이야기하고 있다.



<배경음은 동시에 1개만, 그리고 다른 FX 소리 객체들과 독립적으로 재생한다.>

- **B-14-2-8.** 모든 사운드 객체에 대해서, 3차원 위치에 기반하여, 소리의 방향과 강도를 조절하는 기능은 필수적이지 않다.
: 게임 엔진에서 그러한 기능을 지원하지 않는다면, 구현하기 위해 필요한 노력이 많이 드는데다, 게임의 특성상 그다지 중요하지 않은 요소이다.

※ Unity 엔진에서 사운드 컴포넌트는 3D 위치에 의해 소리의 방향과 강도를 조절하는 기능을 가지는 사운드를 별도의 큰 노력 없이 구현할 수 있다.

그러나, 기본적으로 3D 위치 기반 사운드는 전반적으로 3D 위치를 고려하지 않는 사운드보다 성능을 더 많이 차지한다. (거리에 따른 소리 강약을 도플러 효과로 조절하는 기능은 단순히 거리에 따라 소리를 켜고 끄는 것보다 더 복잡한 작업이다.)
또한, 이 게임의 구성 요소로 비추어 봐도, 소리가 들리는 거리의 멀고 가까움을 플레이어가 꼭 느껴야 할 필요가 없다.
(1인칭 슈팅 게임인 경우에는 소리의 거리감이 게임의 사용자 경험에 큰 영향을 미치지만, 탑뷰 – 쿼터 뷰 방식의 RPG에서는 그다지 중요한 요소라고 볼 수 없다.)

- **B-14-2-9.** 여러 개의 사운드 이벤트가 동시에 발생할 수 있어야 한다.
: 하나의 사운드가 재생되는 도중에, 다른 사운드가 '끼어들어서' 재생하는 게 가능해야 한다.

※ Unity3D 엔진을 사용할 경우, 그다지 신경 쓰지 않아도 되는 사항이다.

◆ B-14-3. 사운드 재생 조건

- **B-14-3-1.** 재생할 사운드 포맷
: **ogg, mp3** 포맷을 사용한다.

※ Unity 엔진 관련 포맷에서는 ogg 또는 wave 포맷을 사용하기를 추천한다.
Unity 공식 문서에서는 모바일 디바이스를 타겟으로 한 빌드에서는 오디오 포맷은 일반적으로 mp3로 인코딩한다고 되어 있다. (ogg, wave 사용을 추천한다면서...?)

ogg, mp3의 사용을 추천하는 이유는, 이들이 **손실 압축 방식이기 때문**에, 압축을 통해서 **음질을 어느 정도 유지하면서도 효과적으로 용량을 줄일 수 있기 때문**이다.
사운드 데이터는 범용적으로 사용하기 어려운 데이터일 경우가 많으므로, 개수가 많아지면 메모리 사용량에 있어 부담을 준다.
비록 wave 형식은 손실 데이터가 없으므로 가장 정확한 음을 들려주겠지만, 모바일 기기는 음향 전문 기기가 아니고, 모바일 게임도 음악만을 위해 존재하지 않는다. 그러므로 게임 내 사운드 데이터는, 이상하게 들리지만 않을 정도까지는 음질을 희생하더라도 사운드 데이터의 용량을 줄여줄 필요가 있다.

- **B-14-3-2.** 전화가 오는 경우, 모든 게임 관련 사운드는 자동으로 꺼져야 한다.
인터럽트가 종료된 뒤, 다시 프로그램을 실행시킬 때는, 기존에 사용자가 선택한 옵션대로 되돌아간다.

※ 이 부분은 굳이 별도의 구현이 필요하지 않은 것으로 보인다.
이미 사용하는 엔진(Unity3D) 단계, 또는 운영 체제 단계에서 알아서 처리하는 것으로 확인하였다.

- **B-14-3-3.** 재생하는 인스턴트 사운드의 개수는 일정한 개수 이내로 제한할 수 있다.
: 구체적인 개수는 구현할 때 성능 측정 결과에 따라 결정한다.

- **B-14-3-4.** 재생하는 인스턴트 사운드의 개수를 제한하는 방식은, 일종의 **허용하는 목표치를 제공하는 방식**이다.
현재 재생하는 전체 개수가 목표치를 초과하는 경우에 대해서는 기존 재생 사운드들을 중간에 자르지 않고 허용하지만, 한 번에 목표치 이상의 인스턴트 사운드를 생성하지는 않는 방식

이다.

※ 이를테면, 인스턴트 사운드를 10개로 제한하는 경우, 한 번에 10개를 넘어서 재생하지 않는 것일 뿐이다. 만약, 이미 5개의 다른 인스턴트 사운드가 재생되는 중이라면, 그대로 10개의 인스턴트 사운드를 추가해서 총 15개의 인스턴트 사운드가 재생되는 상태라도 그대로 허용한다는 의미이다.
이렇게 하는 이유는, 시각적으로 연출하는 FX들과 달리, 사운드는 청각으로 느끼기 때문에, 한 번 재생하면, 중간에 뚝 끊어버릴 수 없는 연출이라서 그렇다. (소리를 중간에 뚝 끊어버리면 아주 어색하게 느껴진다.)

- **B-14-3-5.** 재생하는 인스턴트 사운드의 개수 제한을 셀 때, **배경 음악(BGM)과 이야기 연출(나레이션, narration) 사운드들은 포함하지 않는다.**
: 이들은 필수적으로 재생해야만 하는 사운드들이다.

- **B-14-3-6.** **빠른 재생 전환이 필요하고 용량이 작은** 사운드 객체에 쓸 오디오 소스들은 **미리 메모리에 올려두고 사용**한다.
: 각종 타격 / 피격 음성, 스킬 관련 FX 사운드들이 이에 해당한다.

- **B-14-3-7.** **재생 전환 속도가 그다지 중요하지 않고, 용량이 큰** 사운드 객체에 쓸 오디오 소스들은 **디스크로부터 직접 불러와서 사용**한다.
: 배경음이 대표적이다.

- **B-14-3-8.** 모든 사운드의 재생 대역폭은 가급적 32Kbps 수준까지 억제한다.
: 원본 오디오 소스와 비교하여 아무래도 영 이상하게 들리는 경우에는 좀 더 대역폭을 넓게 가져갈 수 있다.

※ 사운드의 대역폭을 압축하면 용량이 줄어들지만, 손실 압축이기 때문에 음질도 저하된다.
어느 정도까지를 허용할지는 정말 사람의 느낌에 대한 부분이라, 수치를 통해 표현하기는 어려운 부분이다. 단지, 원본 오디오 소스의 품질 그대로를 사용하면 메모리 사용량이 매우 늘어날 가능성이 있기 때문에, 허용 가능한 음질 수준 안에서 최대한 압축해서 사용해야 한다.

◆ B-14-4. 음성 관련 기능

- **B-14-4-1.** 음성 대화 기능
: 해당 사항 없음.

- **B-14-4-2.** 음성 인식 기능
: 해당 사항 없음.

- **B-14-4-3.** 특정한 현실의 언어를 이용한 음성 데이터는 사용하지 않는다.
: 대표적인 예로는 음성 안내 메시지에 사용하는 사운드 데이터가 있다. 이야기의 Narration을 위한 데이터 역시 마찬가지다.

※ 배우들이 녹음해야 하는 음성 데이터는 결코 공짜가 아니다.
배경음이나 환경음 제작 역시 그 점에 있어서는 마찬가지지만, 언어를 사용하는 음성 데이터의 가장 골치 아픈 점은, 수정해야 할 때의 비용이 매우 크다는 점이다.
글자로 되어 있는 텍스트 데이터는 그냥 글자를 수정해버리면 그만이다. 그림 데이터인 경우에도, 어차피 개발사에서 직접 개발한 데이터인 경우가 많으므로 대부분은 수정이 용이하다.
그러나 음성 데이터는 전혀 그렇지 않다. 텍스트로는 단 한 글자, 한 단어의 수정일지라도, 음성 데이터로는 전체를 다시 제작해야 하는 공정이 된다.
따라서, 음성 데이터를 이용하고자 할 때는, 비용 문제와 데이터 내용의 고정(freezing) 시점을 반드시 고려해야 한다.

※ 추가적으로, 현실 세계의 특정 언어를 이용하는 경우, 소프트웨어를 수출하는 경우에 지역화 관련된 이슈가 따라오게 된다.
특별히 음성 데이터의 언어 때문에 지역화를 할 생각이 없었다면, 이 부분은 예상 외의 부담스러운 이슈가 될 수도 있다. 음성 데이터를 현지화하는 비용은, 텍스트 데이터를 현지화하는 비용보다 훨씬 비싸기 때문이다.

- **B-14-4-4.** 음성 데이터가 존재해야 하는 경우, 음성 데이터를 텍스트 데이터로 대체할 수 없는 경우를 제외하고, **사용자가 음성 데이터의 다운로드 여부를 선택할 수 있어야 한다.**
: 음성 데이터가 여러 개인 경우 용량이 매우 커질 수 있다. 사용자가 이를 선택할 수 없다면, 불필요한 내려 받기이므로 성가시게 느낄 수 있다.

B-15. 조명 처리

◆ B-15-1. 조명 처리 정책

- **B-15-1-1. 환경 광원(Ambient Light) 외에는, 실시간으로 텍스처의 각 픽셀 단위마다 계산하는 조명은 사용하지 않는다.**
: 사실상, **게임을 실행할 때는 실시간 광원을 주는 조명 객체를 사용하지 않는다.** 실시간 조명은 성능에 있어 상당히 큰 영향을 주는 것으로 알려져 있다.

※ Unity 엔진에서의 그동안 연구 / 테스트 결과, 실시간으로 조명을 픽셀 단위로 계산하는 행위는 투명도 점검(Alpha Testing) 다음으로 성능 하락에 영향으로 미치는 것으로 보인다.

- **B-15-1-2.** 위치가 고정되어 있고, 세밀한 조명 효과가 들어가야 하는 부분은 조명 맵(Lightmap)을 사용한다.
: 지형 맵이 이러한 경우에 가장 적합하다.

※ Unity에서 조명 맵을 사용하기 위해서는, 조명 맵을 구울(Bake) Mesh들이 미리 해당 *.unity 파일의 객체 계통(Hierarchy)에 올라와 있어야 한다. 그렇지 않으면 조명 맵이 올바르게 작동하지 않는다.
Unity 조명 맵의 이러한 특성은 게임 맵마다 *.unity 파일이 따로 생성되어야 하는 이유 중 하나이기도 하다.

- **B-15-1-3.** 게임에서 동적으로 로딩해야 하는 모델 객체들은 **조명의 영향을 받지 않는 Shader를 사용**해야 한다.

※ 동적으로 로딩해야 하는 모델 객체들은 조명 맵을 사용할 수 없다. 또한 실시간으로 계산하는 조명도 사용할 수 없기 때문에, 조명에 대한 계산을 하지 않는 Shader를 사용해야 올바른 텍스처 색상을 표현해줄 수 있다.

- **B-15-1-4.** 1개의 환경 광원(Ambient Light)의 사용을 허용한다.

※ Unity에서 환경 광원은 전역 렌더링 설정에서, 각 unity 파일마다 1개씩만 줄 수 있는 것으로 알고 있다.
다른 방식으로도 환경 광원을 줄 수 있는지는 조사해봐야겠다. (뭐, 1개 이상 필요가 없다고는 하지만 말이다...)

◆ B-15-2. 조명 맵(Lightmap) 내용

- **B-15-2-1.** 조명 맵은 단일 조명맵(Single Lightmap)으로 구성한다.

※ Unity에서는 Dual Lightmap과 Directional Lightmap도 지원은 하지만, 모바일 기기에서 조명 맵을 Dual Lightmap으로 할 필요가 없다.
왜냐하면, Unity 공식 문서에서 따르면, Mobile 기기에서는 Forward Render를 사용할 수 없기 때문이다.
적용해봐야 공연히 텍스처 용량만 두 배씩 차지할 뿐이다.

- **B-15-2-2.** 조명 맵으로 사용하는 텍스처의 크기는 **최대 한 장당 1024 × 1024(픽셀)**로 한다.

※ Unity에서 1024 × 1024 픽셀 크기를 초과하는 조명 맵을 만들 수 있는 옵션이 있기는 한가?
그냥 1024 × 1024 픽셀 크기의 텍스처를 여러 장 만들어내는 것 같다.

- **B-15-2-3.** Light Probe는 사용하지 않는다.
: Light Probe의 성능 부담은 '일반적인 조명 객체'에 의한 조명 계산보다는 훨씬 가볍지만, 그렇다고 해서 성능 비용이 공짜인 것은 아니다. ~~(격자로 배치해야 효율이 좋은데, 그렇게 구성하기 위한 노가다가...)~~

※ 그래픽 팀에서 검토한 결과, 이 게임의 3D 모델 객체에 대해 Light Probe를 반드시 사용해야 할 필요성이 없다고 판단하였다.

◆ B-15-3. 그림자 처리

- **B-15-3-1.** 일반적으로, 실시간으로 Mesh 모양을 따라 생성하는 그림자는 사용하지 않는다.
: 모두, **미리 그려진 그림자 텍스처를 입힌 사각형 판(Panel) 객체를, 모델의 발 아래에 붙어 다니게 하는 것으로 대신**한다.

※ OpenGL ES 3.0 이상을 지원하는 모바일 기기들은 Unity 4.x 이상의 버전을 이용한 응용 프로그램에서 실시간 그림자를 이용할 수 있다.
하지만 그것을 위한 연산량을 고려해보면, 다수의 캐릭터 개체가 등장하는 게임에서는 효과적이라고 보기 어렵다. 모바일 기기들이 언제나 배터리에 의존해야 하는 점을 고려해야 한다.

<사각형 Mesh로 만든 가짜 그림자>

- **B-15-3-2.** 예외적으로, **내 캐릭터에 대해서만큼은 실시간으로 생성하는 그림자 기능을 사용할 수 있다.**
  : 대부분의 사용자들이 가장 많이 봐야 하는 캐릭터 개체는 바로 내 캐릭터이기 때문에, 예외적으로 내 캐릭터에 대해서만큼은 실시간 그림자 렌더링을 허용할 수도 있다,.

- **B-15-3-3.** 아주 중요한 캐릭터는 내 캐릭터가 아니라도 실시간으로 생성하는 그림자 기능을 사용할 수 있다.
  : 예를 들자면, 보스 몬스터의 경우에는 덩치도 크고, 주목도가 높기 때문에 좀 더 고품질의 연출을 해야 할 수 있다.

※ 결국 모두 다, '비용 대 효과'의 문제일 뿐이다.

B-16. 게임 상태값

◆ B-16-1. 게임 상태값(Game Status)

- **B-16-1-1.** 게임 상태값은 **게임 플레이 중에 각종 판정을 위해 사용하는 값**들을 말한다.
: 일반적으로 '능력치'라는 말로 표현하는 값이지만, 실제로는 능력치를 포함해서 더 광범위한 내용들을 포함하는 개념이다.

※ 명세서에서 '능력치'라는 말을 많이 썼지만, 사실 '능력치'의 실제 내부 구조는 그것보다 더 복잡하다.
사실 명세서 전반에서 쓰이고 있는 '능력치'라는 용어의 실제 구현 내부상 분류로 봤을 때는 이 수치들은 '게임 상태값(Game Status)'라고 불러야 더 적합하다.

단지, '능력치'라는 용어 자체가 일반적으로 훨씬 더 광범위하게 사용하고 있고, 따라서 의사소통 하는데 더 편리하다. 명세서의 다른 부분에서는 굳이 능력치에 관한 세부적인 분류를 명백하게 할 필요는 없기 때문에, 얼른 이해가 안 되는 용어를 피하려고 의미를 넓혀서 사용했을 뿐이다.

- **B-16-1-2.** 각 게임 상태값들은 수행치를 도출해내기 위해, 특별한 값 공식(Fomula)에 의한 결산 수치 (Computation Result)'로 구한다.

- **B-16-1-3.** 게임 상에서 사용하는 게임 상태값 구조체 다음 세 부분으로 이루어진 구조로 되어 있다.

수치 분류	설명
능력 수치 (Ability)	• 게임에서 분류하는 각 능력의 계산 공식에 의거해 계산하는 값 : 예를 들자면, 물리 공격 수행치는 아래와 같은 계산을 거친다고 가정하자. **(물리 공격력) = (힘) × 3.0** 이 때, (힘)에 해당하는 게 바로 여기서 말하는 능력 수치다. • 보통 사용자에게는 이런 식으로 인식된다. : 힘 +5(힘 능력 공식에 의한 전투 수치에서는 물리 공격력 +20), 지능 -3(지능 능력 공식에 의한 전투수치에서는 마력 -15 등)
상수	• 능력 수치(Ability)와 다르게, 계산 공식에 넣는 값이 아니라, 넣어진 숫자 값

(Constant)	그 자체를 더한다. : 쉽게 예를 들자면, 생명력 +250, 물리 공격력 -15과 같은 방식으로 사용한다.
백분율 (Percent)	• 능력 수치(Ability)와 상수(Constant)의 합으로 나온 결과에 더할 백분율 : 쉽게 예를 들자면, 공격력 10% 증가, 방어력 5% 감소 등에 사용하는 값이다.
결산치 (Result)	• 능력 수치(Ability), 상수(Constant), 백분율(Percent)의 각 수치 부분이 결과로써 내놓는 통일된 단위의 능력 값 • 게임 상태값은 위 3개의 수치 부분 결산치들의 총합이다. • **게임에서의 전투 판정은 이 결산치를 가지고 수행한다.**

- **B-16-1-4.** 게임 상태값(=수치 분류에 나온 결산치)를 게임에서 사용하는 방식은, 개별 수행치의 사용 정의에 따른다.
: 이에 대해서는 **[A-1-3-5]**에 나오는 표에서, **[전투 수치 변환]** 열을 참고한다.

※ 게임 상태값은 반드시 상수로 사용한다는 보장이 없다.
전투를 예로 들어보면, 공격력에 대한 부분은 400 ~ 450 사이의 피해를 주는 방식으로 상수로 사용하는 경우가 대부분이다. (이 게임도 그렇다.)
그런데, 방어력에 대한 부분은 예를 들면 300 이라는 방어 수치(수행치)가 있으면, 이 수치가 물리 피해치 400 에서 300 방어 수치 그대로 감산해서 100 의 피해를 입히게 만드는지, 아니면 방어력 300 이라는 수치가 피해를 80% 감소시키는 효과가 있어서 400 – 320(= 400 × 0.8) = 80 의 피해를 입히게 만드는지는, 전투 수치를 사용하는 방법을 정의하는 방식에 따라 다르다.

- **B-16-1-5.** 게임 상태값의 분류
: 전투 등의 판정에서 사용하는 게임 상태값은 분명 하나의 '최종값'으로 사용한다.
하지만 이 '최종적인 게임 상태값 값'은 그냥 명시되어 있는 상수 리터럴(Literal)이 아니다. 캐릭터의 종류, 레벨, 착용한 아이템, 적용된 부여효과 등에 매겨져 있는 게임 상태값들이 합쳐진 것이다. 일반적으로, 다음과 같은 종류의 **게임 상태값들이 모두 합쳐져 '최종값'을 만들어낸다**

1. 캐릭터 개체 종류별 기본 게임 상태값
: 이 부분은 캐릭터의 레벨에 비례해서 올라가는 능력 값, 혹은 사용자가 선택해서 향상해주는 (그러니까 스탯 올리는 거 있잖아...) 능력 값 같은 게 아니다. 그냥 캐릭터가 존재하기 때문에 기본적으로 붙는 능력 값들을 말한다.
예를 들면, 기본 이동 속도, 기본 공격 속도 등은 아주 예외적인 경우가 아니라면 보통 0으로 설정되어 있지는 않을 것이다. 이 부분들에 대한 능력 값들은 캐릭터의 존재와 늘 따라다니며, 당연히 게임 상태값의 구조체로 표현하는 값이다.

2. 캐릭터 개체 종류별로, 레벨 당 향상되는 게임 상태값
: 캐릭터들은 보통 레벨 단계마다 캐릭터의 '강한 정도'를 나타내는 능력 값들을 종류마다(힘, 민

첩성, 체력, 공격력...) 가진다. 캐릭터가 존재하는 한, 이런 종류의 능력 값들은 언제나 존재할 수 밖에 없으므로, 최종적으로 계산되는 게임 상태값은 최소한 이 부분만큼은 포함할 것이다.

3. 캐릭터가 가지고 있는 장비 아이템들의 능력이 되는 게임 상태값들의 합

: 캐릭터가 아이템을 착용하게 되면, 아이템들마다 붙어 있는 능력들이 더해져서 캐릭터를 더 강하게 만든다. (RPG에서는 아주 상식인 얘기다.) 아이템들의 능력 값을 표현하는 형태 역시 게임 상태값의 형식이다.

아이템을 착용한 경우, 최종적인 게임 상태값의 합산에 반영한다.

4. 캐릭터에게 부여된 부여효과들에서 비롯된 게임 상태값들의 합

: 캐릭터에게 부여된 부여효과들도 역시 게임 상태값들로 표현하는 능력 값들로 이루어져 있다. 부여효과가 걸려 있을 경우에는, 이것도 최종적인 게임 상태값의 합산에 반영한다.

<게임 캐릭터의 총 능력 값(게임 상태값)을 구성하는 각 능력 값들의 분류>

- B-16-1-6. 게임 상태값의 합산 공정

: 간단히 요약하면, 합산한 수행치들의 수치 부분들을 각각 더한 다음, 최종적으로 더한 수치 부분들의 결산치를 구해서 최종적인 게임 상태값의 값을 얻어낸다.

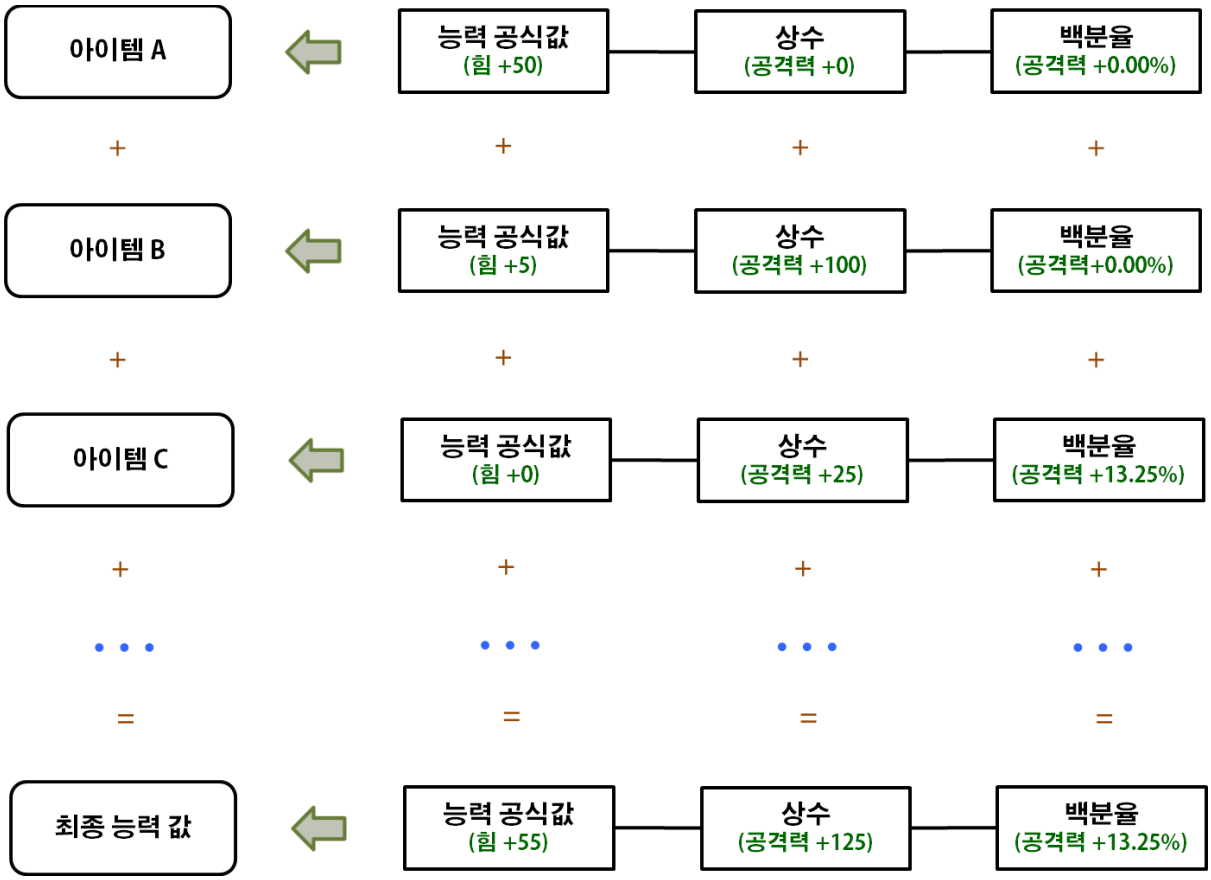
1. 각 게임 상태값 부분 중, **능력 수치(Ability) 부분끼리 전부 더한다.**
2. 각 게임 상태값 부분 중, **상수(Constant) 부분끼리 전부 더한다.**
3. 각 게임 상태값 부분 중, **백분율(Percent) 부분끼리 전부 더한다**
4. **모두 합한 게임 능력치, 상수, 백분율을 합**하여 최종 결과값을 구해낸다.

※ 이런 방식을 쓰는 이유는 **여러 개의 능력을 더하거나 곱하는(혹은 빼거나 나누는) 순서에 상관없이 같은 결과값이 나오도록 하기 위함**이다.

사실, 여러 개의 능력 값들을 어떻게 합산하는지는, 게임 규칙에서 정하기 나름이다. 하지만, 여러 가지 아이템, 스킬 등의 부여효과들의 능력을 일관성 있는 규칙대로 합산하기 위해서는 일정한 기준이 필요하다.

이를테면, 공격력 100 에 아이템의 공격력 +50 을 더하고, 그 다음에 치명타 피해 ×2 를 적용하는 경우(결과는 300)와, 공격력 100 에 치명타 피해 ×2 를 적용하고 아이템의 공격력 +50 을 더하는 경우(결과는 250)는 최종 결과값이 완전히 달라진다.
위와 같은 사례는 심지어 여러 개의 아이템의 능력, 혹은 여러 개의 부여효과의 능력을 합치는 순서에서도 마찬가지로 발생할 수 있다. 더구나 능력 값을 표현하는 게임 상태값이 능력 공식에 의한 능력 수치(Ability)일 수도 있고, 그냥 상수값으로 향상시키는 경우도 있고, 방어력 xx%하는 식으로 백분율일 수도 있다.

그러나, 아래와 같은 개념의 합산 방식을 사용한다면, 능력 값들을 합치는 순서에 관계없이 항상 일정한 결과를 도출할 수 있으며, 각 단계별 수치도 정확하게 추적할 수 있을 것이다.



<여러 개의 아이템들의 게임 상태값을 합산하는 방식의 예>

- **B-16-1-7.** 게임 상태값의 합산 공식은 다음과 같다.

(합산치) = (능력치 공식 결과값) + (상수) +

{ (능력치 공식 결과값 + 상수) × (백분율) }

※ 예를 들자면 이렇게 계산하게 된다. (설명의 편의상 아이템들의 능력치 종류를 제한하였다.)

[마왕의 집행검] : 공격력 400 ~ 450, 힘 +100(단 힘은 1 당 공격력 3 증가시켜 줌)
[학살자의 흉갑] : 방어력 500, 방어력 7.5%, 힘 +30, 공격력 +75
[언덕거인의 반지] : 공격력 +10%, 방어력 5%

위 아이템들을 캐릭터가 장비한다고 치자. 그러면 아이템에 의해 향상되는 게임 상태값들의 합은 다음처럼 계산한다.

1. 우선 공격력의 경우를 보자

아이템	게임 상태값	능력치	상수	백분율(%)
마왕의 집행검	•	100	400 ~ 450	0
학살자의 흉갑	•	30	75	0
언덕거인의 반지	•	0	0	10
합계	951.5 ~ 1,006.5	130	475 ~ 525	10

힘 1 당 공격력 3 증가라고 가정했으므로, 힘에 의한 공격력 증가는 130 × 3 = 390 이다.
상수 부분은 집행검(~~리니지의 집행검이 생각난다면 그것은 기분 탓이다.~~)과 흉갑에만 있으므로, 합치면 475 ~ 525 가 된다. 따라서, 390(= 130 × 3) + 475(혹은 525) = 865 ~ 915 의 값을 얻을 수 있다.
아직 백분율 부분이 남아 있다. 백분율은 반지에만 존재하고 +10% 공격력 증가인데, 어떤 값의 10%인가 하면, 아까 구했던 능력치 공식의 결과 값과 상수 부분 값을 합해서 나온 값의 10%이다. 즉, 아까 계산해서 나온 865 ~ 915, 요 값의 10%를 다시 더한다.
그러면 865 + (865 × 10%) 혹은 915 + (915 × 10%)라는 의미이므로, 계산하면 951.5 ~ 1,006.5 의 값이 최종적으로 나온다.

2. 방어력에 대한 부분은 아래처럼 산출한다.

아이템	게임 상태값	능력치	상수	백분율(%)
마왕의 집행검	•	0	0	0
학살자의 흉갑	•	0	500	7.5
언덕거인의 반지	•	0	0	5
합계	562.5	0	500	12.5

1.에서 했던 계산 과정과 같다. 단, 능력치 부분은 셋 다 0 이므로 그냥 넘어가면 되고, 상수 부분도 흉갑에만 500 있다.
중요한 부분은 백분율이다. 흉갑과 반지 둘 다 능력이 존재하는데, 백분율 수치를 그냥 더한다.
그래서 12.5%의 결과가 나왔고, 1.과 마찬가지로 게임 상태값의 합을 구하는 공식에 의해 다음처럼 합계를 구한다.
562.5 = 0 + 500 + { (0 + 500) × 12.5% }

◆ B-16-2. 게임 상태값의 적용

- **B-16-2-1.** 미리 설정된 공식과 규칙에 의해 결과로 도출한 게임 상태값의 단위와, 실제로 게임 플레이의 판정에 이용할 때, 어떤 단위로써 적용할지 여부는 다를 수 있다.

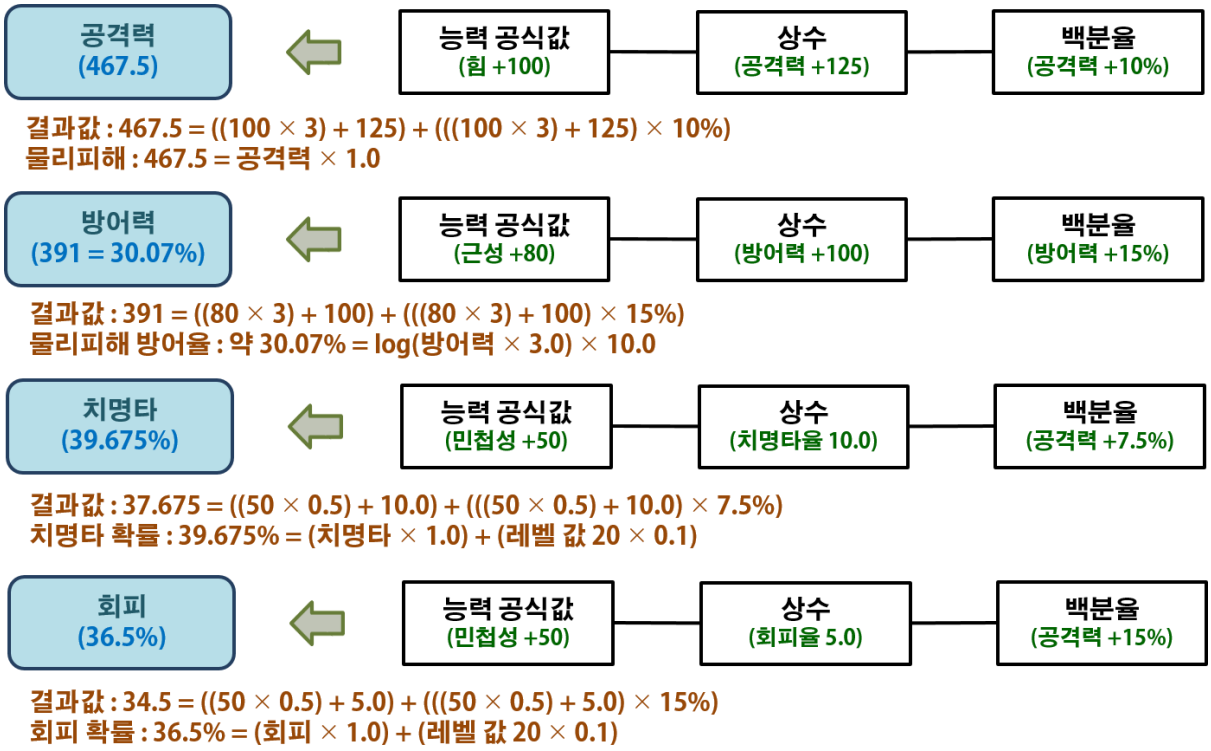
※ 어째서 게임 상태값을 그대로 실제 적용하는 전투 수치로 사용하지 않는지 의문을 가질 수 있다.
다른 게임은 몰라도, RPG의 경우, 게임 내부에서 사용하는 전투 수치 공식의 결과가, 플레이어가 GUI에서 확인할 수 있는 수치와 반드시 같지 않을 수 있다.

예를 들어, 어느 한 전사 캐릭터의 방어력이 500이라고 가정하자.
방어력은 500이라는 수치로 나와 있지만, 이는 플레이어가 확인할 수 있는 '이 캐릭터가 방어할 수 있는 정도'를 나타내는 어떤 관념적인 숫자이다.
실제 전투에서 저 방어력 500은 피해를 정말로 500만큼 줄여줄 수도 있지만, 전투 기획에 따라서 다른 공식을 적용할 수도 있다. 어떤 일련의 공식에 따라, 방어도가 500이면 약 61%의 피해를 감소하게 만드는 것도 가능하다는 말이다. 이는 즉, **게임 상태값이 반드시 전투 판정에 적용할 수치 그 자체는 아닐 수 있다**는 점을 알려준다.

또 다른 경우로, 치명타나 회피 등은 게임 상태값을 그 자체로, 혹은 변형된 별도 공식에 의거해서 확률 값으로 사용할 수도 있다.
예를 들어, 민첩성 능력치 100에, 아이템에 의해 민첩성 5%가 증가했다고 치자. 그러면 민첩성 능력치는 총 105가 된다. 민첩성이 치명타 확률을 0.1% 높여준다고 가정하면, 이 캐릭터는 10.5%의 치명타 확률이 더 높아지는 셈이다.
위와 같은 예에서 혼동할 수 있는 게, '민첩성 5%' 부분과 치명타 확률 10.5%의 관계이다. 이 경우는 능력을 표현하는 값 그 자체가 쓰는 단위가 백분율인 경우다. 이는 능력 값을 구성하고 있는 세 부분(각각 능력치, 상수, 백분율)에서의 백분율 값과는 같은 관계가 아니다.

- **B-16-2-2.** 계산 결과로 나온 게임 상태값을 가지고 또 다른 별도의 공식을 적용하여 판정에 이용할 수 있다.

※ 이런 현상 역시, 사용해야 하는 '단위'의 차이로 인해 발생한다. 그리고 꼭 단위의 차이가 없더라도, 이 원칙 자체는 어떤 경우에든 적용 가능하다.
계산 결과의 게임 상태값이 그대로 판정에 이용하는 값과 같다면, 판정에 이용하는 값은 게임 상태값에 × 1 의 공식을 적용한 것과 다름 없기 때문이다.



<게임 상태값을 실제로 적용하는 방식은 기획 의도에 따라 다양하다.>

- **B-16-2-3.** 게임 상태 값의 최소 / 최대 범위가 필요한 경우는 아이템, 스킬의 능력치를 부여할 때 뿐이다.
  : 그렇게 해서 **생성한 아이템과 스킬의 능력은 최소 / 최대 값 없이 고정된 능력을 갖는다.**

- **B-16-2-4.** 그 외 모든 판정에 최소 / 최대값 범위가 없이, 고정된 게임 상태값을 사용한다.

◆ B-16-3. 확률 처리

- **B-16-3-1.** 모든 확률 관련 값들은 확률 판정을 할 때 다음의 규칙을 따라야 한다.

1. 확률 값보다 작은 값 = 판정 성공
2. 확률 값과 같은 값 = 판정 실패
3. 확률 값보다 큰 값 = 판정 실패

※ 확률 65%의 성공 여부를 판정한다.
의사코드로 봐서, 대략 random(0, 100)의 결과를 가지고 판정할 것이다. 이 함수의 결과는 0 ~ 99 사이의 무작위 값을 돌려준다. (정확히 100 개다.)
0 ~ 64 는 정확히 숫자가 65 개이고, 그 외 나머지는 35 개이다. 따라서 판정할 때 0 ~ 64 까지는 성공으로 판정하고, 그 외 경우는 실패로 판정한다.

- **B-16-3-2.** 확률 판정은 기본적으로 정수형을 사용해야 한다.

- **B-16-3-3.** 확률 판정을 통해 실수(real number)를 가져오고자 하는 경우, 아래의 과정을 따라야 한다.

1. 원하는 실수 범위 값의 최소 / 최대 값에 1,000을 곱해서 정수로 변환한다.
2. 1000을 곱해서 변환한 최소 / 최대 정수값들의 확률 결과를 구한다.
 : 확률 결과도 정수형으로 반환될 것이다.
3. 구해진 정수형 확률 결과값을 실수형으로 변환하고, 1,000으로 나눈다. (혹은 0.001을 곱한다)

- **B-16-3-4.** 실수를 확률 판정에 사용하는 경우, (실수 × 1,000)의 값이 4바이트 혹은 8바이트 부호 없는 정수형의 한계 값 범위를 넘지 않아야 한다.
 : 값 범위를 생각하지 않으면, 바이트가 잘라 먹혀서 전혀 다른 결과가 나올 수 있다.

※ 4 바이트 부호 없는 정수형 범위 : 0 ~ 약 42 억
 8 바이트 부호 없는 정수형 범위 : 0 ~ 약 1844 경

- **B-16-3-5.** 실수를 정수로 치환할 때, 필요한 정밀도에 따라, 1,000을 초과하는 수(10,000 등)를 곱해도 무방하다.
 : 정밀도를 더 향상하는 대신, 그만큼 허용 가능한 숫자 크기 범위는 줄어든다.

※ 정밀도를 소수점 세자리 외 다른 방식을 중구난방으로 적용하는 건 추천하지 않는다.
 (10 ^ 필요한 자릿수)를 곱해서 정수형으로 변환할 때, 이 자릿수 값이 중구난방이면, 원래 실수였던 상태의 값이 무엇이었는지 알기 어렵기 때문이다. 가급적 표준으로 정한 소수점 세자리를 지키는 게 좋다.

B-17. 상태효과와 부여효과

◆ B-17-1. 정의와 차이점

- **B-17-1-1.** **상태효과(State Effect)**
: 게임 월드에 등장하는 객체에게 어떠한 능력을 향상시키거나 혹은 감소시키는 효능을 주는 객체를 말한다.

※ 여기서 '어떤 객체'라는 말은 캐릭터든, 아이템이든 모두 동일한 성격의 부여효과가 붙을 수 있다는 뜻이다.
예를 들면, [생명력 +20] 상태효과는 캐릭터에게 직접 부여될 수도 있고, 아이템에 영구 향상 능력으로써 붙을 수도 있다.
대표적인 부여효과의 예로써는, 공격력 +10, 생명력 회복 +200 등과 같이 RPG에서 '스킬 능력'이라는 명목 하에 자주 쓰이는 내용들이 있다.

- **B-17-1-2.** **부여효과(Enchanted Effect)**
: 한 개 이상의 상태효과들로 구성되며, 그렇게 묶인 상태효과들을 실제 게임 플레이에 적용할 때, 어떻게 적용할 것인지에 대한 구체적인 제한조건들을 정의하는 객체이다.

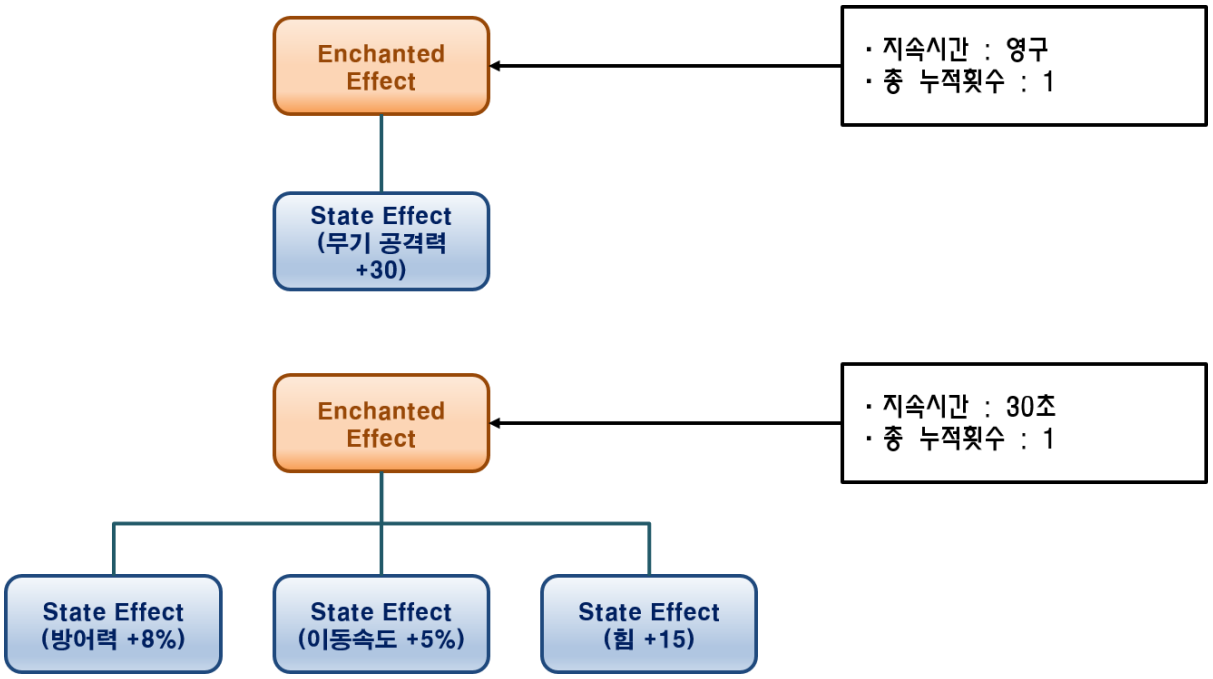
- **B-17-1-3.** **상태효과는 언제나 부여효과 객체를 통해서만 적용**한다.
: 즉, 하나의 상태효과를 적용하려면, 그 상태효과 객체를 직접 적용하는 게 아니라, 부여효과 객체가 소유하도록 만든 상태에서, 그 부여효과 객체를 적용한다는 말이다.

- **B-17-1-4.** 상태효과 하나에는 한 종류의 능력치 변화(향상 혹은 감소) 기능, 혹은 능력치가 아닌 다른 조건에 대한 변화 능력을 포함한다.
: **상태효과 하나가 여러 개의 능력치를 변화하게 할 수 없다.**

※ 여러 종류의 효과를 한 번에 주도록 만드는 단위는 상태효과가 아니라 부여효과다. 상태효과는 그러한 부여효과의 개별적인 각각의 효과 한 개를 나타낸다.
상태효과는 정말 가장 기본이 되는 구현의 단위 정도 역할만 하기 때문에, 사실 별로 주절주절 쓸만한 게 없다.

- **B-17-1-5.** 상태효과는 부여효과를 여러 개 소유할 수 있으므로, 상태효과 하나가 여러 개의 능력치 혹은 게임 플레이에서의 액션 상태 변화를 이끌어낼 수 있다.

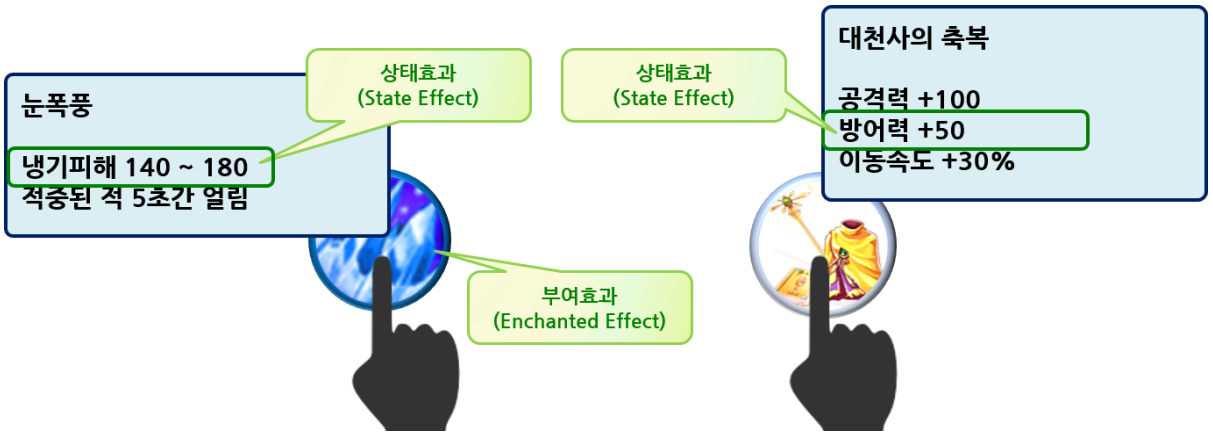
※ 뭔가 위엄이 쩌는 ['대군주의 분노']라는 이름의 부여효과를 받았다고 치자.
그리고 이 부여효과는 [공격력 +100] + [공격속도 20% 향상] + [방어력 +50 향상] 능력이 주어진다고 가정하자.
그러면 ['대군주의 분노']라는 부여효과 1 개는 3 개의 능력치 변화 객체로 구성되어 있는 것이다.



<부여효과와 상태효과의 관계 모델>

- **B-17-1-6.** GUI를 통해 플레이어에게 직접 노출하는 효과 단위는 부여효과이다. 상태효과는 GUI 등을 통해 플레이어에게 직접 노출하지 않고, 내부 처리에서만 사용하는 효과 단위이다.

※ 소위 '버프 / 디버프'로 부르는 건 상태효과가 아니라, 부여효과 단위를 말한다.
물론, '버프 / 디버프'가 걸렸을 때의 아이콘 또한 부여효과 단위로 하나씩 생긴다.



<좀 더 실감나는(?) 관계 모델>

◆ B-17-2. 상태효과와 부여효과의 적용 조건

- **B-17-2-1.** 상태효과와 부여효과는 오직 캐릭터 개체(Entity)들에게만 적용할 수 있다.
  : 자신에게 부여된 상태효과 / 부여효과들을 관리하는 컴포넌트 역시 캐릭터 개체들만 소유하고 있다.

※ 이러한 제약을 정하지 않는다면, 캐릭터 개체가 아닌 다른 게임 객체들에게 상태효과나 부여효과를 소유하게 하려고 할 수가 있다. 심지어, 부여효과가 또 다른 부여효과를 소유하려고 하는 등의 설계까지 나올 수 있는데, 이런 건 설계상으로 원하는 바가 아니다.

- **B-17-2-2.** 부여효과를 적용하는 조건

조건(단위)	설명
시작시간(초)	부여효과가 시작되는 시간
적용 횟수(상수)	부여효과를 몇 번 적용할지 여부 : '독 피해를 5회에 걸쳐 적용한다'와 같은 경우...
적용 속도(1초 당 상수)	부여효과를 초당 몇 번 적용하는지에 대한 속도 : '초당 15.5씩 체력을 회복한다'와 같은 경우...
회 당 유지시간(초)	부여효과가 X회에 걸쳐 영향을 주어야 할 때, 각 회차에서 다음 회차의 효과 부여까지 걸리는 시간
총 유지시간(초)	부여효과의 총 유지 시간. 총 유지시간에 도달하면 다른 조건들이 아직 살아 있어도 부여효과는 종료한다.
최대 중첩 횟수(상수)	부여효과를 중첩할 수 있는 최대 개수
발동 조건(구현에 따라 다름)	부여효과가 발동할 조건. : 간단한 패턴으로 정의하기 어려울 수 있어서, 사전에 정의한 식별 값이 아닌, 관련 코드 그 자체일 수 있다.
제거 조건(구현에 따라 다름)	부여효과가 제거될 조건 : 간단한 패턴으로 정의하기 어려울 수 있어서, 사전에 정의한 식별 값이 아닌, 관련 코드 그 자체일 수 있다.
타입(열거형)	부여효과가 적용되는 대상에게 유리한 효과인지 불리한 효과인지 정의한다.

- **B-17-2-3.** 지속적으로 갱신하는 성질의 부여효과인 경우, 갱신이 가능한 최소 주기는 **초당 15FPS(약 0.0667초)**이다.

※ 즉, 지속적인 회복 스킬이나, 지속적인 독 피해를 주는 스킬 같은 것들은 아무리 점검 주기를 빠르게 설정한다고 해도, 초당 15 번까지만 효과가 들어간다는 말이다.

- **B-17-2-4.** 부여효과를 적용할 때, **업데이트 1회당 1번만 적용할 수 있다.**
: 같은 업데이트 안에서 여러 번 부여효과를 적용하지 않는다.

※ 한 번의 업데이트에서 부여효과를 여러 번 적용해야 할 이유가 전혀 없다. 그럴 거면 그 여러 번의 적용 효과의 합한 수치를 1 회 적용해야 마땅하다.
만약 의도가 '연속 타격'과 같은 의도였다면 당연히 업데이트 시간차를 조금이라도 두고, X 회 업데이트하는 방식으로 부여효과를 적용해야 한다.

- **B-17-2-5.** **부여효과를 구성하는 상태효과들은 부여효과의 조건들을 모두 똑같이 적용**한다.
: 소속된 상태효과마다 다른 조건들을 줄 수 없다. 아니, 그 전에 상태효과 객체에는 제한 시간 등의 조건을 지정하는 부분이 전혀 없다.

※ 상태효과는 효능을 어떻게 적용할지에 대한 구현만 담당할 뿐이며, 제한 조건을 점검하거나 효과에 대한 아이콘이 무엇인지 결정하는 등의 책임이 전혀 없다.
따라서 모든 상태효과 객체 인스턴스들의 수명은, 그들을 소유한 부여효과에서 적용한 제한 조건에 달려있다.

◆ B-17-3. 부여효과의 중첩과 갱신 처리

- **B-17-3-1.** 원칙적으로는, 같은 종류의 부여효과가 겹친 경우에 대한 처리 역시, 부여효과의 종류마다 개별적으로 구현한다.

※ 이와 관련한, 코드를 설계할 때도, 부여효과가 적용되거나 취소되는 부분에 대해 부여효과별로 조건을 설정할 수 있도록 설계를 유연하게 해야 한다.

- **B-17-3-2.** 그러나 부여효과가 처리되는 과정은 대부분 같은 과정을 거쳐야 하는 경우가 많다.
그렇기 때문에, 특수한 경우를 제외한 일반적인 부여효과의 중첩과 갱신에 대한 처리를 정의해야 할 필요가 있다.
이는 또한, **부여효과를 처리할 때의 일반적인 원칙을 정의**하는 과정이기도 하다.

- **B-17-3-3.** 같은 능력치를 상향 / 하향시키는 부여효과를 중복해서 적용하는 경우, **무조건 나중에 걸린 부여효과가 살아남는다.**

- **B-17-3-4.** 부여효과의 지속시간은 부여효과의 중첩 횟수 제한에 따르며, 그 이상으로 늘어나지 않는다. 또한, 이 사항은 해당 부여효과가 중첩되면 유지시간이 늘어나도록 정의한 경우에만 해당한다.

※ 게임 '리니지'에서 예전에 이 현상을 악용하여 활발히 성행했던 '헤이스트 샵'을 생각해보면 된다. (이동속도와 공격속도를 대폭 늘려주는 '헤이스트' 마법을 수십 명의 마법사 캐릭터들이 일시에 사용하여, 수십 번 헤이스트를 중첩시켰다. 사실상 플레이어들이 돈만 내면 무한히 가속된 상태로 플레이 할 수 있게 했다.)
물론, 이 게임은 MMORPG 가 아니지만 말이다. (...)

- **B-17-3-5.** 중첩이 허용되는 부여효과인데, 중첩되는 여러 개의 부여효과의 레벨이 서로 다른 경우에는, **가장 강한 부여효과의 능력을 따른다.**

※ 예를 들어, 5 단계까지 중첩이 되는 [출혈]이라는 스킬이 있다고 치자.
그런데 2 개의 캐릭터가 서로 [출혈 1 레벨]과 [출혈 3 레벨]이라고 가정한다. 이들이 동시에 하나의 적을 대상으로 [출혈] 스킬을 사용한다면 중첩을 어떻게 처리해야 할까?
처리 방식에 하나의 답만 존재하지는 않을 것이다. 어떤 면에 초점을 두느냐에 따라 해법은 다양하게 존재할 수 있다.

이 게임에서는 이러한 경우, **무조건 높은 단계의 스킬에 의한 효과를 우선**하기로 한다.
이렇게 하는 이유는, **일반적으로 플레이어에게 더 유리한 방향이 되는 방식**으로 해결하기 위해서다. 약간의 부작용이라면, 플레이어에게 해로운 부여효과의 경우에도 마찬가지 방식이라는 점이다. 그러나 서로 다른 레벨의 부여효과를 플레이어가 맞아야 할 경우가 P vs P 빼고는 별로 없기 때문에(이 때는 둘 다 플레이어 캐릭터이므로, 플레이어 중 누군가는 이득을, 누군가는 손해를 본다.) 이 부분은 무시한다.

먼저 3 레벨 [출혈]이 먼저 부여된 상태에서 더 나중에 1 레벨 [출혈] 효과로 중첩하려고 하였다면, 1 레벨 [출혈] 부여효과가 더 나중에 부여되었음에도 불구하고 중첩은 3 레벨의 [출혈] 효과로 중첩이 된다. 중첩 횟수와 재사용 대기 시간 역시 3 레벨 [출혈] 효과의 것으로 계산한다.

◆ B-17-4. 유효조건 처리

- **B-17-4-1.** 유효한 장면 프로세스의 범위를 제한할 수 있다.
: 대부분의 경우, 부여효과 및 그에 속한 상태효과들은 게임 플레이 장면을 벗어나면 없어진다.

- **B-17-4-2.** 부여효과의 효과가 나타나는 제한 시간이 게임 플레이 시간인지, 실제 시간인지 구분할 수 있어야 한다.

※ 운영 정책에 의해, 실제 시간으로 동작하는 부여효과를 사용자들의 캐릭터에게 부여해주는 경우는 게임 서비스에서 생각보다 흔하게 나타난다.
(3일간 골드 2배! 경험치 2배! 이런 거...)

그런 '이벤트'를 할 때마다 서버에서 특수한 구현 코드를 집어넣고, 이벤트 종료 후 빼기를 반복하는 것보다는 이런 조건들을 사전에 코드 수정 없이(아니면 덜 하면서) 할 수 있는 구조를 미리 만들어 놓는다면, 서비스 담당자도 덜 피곤하고, 개발자도 덜 피곤하고 모두에게 평화를 가져다 줄 수 있다.
(서버를 재실행하지 않고 이벤트의 시작 / 종료를 할 수 있다면 금상첨화다!)

※ 각 부여효과를 표현하는 객체들은 지속 시간에 대한 타입을 지정해줄 수 있어야 한다.
게임 플레이 시간으로만 동작하는 게임 플레이 장면에 들어갔을 때만 부여효과의 지속 시간이 갱신되며, 게임 플레이 장면을 벗어나면 초기화되거나(유효한 장면 영역이 게임 플레이 장면 뿐일 때), 지속 시간 갱신이 중단된다. (유효한 장면이 게임 장면 전체일 때)
반면, 실제 시간으로 갱신하는 부여효과의 경우, 플레이어가 접속하지 않더라도 계속해서 시간을 소모한다.
특히 이러한 기능은 서버 단계에서 구현하여야 한다. 캐릭터의 정보를 저장하는 DB 에는 이러한 '실제 시간을 사용하는 부여효과들의 목록'을 저장하는 부분이 필요하다.

B-18. 스킬

◆ B-18-1. 정의 및 구성요소

- **B-18-1-1.** 플레이어의 직접 / 혹은 간접적인 명령을 통해서, 게임 플레이에 어떤 영향을 주는 모든 행위를 **스킬**이라고 통칭한다.

- **B-18-1-2.** 아래와 같은 경우는 다 '스킬을 사용한다'는 범주 안에 들어가는 예시이다.

1. 일반 공격 = 일반 공격 스킬 사용
2. 마법 공격 = 마법 공격 스킬 사용
3. 버프 / 디버프 부여
4. 소환

그 외에도 다양한 스킬이 있을 수 있다.

- **B-18-1-3.** 일반 공격은 기능적으로는 스킬의 일종이지만, 아래와 같은 점에서 다른 스킬과는 다르게 특수 취급한다.

1. 일반 공격에만 적용되는 치명타율 능력치가 존재함
2. 일반 공격인 경우에만 범용적으로 회피에 대한 능력치가 작용한다. (다른 스킬들은 특별히 정의하지 않는 한, 회피 개념이 없다.)
3. 공격 속도에 대한 능력치는 일반 공격에만 적용한다. 다른 스킬들은 공격 속도 능력치가 적용되지 않는다.

- **B-18-1-4.** 스킬의 각 능력들은 한 개 이상의 부여효과들로 구성된다.
: 스킬을 사용했을 때 게임 플레이에 영향을 미치는 능력은 스킬 객체 그 자체에 있지 않고, 그 스킬이 발동하도록 사전에 정의되어 있는 부여효과들로부터 발생한다.

- **B-18-1-5.** **하나의 스킬은 총 3개까지의 부여효과로 구성할 수 있다.**
: 1개의 부여효과는 그 스킬의 주 능력이며, 나머지 2개의 부여효과는 룬(Rune)에 의해 추가되는 부여효과다.

※ 부여효과가 여러 개의 상태효과로 구성되어 있으면, 실질적인 능력 단위 개수인 상태효과의 개수 단위로 따진다면, 상태효과의 개수가 3개가 넘을 수는 있다.

(설마 부여효과가 여러 개의 상태효과로 구성할 수 있다는 사실을 잊은 건 아니겠지?)

◆ B-18-2. 일반 공격 스킬

- **B-18-2-1.** 적대 대상을 공격할 때, 기본적으로 항상 사용하는 스킬은 **[일반 공격]**으로 특별하게 취급한다.

- **B-18-2-2.** 일반 공격에 대한 특별한 부여효과가 걸려있지 않은 한, 일반 공격 스킬은 아무런 비용 소모 없이 사용할 수 있다.

※ 상식적인 이야기겠지만, 대부분의 RPG에서는 특별한 스킬을 사용하기 위해서는 그에 상응하는 비용을 치러야 한다. (대개는 '마나'를 소모시키는 방식이다.)
일반 공격은 플레이어 캐릭터가 스킬 사용 비용을 전혀 치르지 못하는 상태에서도 사용이 가능한 스킬이다.

- **B-18-2-3.** 플레이어가 적 캐릭터를 직접 선택하면, 내 캐릭터는 일반 공격 스킬을 사용한다고 간주하고 목표 대상에게 다가가 일반 공격 스킬을 사용한다.

※ 이 게임에서는 언제나 적대적인 대상을 선택한 경우, **'일반 공격 스킬을 사용해서 공격하러 간다'**고 가정하면 된다.

◆ B-18-3. 선택적 사용 스킬

- **B-18-3-1.** 플레이어의 선택적 사용 스킬은, 사용자가 화면 오른쪽 하단의 스킬 버튼을 직접 터치해서 스킬을 발동할 것을 명령해야 한다.

- **B-18-3-2.** 이 게임에서의 선택적 사용 스킬들은 발동하기 위한 **시전 과정(Casting Phase)이 필요 없는 스킬만으로 구성**한다.
: 스킬의 사용을 명령하였고, 그 순간에 스킬 사용 조건을 충족했다면 스킬이 즉시 발동된다.

※ 빠르고 화끈한 액션을 강조하는 RPG를 표방하기 때문에, 좀 더 느릿해 보이고 전술적으로 느껴지는 시전 액션이 들어가는 스킬을 디자인하지 않기로 했다.
다만, 그렇다고 해서 이 기능을 전투 시스템의 설계 단계에서부터 아예 지원하지 않는다는 뜻은 아니다!

<이렇게 주문 외우는 시간이 들어가는 스킬들은 안 만든다.>

- **B-18-3-3.** 인공지능 캐릭터들은 플레이어처럼 스킬 버튼을 실제로 '선택해서' 사용할 수는 없겠지만, 조건에 의해 플레이어의 스킬 사용과 유사하게 흉내 내는 기능을 가진다.
  : 스킬을 사용할 수 있는 일정한 조건이 되면, 인공지능 코드에 의해 자신들이 보유한 일반 공격 외의 다른 특수한 스킬들을 사용할 수 있다. (더 강력한 공격, 회복 등)

- **B-18-3-4.** **결정성 스킬(Active Type Skill)**과 **항상성 스킬(Passive Type Skill)**
  : 모든 선택적인 사용 스킬들은 결정 방식의 스킬이다.
  항상성 스킬이란, 사실상 특정한 하나의 부여효과를 말한다.

※ 좀 더 상세한 내용은 부여효과에 대한 명세 내용을 참고할 것.
스킬이 결정성이냐 항상성이냐는 사실 기획적인 관점에서의 문제이다. 프로그램 설계의 관점에서는 두 가지는 내부적으로 다른 층위에 속하는 객체로 볼 수 있다.
이는 마치 아이템과 장착 부위, 장착 부위와 파츠 모델의 관계에 비유할 수 있다. 모두 다 기획적인 관점에서는 같은 분류로 묶이는 개념이지만, 코드 내부 설계에서 처리할 때는 별개의 층위와 별개의 개념으로 다룬다.

◆ B-18-4. 적용 특성

- **B-18-4-1.** 캐릭터 개체의 스킬이 대상에게 어떤 형태로 적용되는지에 대한 내용을 분류한다.

※ RPG 에서 전투가 발생했을 때, 공격 스킬이 피해를 주는 방식은 어떤 스킬을 사용했는지에 따라 다양하다
다만, 여기서의 분류는 스킬의 '시각적 / 청각적 연출'에 대한 부분이 아니다.
내부 전투 시스템이 피해를 어떤 형태로 판정하고 처리할지에 대한 분류를 나타낸다.

예를 들자면, 광범위하게 폭탄이 터지는 시각 효과를 보여주더라도, 그 공격 자체가 단일 대상에게만 피해를 주도록 설정되어 있다면, 아무리 집채만한 폭발 효과가 나더라도 실제로 피해 수치가 뜨는 것은 1 개체 뿐인 것이다.

- B-18-4-2. 적용 특성의 명칭 및 세부 내용은 아래와 같다.

명칭	설명
단일대상 (Single Target)	• 지정된 1 개체만 적용하는 방식 • 단, 여러 대상에게 스킬을 적용하는 경우라 할지라도, 각 적용부위가 1 개체에게만 스킬을 적용하는 방식이면, 그것도 단일대상 적용이다. Ex) Diablo II의 Amazon 캐릭터의 Multiple Shot 스킬  <투사체 1 개가 1 대상에게 적용되면 단일대상 적용 스킬이다>
방사형 (Splash)	• 스킬이 작용하는 중심점으로부터 일정한 반경 이내에 존재하는 모든 개체들이 적용된다. • 적용 중심점으로 멀어질수록 적용되는 정도를 조절할 수도 있다. Ex) Starcraft 의 Siege Tank 의 Siege Mode 공격 • 적용 영역 내의 대상 개체들에게 모두 동일한 효과로 적용할 수도 있다. Ex) Starcraft 의 Reaver 의 Scarab 공격  <방사형 스킬의 예>
관통형	• 특정 방향에 있는 모든 대상에게 적용한다.

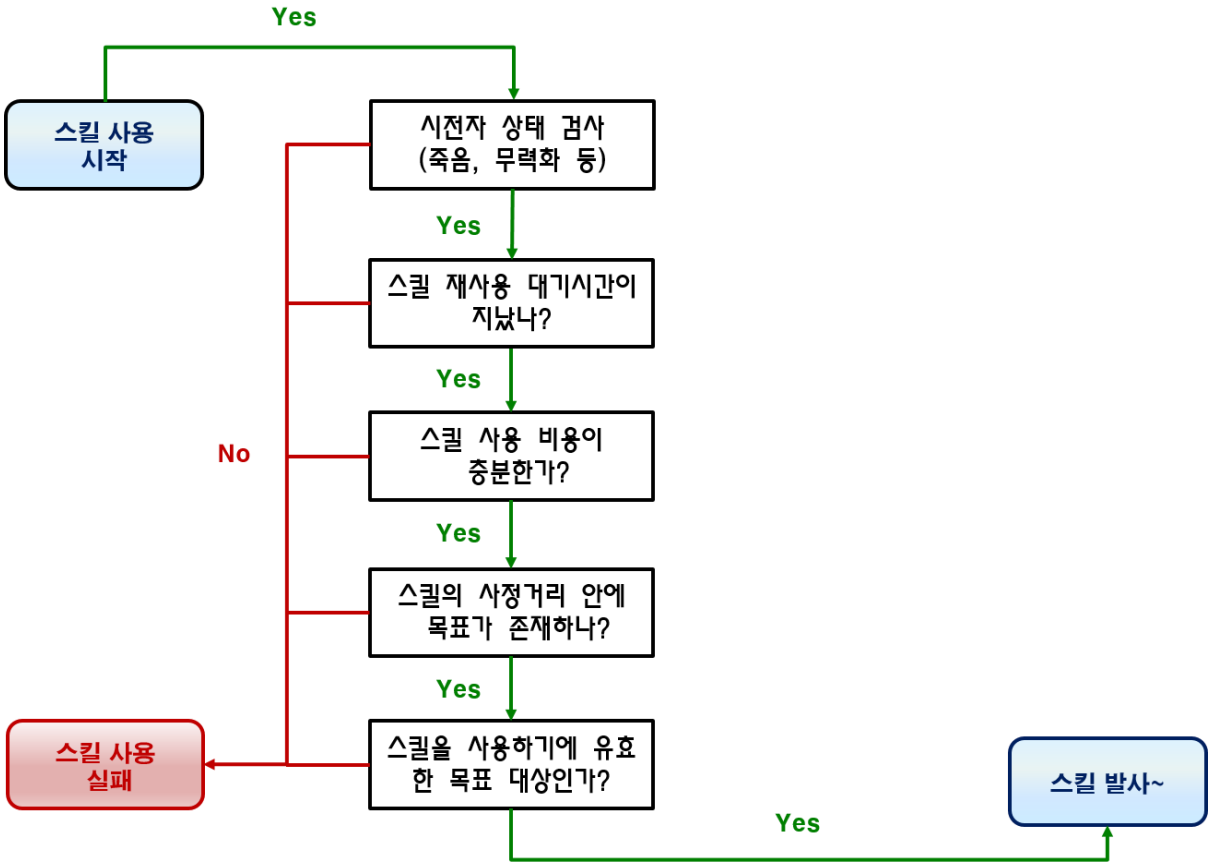
(Piercing)	• 스킬의 관통형 투사체에 접촉해야만 효과가 적용된다.  <관통형 스킬의 예>
소통형 (Channeling)	• 대상과 연결되어 있는 동안 효과를 적용한다. • 연결 조건이 성립하지 않으면, 효과 적용이 끊어지거나 취소된다. : 연결 가능 거리를 벗어난다든가, 시전자가 불능 상태에 빠졌다든가 등등...  <연결형 스킬의 예>
전달형 (Transfer)	• 적용할 대상을 순차적으로 건너 뛰어다니는 방식으로 적용한다. • 일단 전달이 성공하면, 시전자의 현재 상태가 효과 적용에 어떤 영향도 주지 않는다. ※ 구현하기에 따라,. 소통형 스킬의 변형된 형태로 분류할 수도 있다.  <전달형 스킬의 예>

※ 전투 시스템을 설명하는 부분에서 이 부분의 내용과 연결 지을 수 있는 항목들이 존재한다. 그 부분도 같이 읽어보기 바란다.

- **B-18-4-3.** 스킬들은 치명타(Critical)를 적용할지 여부를 결정할 수 있다.
: 필요한 스킬들은 치명타의 적용을 받도록 구성할 수 있다. 치명타가 없어야 하는 스킬들은 적용하지 않으면 치명타 판정 자체를 건너뛴다.

※ 스킬들 중 많은 부분을 차지하는 '적을 공격하는' 스킬들은 치명타 여부를 판정해야 하겠지만, 아군에게 도움을 주는 스킬이라거나, 직접 공격 외의 간접적인 공격 수단, 소환 같은 경우에는 치명타 능력이 의미가 없다.

◆ B-18-5. 스킬 사용 프로세스



<스킬을 사용할 때의 조건 검사 흐름>

- **B-18-5-1.** **시전자의 조건 검사**
: 가장 먼저 스킬을 시전할 캐릭터가 현재 스킬을 사용할 수 있는 상태인지 조사해야 한다.
캐릭터가 죽어 있는 상태라거나, 아무런 행동도 할 수 없는 무력화된 상태라면, 볼 것도 없이 스킬 사용 시도는 실패해야 한다.

- **B-18-5-2.** **스킬의 재사용 대기시간이 지났는지 검사**
: 모든 스킬들은 재사용 대기 시간이 존재한다. 이 대기 시간이 지나기 전에는 같은 스킬을 다

시 사용할 수 없다.

※ 재사용 대기 시간이 없는 스킬들은, 구현 단계에서 볼 때는 재사용 대기 시간이 0으로 설정되어 있는 스킬로 취급한다.

- **B-18-5-3. 스킬을 사용하는데 대한 비용을 지불할 수 있는지 여부 검사**
: 스킬들의 능력이 다양하기 때문에, 보통 스킬의 강력한 정도에 따라 스킬을 사용하는 데 따른 대가를 지불하도록 디자인을 한다. 그 형식과 값은 기획 내용을 따른다.
스킬 사용에 대한 비용을 지불할 수 없으면, 스킬을 사용할 수 없다.

※ 보통 마력(Magical Power)이나 마나(Mana)를 소모하는 방식으로 표현하지만, 스킬에 따라서 별도의 또 다른 재료를 소모해야 할 수도 있다.
데이터 모델을 설계할 때, 기획 파트에서 이러한 스킬 사용에 대한 소모 비용을 디자인하면서 원하는 종류의 소모 아이템(재화)타입과 소모할 양의 값을 지정할 수 있도록, 테이블 데이터를 설계해야 한다. 그저 무작정 '마나 XXX 소모' 이런 식으로만 비용을 잡을 거라고 가정하면 안 된다.

- **B-18-5-4. 스킬의 유효 사정거리에 목표 대상이 있는지 검사**
: 게임 플레이에서 사용하는 스킬들의 공통적인 제약 조건 중 하나는 스킬의 유효 사정거리이다. 이 사정거리를 통해서 스킬이 '근접형'이냐, '원거리형'이냐를 구분하기도 한다. (거리가 근접인지 원거리인지 애매한 스킬들도 있을 수 있다.)
아무튼, 대상이 스킬의 유효 사정거리 밖에 있는 경우, 스킬은 실패한다.

- **B-18-5-5. 목표 대상 유효성 검사**
: 스킬마다 목표 대상을 지정할 수 있는 방식이 다르다. 정확한 대상 타입에 대해서만 스킬을 사용해야 한다.
만일 대상 타입이 스킬이 원하는 조건이 아니라면, 스킬 사용은 실패한다.

※ 스킬의 대상 타입에 대해서는 관련 항목에서 더 상세하게 설명하겠다.

- **B-18-5-6.** 여기까지 문제없이 통과했으면, 스킬을 사용하는데 지장이 없는 상태다.
: **스킬을 사용한 것으로 처리**한다.

- **B-18-5-7. 스킬 사용 결과 처리**
: 크게 두 가지를 처리해야 하는데, 스킬의 효과 처리가 하나이고, 나머지 하나는 스킬 사용에 따른 시전자의 비용 처리이다.

- **B-18-5-8.** 스킬의 결과 처리는 스킬에 따라 비동기적인 방식으로 처리할 수 있다.
: 스킬 사용자가 스킬을 사용했다고 판정하는 시점과, 목표 대상이 스킬에 적중했을 때, 그 결

과를 판정하는 시점이 다른 스킬들은 여기에 해당한다.

※ 스킬을 사용한 시점에서 즉각 명중과 피해 판정을 하는 스킬은 결과 처리가 동기적이라고 볼 수 있다.
하지만 많은 스킬들은 **피해를 판정하는 시점이 스킬 사용 시점과 다르다.** 스킬에 의한 투사체가 날아가는 시점까지 피해 판정을 미루어야 하는 경우도 있고, 아예 스킬의 투사체에 충돌하는지 여부를 검사해서, 충돌이 일어나야지만 스킬에 대한 피해 판정에 들어가야 하는 경우도 있다.

<스킬을 사용한 시점과, 결과를 판정하고 적용하는 시점이 다른 상태인 예>

◆ B-18-6. 스킬 사용의 처리 과정

- **B-18-6-1.** 캐릭터가 스킬을 사용할 때는 반드시 다른 행동을 멈춰야 한다.
: 다른 행동을 하면서 동시에 스킬을 사용할 수는 없다.

※ 게임에 따라서는 이동하면서 스킬을 쓰는 등, 두 가지 액션을 혼합해서 사용할 수 있는 방식으로 디자인하는 경우가 있다.
그러나 이 게임은 액션 시스템에서부터 그러한 경우를 허용하지 않는다.

- **B-18-6-2.** **동시에 두 개 이상의 스킬을 한 번의 액션으로 사용할 수 없다.**

: 반드시 하나의 스킬 액션이 완전히 완료된 후에야, 다음 스킬을 사용할 수 있다. 즉, 하나의 스킬은 하나의 액션만 대응한다.

- **B-18-6-3.** 일단 스킬의 사용을 시작하였으면, 그 스킬은 일반적인 명령으로는 중간에 취소할 수 없다.
: 스킬 액션이 끝날 때까지는 그 캐릭터는 플레이어의 통제 밖에 있다.

- **B-18-6-4.** 스킬 액션을 중간에 중단하려면, 게임 내에서 무력화 스킬을 통해서 스킬 액션을 중단하게 할 수 있다.
: 구현 단계에서 표현하자면, 당한 캐릭터는 무력화 액션 상태로 강제로 전이한다고 표현할 수 있다.

※ 각 스킬들은 중간에 무력화가 가능한지를 점검하는 별도의 검사장치와 방식을 구현해야 한다. 왜냐하면 사용 중에는 어떤 방해효과도 받을 수 없게 디자인해야 할 수도 있기 때문에 그렇다.

- **B-18-6-5.** 스킬 사용에 대한 진행 과정과, 해당 스킬 사용 액션의 진행 과정은 병렬적이지만 하나의 쌍으로 진행된다.
그리고, **스킬 사용에 대한 진행 과정은 게임 플레이에서의 상태 및 수치 판정에 대한 부분에만 관여**하고, 그에 대응하는 **액션 객체는 그러한 스킬 사용의 연출만 담당**한다.

※ 이 역시, 판정과 연출을 담당하는 부분을 분리하기 위한 설계의 결과이다.
스킬을 사용했을 때, 스킬 객체는 해당 스킬의 효력을 언제 어느 때 적용할지에 대한 일종의 시간 순 이벤트(Time Line Event)를 가진다고 보면 된다. 이때의 스킬이 소유한 시간 순 이벤트는 눈에 보이거나 귀로 들리는 요소가 없는, 오직 스킬 판정에 대한 내용만을 담고 있다고 보면 된다.
그리고, 그 스킬을 사용하는 시점의 캐릭터의 액션은, 그 스킬을 사용할 때의 캐릭터 애니메이션과 FX, 사운드 등의 연출 과정에 대한 시간 순 이벤트를 담당한다.
즉, 이 때의 스킬 객체와 액션 객체들은 서로에 대해 직접적으로 소스 객체 상의 의존 관계를 갖지는 않으나, 서로가 사전에 같은 시간 맞춰 한쪽은 판정 이벤트를, 한쪽은 연출 이벤트를 담당해서 결과적으로 스킬 하나의 사용에 대한 전체 이벤트(판정, 연출)들을 완성한다.

◆ B-18-7. 스킬 투사체(Skill Projectile) 또는 스킬 효력자(Skill Affecter)

- **B-18-7-1.** 스킬 투사체는 스킬을 발동했을 때, 실제로 효력을 주기 위한 인스턴스 객체이다.

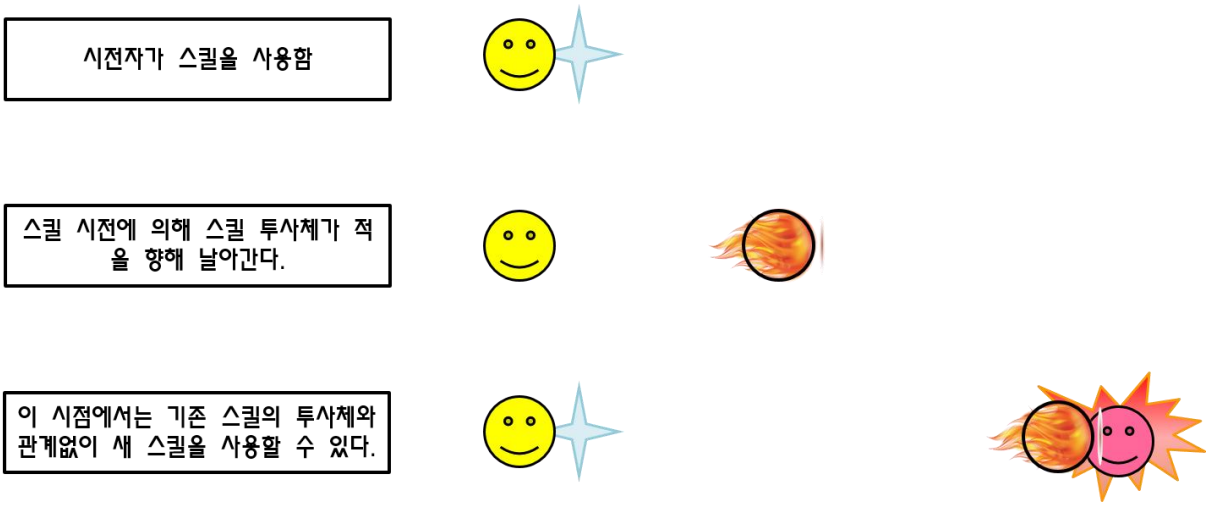
- **B-18-7-2.** 스킬의 외형이나 소리 등의 연출과는 직접적인 관계가 없는, 오직 **판정에만 관여하**

는 객체이다.
: 따라서, 스킬 투사체 그 자체는 눈에 보이는 외형을 가지고 있지 않다. 외형을 보이게 하는 것들은 스킬 투사체에 붙은 모델이나 FX 등일 뿐이다.

※ 10미터 앞의 적에게 날아가는 스킬 투사체는 그 외형이 파이어 볼처럼 생겼건, 화살 모양이건 아무런 관계가 없다. 그저, 투사체 자체가 가지고 있는 충돌 판정 방식과 사정거리, 피해 반경 등에만 의지해서 스킬의 적중 여부와 피해 여부를 적용할 뿐이다.

- **B-18-7-3.** 소통형(Channerling) 스킬이 아닌 한, 일반적으로 **스킬 투사체는 일단 생성되면 취소되지 않는다**.
: 스킬을 시전하고 있는 캐릭터를 공격하거나 부여효과를 걸어서 그 캐릭터의 스킬을 '차단'할 수는 있으나, 그러한 차단 전에 스킬 투사체가 먼저 생성되었으면, 생성된 스킬 투사체에 의한 스킬 적용 및 피해 판정 과정은 시전자의 현재 상태와 관계없이 독립적으로 진행한다.

※ 위와 같은 이유로 인하여, 스킬을 발동하는 즉시 판정이 나는 일명 '즉발 스킬'은 시전 도중에 차단하는 게 불가능하다는 점을 알 수 있다.
또한, 판정을 즉시 하지 않더라도 스킬 시전이 일어나자마자 해당 스킬의 투사체가 생성되는 경우 역시, 실질적으로 스킬을 시전 도중에 차단할 수가 없다.
스킬의 시전을 도중에 차단하는지 아닌지에 대한 기준은, 그 스킬의 투사체가 생성되는 시점에 달려 있다.



<스킬 투사체는 캐릭터의 스킬에 의해 생성된 순간, 독립적으로 작동한다.>

- **B-18-7-4.** 스킬 투사체의 소멸 시점은 **스킬의 사용 동작이 종료되는 시점과 독립적으로 각 스킬 투사체 자체에 의해 구현**하여야 한다.
: 이런 구조를 가져야 하는 이유는, 일정 시간 동안은 계속 투사체가 남아 있어서 효력을 발휘하고 있어야 한다는 방식의 스킬도 있을 수 있기 때문이다.

스킬 투사체의 소멸 시점을 정의하는 방식은 다음과 같은 종류가 있을 수 있다.

1. 투사체가 목표 대상에게 적중했을 때
2. 생성된 이후 일정한 시간이 지났을 때
3. 지정한 대상(주로 시전자)으로부터 일정한 거리 이상 멀어졌을 때
4. 지정된 공간, 혹은 대상에게 동일한 투사체가 중복될 수 없는 성격이어서, 같은 투사체가 새로 생겨났을 때

이 밖에도 기획이나 사양에 따라 더 다양한 조건으로 소멸 조건을 다룰 수 있어야 한다.

※ 조건이 워낙 다양할 수 있고, 어떤 특정한 타입 몇 개로 분류하기 어려울 것 같다.
따라서, 구현할 때는 이걸 어떤 열거형(Enumeration)이나 타입 상수로 구분하려고 하기보다는, 구체적인 스킬 객체 각각이, 자신이 생성한 스킬 투사체에 대한 해제 기능을 컨트롤 하는 인터페이스를 일정한 형식으로 구현할 수 있게 설계하는 게 더 좋을 것 같다.

- **B-18-7-5.** 스킬 투사체는 스스로 적용 범위를 가지고 있어서, 범위에 들어온 대상에게 효과를 적용할 수도 있어야 한다.

※ 소위 '장판을 까는' 광역 범위 방식의 상태 효과를 거는 스킬들이 그러한 투사체의 형태로 표현할 수 있다.
이런 점에서 미루어 보면, 스킬 투사체라는 개념은, 결국 트리거 영역과 유사한 기능을 하는 객체라고 볼 수 있다. 다만, 트리거 객체보다 작용 개념의 범위가 좁고, 무엇보다 트리거 객체는 스테이지를 나가기 전에는 객체 자체가 소멸되지 않는다는 점에서(하던 역할을 중단할 수는 있어도, 트리거 객체 그 자체는 소멸하지 않는다.) 분명히 역할과 목적이 구분된다.

◆ B-18-8. 스킬 투사체의 구조

- **B-18-8-1.** 스킬 투사체는 **스스로 위치 정보를 가지고 있고, 이를 갱신할 수 있는 형식의 객체**로 만들어야 한다.
: 어떤 판정이든, 위치에 대한 판정은 스킬 판정에 있어 기본적인 사양이기 때문이다.

※ Unity 프로젝트에서는 스킬 투사체를 UnityEngine.GameObject의 한 종류로 표현하면 된다.
실제로는 엔진 내부의 기능을 활용한 연출 관련 GameObject들이 스킬 투사체 객체의 하위 객체로써 더 많이 붙어 있는, 프리팹 덩어리의 형태로 사용한다.

- **B-18-8-2.** 스킬 투사체는 **그것을 만들어낸 본래의 스킬 객체의 데이터를 물려받는다**.
: 이를 통해, 설령 캐릭터가 다른 스킬로 전환하여, 원래의 스킬 객체가 없어지더라도, 자신을 만들어냈던 스킬의 능력과 특성을 그대로 가진 채 작동한다.

- **B-18-8-3.** 스킬 투사체는 이벤트에 의해 스스로 다른 스킬 투사체를 만들어낼 수 있다. 또한, 그렇게 생성한 스킬 투사체는 원래의 스킬 투사체와 동시에 공존할 수 있다.

- **B-18-8-4.** 스킬 투사체는 그것을 만들어낸 스킬과 스킬 사용자에 대한 정보를 전파한다.
: 스킬 투사체에 의해 생성된 스킬 투사체에게도 자신이 물려받았던 스킬 객체와 그 사용자에 대한 정보를 물려준다.

※ 이를 통해, 아주 오랫동안 게임 세계에 머물면서, 여러 개의 스킬 투사체를 만들어내는 스킬을 구현해야 하는 경우에도 대응할 수 있다. 항상 원래 자신들을 만들었던 스킬이 사용되었을 당시의 스킬 능력 정보와 사용자 정보를 보유하고 있기 때문이다.

- **B-18-8-5.** 각 캐릭터들은 자신들이 만들어 낸 스킬 투사체를 추적할 수 있어야 한다.
: 물론, 각 스킬 투사체는 자신을 만들어 낸 스킬 정보와 사용자를 알고 있으므로, 이를 통해 적절한 시점에 스킬 투사체를 조종하는 스킬 이벤트를 연출할 수 있다.

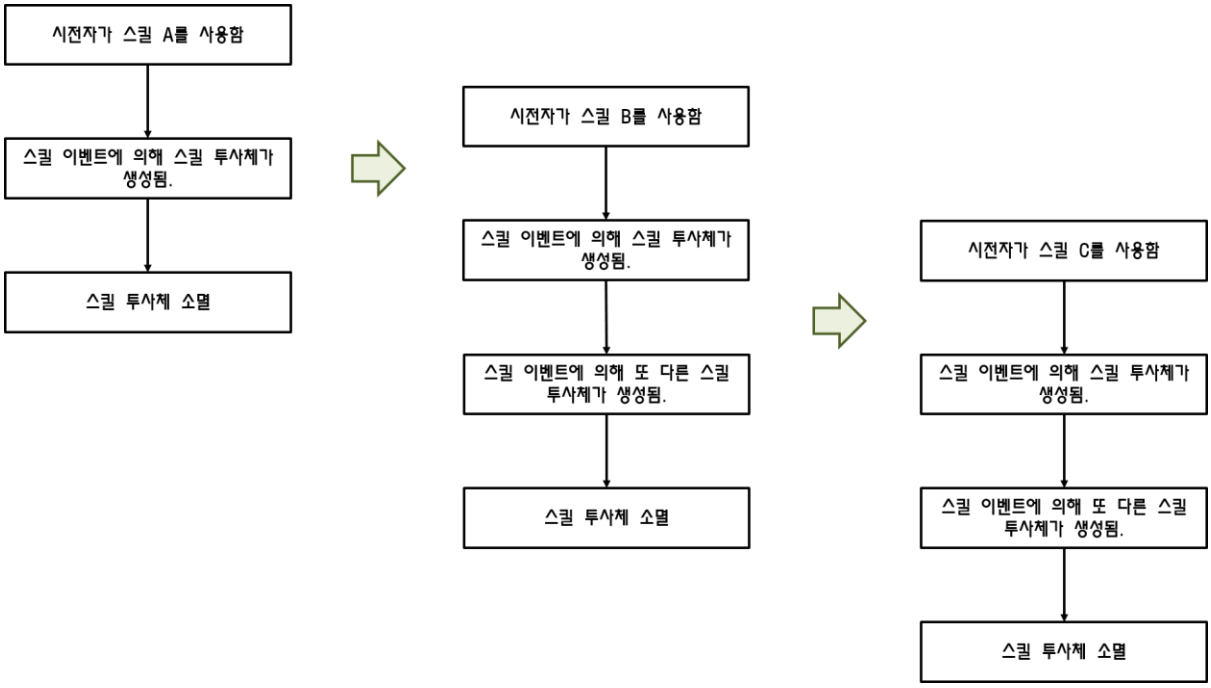
※ 이처럼 스킬 사용자 -> 스킬 투사체, 스킬 투사체 -> 스킬 사용자 추적이 둘 다 가능하다는 점은, 스킬 이벤트 연출을 만들어내는 데 있어서 상당한 장점이다.
꽤 복잡한 조건을 가진 이벤트를 구현하기 위해서는 자신의 객체를 발생시켰던, 혹은 자신이 발생시킨 객체들이 무엇인지 알아내야 하는 경우가 많다. 서로 상대방의 현재 상태라든가, 혹은 이벤트 조건을 검사하기 위한 더 많은 정보들을 얻어내는 손쉬운 통로가 되기 때문이다.

◆ B-18-9. 스킬 투사체의 처리 프로세스

- **B-18-9-1.** 스킬 투사체는 반드시 스킬이 사용되어서, 스킬의 이벤트가 시작한 이후에 등장할 수 있다.
: 스킬이 사용되지도 않았는데 그 스킬의 스킬 투사체가 먼저 등장할 수는 없다.

- **B-18-9-2.** 스킬 투사체의 종료 시점은, 스킬의 구현에 따라 캐릭터가 다른 스킬을 사용하도록 동작을 전환했는지 여부와 무관할 수 있다.
: 캐릭터는 한 번에 하나의 액티브 스킬만 사용할 수 있으나, 스킬 투사체는 캐릭터가 이미 다른 스킬을 사용했는지에 관계 없이, 자기 자신의 소멸 조건이 만족할 때까지 계속 살아 있는다.

※ 이 말은, 캐릭터가 여러 스킬을 연속해서 사용했을 경우, 각 스킬의 구현에 따라서는 여러 스킬의 투사체들을 전부 동시에 객체가 살아 있는 상태로 유지할 수도 있다는 뜻이다.
다음에 나오는 그림을 보면 알겠지만, CPU의 클럭 파이프라이닝 과정과 유사한 면을 발견할 수 있다.



<캐릭터의 스킬 사용 과정과, 각 스킬의 스킬 투사체 소멸 시점은 직접 관계가 없다.>

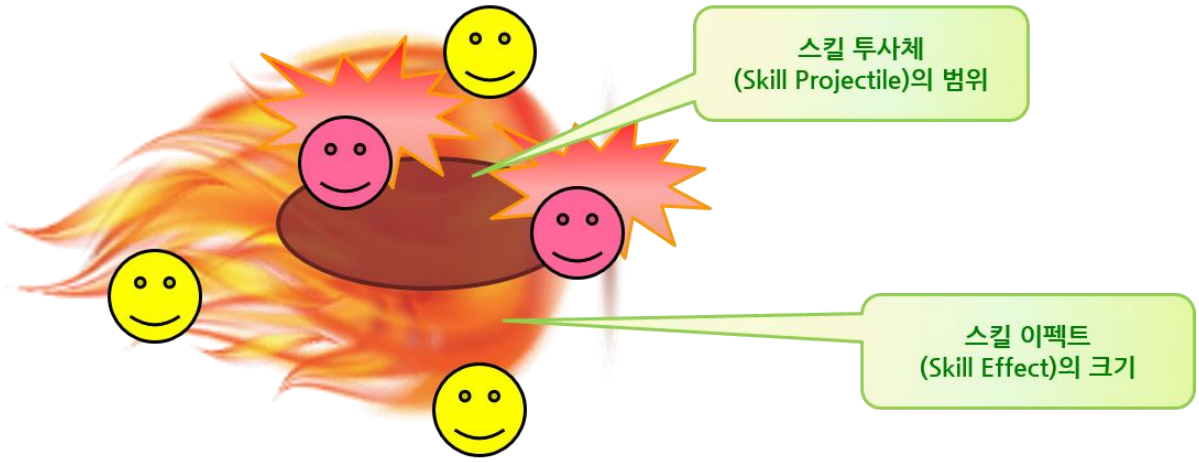
◆ B-18-10. 스킬 투사체와 스킬 이펙트

- **B-18-10-1.** 이 게임 프로젝트의 객체 코드 설계 구조로 보면, 스킬 투사체(Skill Projectile)와 스킬 이펙트(Skill Effect)는 분명히 구분되는 개념이고, 또한 동시에 게임 세계에서 존재할 수 있다.

- **B-18-10-2.** 스킬 투사체는 스킬에 대해, **판정과 관련한 기능**을 수행하는 개념의 객체다.

- **B-18-10-3.** 스킬 이펙트는 스킬에 대해, **연출과 관련한 기능**을 수행하는 개념의 객체다.
  : 단, 스킬 이펙트라는 객체 구분이 따로 존재하지는 않다. 이펙트 연출(Effect Presentation) 객체가 스킬을 연출할 경우에 이렇게 부른다고 보면 된다.

- **B-18-10-4.** **스킬의 범위 판정은 전적으로 스킬 투사체에서 판단한 범위를 근거**로 한다.
  : 스킬 이펙트는 여기에 대해 어떠한 영향도 미치지 않는다. 즉, 연출은 연출일 뿐이며, 판정 크기에는 전혀 영향을 주지 않는다는 뜻이다.



<스킬이 적중하였는지 여부는 오직 스킬 투사체에 달려 있다.>

- **B-18-10-5.** 스킬 투사체는 필요할 때, 연출과 관련한 기능(애니메이션, FX, 사운드 등)을 가진 객체들을 호출할 수 있지만, 이러한 연출 객체들이 스킬 투사체의 동작에 영향을 주어서는 안 된다.

※ 스킬 투사체는 원칙적으로 판정과 관련한 정보만 가져야 하므로 연출과 관련한 동작을 하지 않아야 한다. 하지만, 현실적으로 **연출 시점이 동적인 경우가 많다**는 문제 때문에 스킬 투사체가 연출 객체들의 동작 시점도 정의하게 되는 경우가 종종 발생한다.

그래서 **스킬 투사체를 간략한 캐릭터 객체의 일종**으로 봐도 무방하다. 캐릭터 개체가 연출 관련 정보들을 담당하는 컴포넌트 객체들을 소유하고 있는 관계로 보면, 스킬 투사체의 이벤트에 의해 연출 객체를 다루는 관계가 그다지 어색하지 않다.

이런 점은, 스킬 투사체와 스킬 이펙트 객체의 관계가 수평적이지 않음을 암시한다. 스킬 투사체가 스킬 이펙트보다 더 상위에 있는 개념의 객체이고, 스킬 이펙트는 스킬 투사체에 의해 다루어지는 컴포넌트 객체로 볼 수 있다.

◆ B-18-11. 스킬 이벤트

- **B-18-11-1.** 스킬 전 과정이 스킬의 사용 즉시 완료되지는 않는다.
: 그 때문에, **스킬 사용 명령이 시작된 직후부터, 스킬이 실제 효력을 게임 세계에서 나타낼 때까지 전 과정을 표현하고 관리**하는 객체가 필요하다. 이런 객체를 스킬 이벤트 객체로 부른다.

- **B-18-11-2.** 스킬 이벤트들은 각 스킬 별로 등록한 순서에 의해 순차적으로 수행 조건을 검사하고, 조건에 맞으면 이벤트를 실행한다.

- **B-18-11-3.** 같은 스킬을 구현하는 여러 스킬 이벤트들끼리는, 특별히 조건으로 설정되어 있지 않은 한, **어떤 스킬 이벤트를 먼저 수행했는지 여부가 다음 번 등록한 스킬 이벤트의 수행 여부를 제한하지 않는다.**
: 짧게 말하자면, 각 스킬 이벤트들은 적용 시점만 올바르다면, 등록한 순서와 관계 없이 동시에 수행할 수 있다는 뜻이다.

- **B-18-11-4.** 스킬 이벤트를 등록한 순서와, 실제로 스킬 이벤트가 동작하는 시점은 차이가 날 수 있다.
: 이런 경우는 주로, 이벤트가 동작해야 하는 시점이 사전에 정해지지 않고, 실행 시점에서 결정되는 경우에 발생한다.

※ 화염구가 점점 커지다가 적이 있는 곳까지 날아가서 팍 터지는 연출이 있는 스킬 이벤트가 있다고 치자.
여기서 [1]. 화염구가 점점 커지는 이벤트와 [2]. 적에게 충돌하면 팍 터지는 이벤트로 스킬 이벤트를 구현할 수 있음을 알 수 있다. (실제로 꼭 이렇게 구현한다는 뜻은 아니다.)
그리고 개발자는 [1] 이벤트는 스킬 시작 후 0.05초 뒤에 실행하도록 하고, [2]는 조건을 걸어서 적에게 적중한 시점에 발동하도록 하였다.
이벤트 객체의 등록은 [1], [2]의 순서대로 하였다.

위 예제에서, [1]은 스킬을 사용한 이후에, 상대적으로 고정적인 시간에 처리하지만, [2]는 그렇지 않다. [2] 이벤트의 수행 시점은 스킬을 사용하는 대상이 되는 적이 얼마나 멀리 떨어져 있는지에 따라 달렸다.
극단적으로 본다면, 적이 스킬 사용자의 바로 코앞에 있어서 둘 사이의 거리가 0이라면, [2] 이벤트는 [1] 이벤트보다도 더 빨리 수행된다. 비록 등록 순서는 [1] -> [2] 순서이지만, 이는 어디까지나 이벤트 조건을 검사하는 순서일 뿐이다. 수행 조건만 맞으면 이벤트의 수행 순사는 언제든 뒤집힐 수도 있다.
이런 점에서 스킬 이벤트의 작동 방식은, 트리거 이벤트의 작동 방식에 매우 가깝다.

◆ B-18-12. 스킬 투사체 이벤트

- **B-18-12-1.** 스킬 투사체의 이벤트는 스킬의 이벤트와 기능적으로는 크게 다르지 않다.
: 스킬 투사체의 이벤트들도 스킬 이벤트와 완전히 동일한 범위의 이벤트들을 다룰 수 있다.

- **B-18-12-2.** 스킬 투사체 이벤트는, 그 **이벤트를 소유한 스킬 투사체가 생성되었을 때만 작동할 수 있다.**
: 이게 스킬 이벤트와 스킬 투사체 이벤트가 별도로 존재해야 하는 가장 큰 이유다.

※ 스킬 투사체는 스킬의 생명주기와 완전히 동일하지 않다.

스킬 투사체의 생성 시점은 스킬 사용 이후에 조금 더 시간이 지난 이후일 수 있고, 소멸 시점은 스킬의 사용 시간, 다른 스킬로의 전환, 심지어 사용한 캐릭터가 죽었는지 여부와도 별개일 수 있다.

이 때문에 스킬 이벤트의 진행은 캐릭터가 다른 스킬을 써서 스킬을 전환하거나, 혹은 캐릭터가 사망하는 등의 상황이 발생하면 전부 취소된다. 반면, 스킬 투사체는 일단 생성되기만 하면, 자신을 생성시킨 스킬 객체와 스킬 사용자 캐릭터의 상황과 독립적으로 작동해야 할 수 있으므로, 스킬 투사체 이벤트는 스킬 이벤트와 별도로 존재해야 한다.

- **B-18-12-3.** 스킬 투사체의 이벤트는 **스킬 투사체의 인스턴스 객체 별로 유지**한다.

: 그 때문에, 스킬 투사체의 이벤트는 현재 스킬이 다른 스킬로 전환되었는지 여부와 관계 없이, 종료 조건이 만족할 때까지 이벤트가 계속 진행된다.

또한, 스킬이 몇 개의 스킬 투사체를 생성하는지, 또는 언제 생성하는지에 따라 연출의 복잡도와 방식이 달라진다.

B-19. 스킬 능력 / 강화

◆ B-19-1. 스킬의 강화 방식

- **B-19-1-1.** 각 스킬은 강화 단계가 존재하고, 강화 단계가 높을수록 스킬의 기본 능력이 강해지는 방향으로 설계한다.

- **B-19-1-2.**

- **B-19-1-3.** **공격과 관련한 스킬이 능력들은 캐릭터의 현재 능력에 영향을 받게 할 수 있어야 한다.**
  : 이런 부분들은 특히 스킬에 의한 피해량이나 치유량과 같은 부분들을 계산하는

- **B-19-1-4.** 따라서, 스킬에 의한 피해량, 치유량 등의 영향력 값을 산출하는 과정은, 반드시 **캐릭터의 현재 능력치를 계산하는 과정이 모두 끝난 이후**에 해야 한다.

※ 결국은 이런 부분들은 어쨌든 '기획에 따라 달라지는' 부분이긴 하다.
스킬 능력을 규정할 때, 캐릭터가 맨몸 상태일 때의 순수한 기본 능력만 반영하겠다고 규정했다면 그렇게 해야 한다. 다만, 스킬의 최종 능력을 계산하는 시점을 항상 캐릭터의 최종 능력치를 계산한 시점 뒤로 미룬다면, 기획이 어떤 식으로 스킬 능력을 규정하더라도 코드 베이스를 크게 수정해야 할 필요가 없다. 심지어, 스킬마다 어떤 시점의 캐릭터 능력을 적용할지를 선택적으로 활용할 수도 있게 된다.

◆ B-19-2. 스킬의 강화 체계

- **B-19-2-1.** 스킬 강화 단계의 개수는 관련 기획서의 내용을 따른다.

※ 현재의 기획은, 각각의 스킬 강화 단계는 캐릭터의 레벨 개수와 동일한 단계를 가진다는 점이다.
일반적으로 현재 중국 시장의 게임들은, 이 방식이 보통 가장 익숙하다고들 한다.

- **B-19-2-2.** 스킬의 강화 단계는 마치 캐릭터가 경험치를 가지고 레벨을 성장하는 것처럼, 스킬 강화 포인트를 이용해 성장하는 방식일 수도 있고, 아니면 곧장 강화 단계만 올리는 방식을 수

있다.

◆ B-19-3. 스킬 강화 비용

- **B-19-3-1.** 스킬의 강화 단계에 따라 비용이 다르도록 설정할 수 있어야 한다.

- **B-19-3-2.** 스킬의 강화 단계를 앞서서 습득하고 싶을 때, 이에 대한 추가 비용을 설정할 수 있어야 한다.
  : 주로, 캐릭터의 현재 레벨이 요구하는 것보다 더 많은 스킬을 배우고 싶을 때, 과금 화폐를 지불하면 그것을 허용해주는 경우가 있다.

※ 다만, 이런 기능을 허용한다면, 캐릭터가 현재 레벨에 알맞은 스킬 개수를 가지고 있는지 유효성을 검사할 때, 그 기준을 좀 완화해야 하는 단점은 있다. 이런 '돈 때려박기'로 원래 가질 수 있는 것보다 더 많은 스킬 개수를 가진 캐릭터도 분명 있다고 예상해야 하기 때문이다.
그 경우에는 별 수 없이 캐릭터가 최고 레벨 단계에서 최종적으로 가질 수 있는 스킬 개수의 범위 안에 있는지 여부 정도만 검사하는 것으로 유효성 검사를 마쳐야 할 것이다.

~~◆ B-19-4. 스킬 룬 시스템~~

~~- B-19-4-1. 스킬 룬 시스템이 기획에서 제거됨.~~

~~◆ B-19-5. 스킬 룬의 획득과 사용 과정~~

~~- B-19-5-1. 스킬 룬 시스템이 기획에서 제거되었기 때문에, 이 개념도 마찬가지로 사라졌다.~~

B-20. 아이템

◆ B-20-1. 아이템의 특질

- **B-20-1-1.** 모든 아이템들은, **캐릭터의 레벨에 따른 제한 없이 사용할 수 있다.**

※ 현재 사업적인 의도는 대략 이렇다.
스테이지에서 얻을 수 있는 아이템들은 그 스테이지가 요구하는 레벨 대역에 맞는 아이템들만 등장한다. 하지만, 무작위 뽑기(가챠, Gacha)에서는 정말 무작위로 모든 레벨 대역에서 아이템을 뽑아온다.
그런데 이 아이템들을 사용하는 데 있어 레벨 제한이 있게 되면, 기껏 뽑아 놓은 아이템들을 레벨 제한 때문에 사용하지 못하고 물품 보관함에 처박아놓아야 하는 상황이 벌어진다.
심지어, 물품 보관함조차 (대개는) 슬롯 개수를 요금을 지불하고 사야 하기 때문에 플레이어가 느끼는 불편은 더욱 심하다.
이 게임이 부분 유료화의 요금제를 고수하는 이상, **요금을 지불한 사용자가 그 혜택을 즉각 받지 못하게 되는 부작용을 최소한으로 줄여야 한다**고 판단하고 있다.

- **B-20-1-2.** 사용자는 다른 계정의 캐릭터에게 물품 보관함에 들어가는 아이템을 옮길 수 있는 수단을 가지지 못한다.

- **B-20-1-3.** 자신 외 다른 계정과의 물물교환, 거래 등도 금지한다.
: 거래할 수 있는 대상은 오직 상점뿐이다.

※ 향후 기획에 따라, 아이템을 거래할 수 있는 수단이나, 경매장 등을 둘 수도 있을지 모른다.
하지만 현재로써는 어쨌든 플레이어끼리 아이템을 주고받을 수 없는 것만으로도 아이템 복사나 거래 버그 등의 치명적인 문제점들을 많이 회피할 수 있다.
마케팅 측면에서도, 영구히 유지될 수 밖에 없는 아이템들이 아무런 요금 지불도 없이 소유권만 변경된다면 그다지 유리한 게 없을 것이다.

- **B-20-1-4.** 같은 계정 간에도, 캐릭터끼리 아이템을 옮기는 인터페이스를 별도로 제공하지 않는다.

※ 보통 이런 경우, 계정 내 캐릭터들끼리 공유하는 '창고'에 해당하는 보관함이 존재하고, 개별 캐릭터들의 물품 보관함이 따로 있는 방식으로 설정을 하는 경우가 많다. (MMORPG들이 그러하다.)

그러나 이 게임은 MMORPG도 아니고, 모바일 게임에서 아이템을 개인 보관함과 창고 보관함 사이에서 왔다 갔다 하게 만드는 것 자체가 매우 불편을 야기할 것으로 본다. Tablet 종류가 아닌 Phone 종류의 모바일 기기들은 크기가 작은데다, 마우스처럼 좌우 클릭을 구분할 수 있는 인터페이스가 아니기 때문에 더욱 그렇다.
따라서, 아예 게임 자체에서, 보관함은 같은 계정 내에서도 캐릭터 별로 구분되며, 보관함에 들어가는 아이템은 어떤 형식으로도 공유하지 않는 방식으로 설계한다.

- **B-20-1-5.** 모든 아이템들은 보관과 축적이 가능하다.
: 다만, 보관 / 축적하는 장소와 개수에는 제한이 있을 수 있다.

- **B-20-1-6.** 아이템의 보관과 축적의 제한 사항은 해당 아이템의 종류와 성격에 따라 서로 다르게 구현할 수 있다.

※ 아이템 보관함에 보관하는 아이템들은 아이템 보관함을 구성하는 '아이템 슬롯' 단위에 의해 축적에 제한을 받을 수 있다. 아이템 슬롯이 아닌 단위로 축적하는 경우에도, 최대 축적 단위에 제한 값은 존재한다. 컴퓨터 안에서 무한한 값은 존재하지 않기 때문이다.

◆ B-20-2. 아이템 객체의 내부 구성

- **B-20-2-1.** 아이템의 식별 타입은 크게 두 가지로 구성한다.
: 하나는 프로젝트 전반에 걸쳐 사용하는 **일반 타입 코드(General Type Code)**이고, 나머지 하나는 생성된 아이템의 **고유한 식별 번호(Unique Identifier Number)**이다.

- **B-20-2-2.** **일반 타입 코드(Unique Type Code)**
: 일반 타입 코드는 프로젝트 전반에 걸쳐 타입의 고유한 식별자로 사용하기 때문에 별도의 장에서 설명하고 있다. 상세한 내용은 관련 항목을 참조하기 바란다.
아이템에서의 일반 타입 코드는 **그 아이템이 구성하고 있는 형질(Template)을 인식**하는 데 사용하는 번호이다.

※ 쉽게 예를 들자면 이런 식이다.
일반 타입 코드 10 번 아이템은 도적 전용 가죽 갑옷이고, 옵션이 총 4 개 붙을 수 있으며, 구체적으로 고정적으로 붙는 옵션이 2 개이고, 무작위로 붙는 옵션이 1 ~ 2 개라는 식으로 결정해줄 수 있다. 그러면 서버에서는 이 정보를 토대로 가죽 갑옷 아이템을 생성해준다.

- **B-20-2-3.** **유일 식별 번호(Unique Identifier Number)**
: 이 게임은 아이템의 능력 개수와 종류가 고정적이지 않고 늘 변한다. 겉모습이 완전히 같은 아이템들이 내부 능력은 완전히 다를 수 있다.

그렇기 때문에, 생성한 개별 아이템들마다 전부 고유한 식별 번호를 부여해준다. 이는 일반 타입 코드가 표현하는 아이템의 형질(템플릿)정보가 아닌, 실제 **구체적으로 정해진 아이템의 외형과 능력에 대한 내용**들을 담는다.

- **B-20-2-4.** DB에서 아이템 정보를 저장할 때는 아이템을 구성하는 외형 번호와 능력 타입들의 종류, 각 능력 타입들의 적용 값들을 모두 저장해야 한다.
: 구체적인 아이템의 정보는 아이템을 실제 만들 때 정해지기 때문에, 이렇게 하지 않으면 각 사용자들의 캐릭터가 어떤 능력의 아이템을 가졌는지 알 도리가 없다.

※ 여기에는 보안상의 장점도 존재한다.
개별 아이템마다 유일 식별 번호를 가지는 경우, 어떤 불상사로 인해 아이템이 복제되는 상황이 발생하더라도, 사전에 부여한 유일 식별 번호를 이용해서 복사되었는지 여부를 알아낼 수 있다.
만약, 비정상적으로 특정 아이템에 대해 복사가 일어났다고 가정하자. 그러면 복사의 대상이 된 아이템과, 복사된 아이템은 고유하게 부여되어야만 하는 식별 번호가 똑같을 것이다. 정상적인 과정으로 생겨난 아이템이라면 유일 식별 번호가 같을 이유가 없다.

- **B-20-2-5.** 생성한 아이템마다 소유자가 누군지 표시해주는 기능은 없다.
: 이 기능은 그다지 필요하지 않을 것 같다.
왜냐하면, 소유자 캐릭터가 없어지거나, 계정 자체가 사라져버릴 수도 있기 때문이다. 그럴 때마다 해당 소유자가 소유했던 모든 아이템에 대해 그 이력을 갱신해주는 게 그렇게까지 큰 의미가 있을까?

- **B-20-2-6.** **아이템 객체의 생성과 폐기는 반드시 서버에서만 이루어진다.**
: 클라이언트는 서버에게 아이템의 생성과 폐기를 요청만 할 수 있을 뿐, 이에 대한 어떠한 주체적인 권한을 갖지 못한다.

※ 아이템의 식별 번호, 부여된 능력 값에 대한 모든 사항들은 서버에서 제공하는 데이터만을 신뢰한다.
클라이언트가 아이템에 대해 가지고 있는 정보들은, 아이템을 외형 모델 리소스, 아이콘 / 설명 팝업 창 GUI에 연결시키는 정보들이다.

- **B-20-2-7.** 아이템 객체의 생성 및 삭제 과정에서는 미리 만들어진 아이템 객체를 재활용하는 과정이 없다.
: 클라이언트에서는 주로 객체를 재활용하는 패턴을 사용하지만, 서버에서 아이템 생성할 때는 이런 방식을 쓰지 않는다.

※ 생각해보면, 외형과 능력 종류, 능력 값까지 동일한 아이템이 나올 확률 자체가 거의 없기 때문에, 캐싱 작업을 하는 것 자체가 별로 의미도 없고 낭비에 불과하다.

캐싱이란 건 모아놓은 컨테이너에 검색을 해서 대충이라도 걸리는 게 좀 있을 때나 의미가 있지, 종류 자체가 원자적인 방식에 가까울수록 걸리는 게 없기 때문에 괜히 검색 시간만 낭비할 뿐이다.

- **B-20-2-8.** 아이템은 최대 6개까지의 상태효과로 구성할 수 있다.
: 일반적으로 상태효과는 부여효과를 통해서만 발휘해야 하지만, 아이템은 예외적으로 상태효과를 직접 소유하는 객체이다.

※ 시각을 좀 다르게 한다면, 아이템 객체는 부여효과의 특수한 경우로 볼 수 있다. 본래 부여효과 객체는 상태효과 객체 여러 개로 구성할 수 있으므로, 객체 모델링으로 봐서도 합법적인(?) 경우라고도 볼 수 있다.

- **B-20-2-9.** 아이템의 등급에 따라, 가질 수 있는 상태효과의 개수가 다르다.
: 낮은 등급의 아이템은 더 적은 종류의 상태효과를 가지고, 높은 등급의 아이템은 더 많은 개수의 상태 효과를 가진다.

◆ B-20-3. 아이템의 성질

- **B-20-3-1.** 아이템이 어떤 식으로 작용하는지를 아이템의 성질에 관한 속성으로 분류해야 한다.

- **B-20-3-2.** 아이템의 성질을 분류하는 속성 값은 모든 아이템들이 공통적으로 가지고 있는 값으로 구현해야 한다.

- **B-20-3-3.** **각 종류의 아이템들은 여러 개의 성질을 동시에 가질 수 있다.**
: 즉, 구현할 때, 비트 필드(Bit Field)의 방식으로 구현할 수 있다.
다만, 일부 성질들의 조합은 (대개 개념적으로 말이 안 되는 조합이라서) 무효할 수 있다.

- **B-20-3-4.** 아이템의 성질 분류는 아래와 같다.
: 모든 분류는 가능(true 또는 1)과 불가능(false 또는 0)의 두 가지 값으로 표기할 수 있다.

분 류	설 명
판매 가능 (Enable To Sell)	• true 이면 상점에 팔 수 있는 아이템이다. • false 이면 상점에 팔 수 없는 아이템이다.
(보관함에)보관 가능 (Storable)	• true 이면 아이템 보관함(그것이 캐릭터의 가방이든, 별도의 공용 창고이든)에 보관할 수 있는 아이템이다. • false 이면 아이템 보관함에 보관할 수 없는 아이템이다. : 그렇다고 해서 이 아이템의 개수 축적을 막는 건 아니다.

소모성 (Consumable)	• true 이면, 이 아이템은 사용할 때마다 일정한 개수를 소모하는 아이템이다. • false 이면, 이 아이템은 사용하더라도 아이템의 개수를 소모하는 일이 없는 아이템이다.
개수 없는 아이템 (Enable Emptiness)	• true 이면, 이 아이템은 개수가 0 개인 아이템 상태를 허용한다. : 사용할 수는 없어도, 보관함 슬롯이나 축적 단위는 차지하고 있다는 의미이다. • false 이면, 이 아이템은 개수가 0 개인 경우, 해당 슬롯의 아이템 객체가 파괴된다.

※ 일부 성질들은 실제 구현에 따라 실질적으로는 별 의미가 없을 수 있다.
예를 들어, 화폐나 재료 등을 별도의 전용 슬롯(이나 축적 위치)에 두는 경우에는 개수 없는 아이템 옵션을 허용하지 않는다고 해서 아이템 인스턴스를 제거할 수는 없다. 보통 화폐 단위 등은 돈 한 푼 한 푼마다 인스턴스 객체를 만들어두지는 않기 때문이다.

◆ B-20-4. 아이템의 분류 체계

- **B-20-4-1.** 아이템 분류 체계에 있어 가장 큰 분류는, **어떤 아이템 객체를 생성했을 때, 그 아이템의 세부 내용이 동일한 종류의 다른 아이템 객체와 완전히 같은지, 다른지에 대한 점**이다.
: 아이템 타입 A로 생성한 두 아이템 객체 a, b, c는 각 객체의 내용이 완전히 동일하거나, 혹은 a, b, c의 내용이 각각 다를 수 있다.

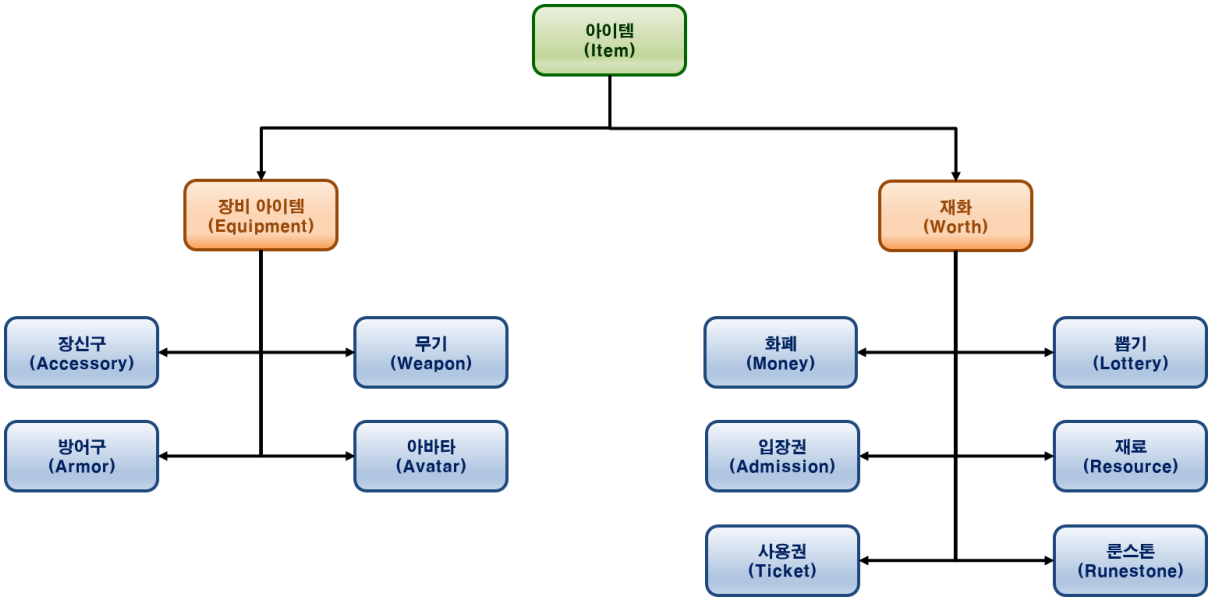
- **B-20-4-2.** 따라서, 아이템의 분류체계는 아래의 두 가지 큰 분류로 나눌 수 있다.

분 류	특 성
장비 (Equipment)	• 캐릭터의 특정한 장비 위치에 장착할 수 있다. • 아이템 보관함의 슬롯 공간을 차지하는 방식으로 축적한다.
재화 (Worth)	• 축적을 위한 별도의 전용 슬롯이 존재한다. • 일반적으로 화폐와 티켓 종류들은 전부 재화료 분류한다.

- **B-20-4-3.** 위의 최상위 분류 외에도, 각각 장비와 재화들은 다시 세부적인 성질에 의해 하위 분류가 존재할 수 있다.

※ 설계하는 관점에서, 아이템을 분류하는 관점은 정말 다양할 수 있다.
일반적으로는, 상위 분류로 갈수록 (좀 더 프로그래머의 방식에 가까운) 일반적인 특성으로 아이템을 분류하려 할 가능성이 높다. 반대로 하위 분류로 갈수록 (좀 더 기획에 가까운)특정한 게임의 내적인 분류 관점에 의해서 분류 방식을 결정할 확률이 더 높다.
예를 들자면, 보관 가능한 아이템과 화폐 아이템에 대한 구분은 좀 더 일반적인 분류이지만(따

라서 상위 분류에 속할 가능성이 높지만), 캐릭터의 장착 부위에 따른 분류라든가 해당 게임 내에서만 필요한 특수한 재료 아이템의 종류에 따른 분류는 하위 분류로 두는 게 바람직하다.



< 아이템 분류 체계(실제 제품에서는 세부 분류는 조금 달라질 수 있다.) >

◆ B-20-5. 판매 가능 / 불가능한 아이템

- **B-20-5-1.** 플레이어의 선택에 따라 해당 아이템을 판매할 수 있는지 여부를 결정한다.

- **B-20-5-2.** 판매 단위는 오직 게임 플레이 화폐로만 고정한다.
  : 다른 아이템 단위로 판매할 수 없도록 한다.

- **B-20-5-3.** 아이템의 판매가 가능하지 않다고 해서, 그 아이템을 보관함 등으로부터 제거할 수 없다는 의미는 아니다.
  : 그러니까, 돈 받고 팔 수는 없어도 그냥 버릴 수는 있다는 뜻이다. 물론, 일부 필수적인 경우에는 팔거나 버릴 수 없게 만들 수도 있다.

- **B-20-5-4.** 모든 재화들은 판매 불가능한 아이템이다.
  : 화폐 단위처럼 취급하는 아이템들이 재화들이다. 재화로 다른 재화를 구입은 할 수 있어도, 되팔 수는 없다.

◆ B-20-6. 보관 가능 / 불가능한 아이템

- **B-20-6-1.** 캐릭터의 물품 보관함(아이템 인벤토리)에 보관 가능한지 여부를 결정한다.

- **B-20-6-2.** 캐릭터의 물품 보관함에 보관 가능한지 여부는 해당 아이템을 장착할 수 있는지 여부와 전혀 관계가 없다.
: 이 부분은 아이템의 성질 분류 말고, 종류에서 장비(Equipment)로 분류할 수 있어야 가능하다.

- **B-20-6-3.** 모든 장비(Equipment)들은 보관 가능한 아이템이다.
: 장착 / 탈착이 가능해야 하기 때문에, 장비 아이템들이 보관 가능하지 않다면 모순이 된다.

- **B-20-6-4.** 모든 재화(Worth)들은 보관 불가능한 아이템이다.
: 여기서의 보관 가능성은 오직 캐릭터의 물품 보관함, 즉 보관 슬롯을 차지하는 형태로 보관할 수 있는지 여부를 나타낸다.
모든 재화들은 각 재화의 종류에 따라 전용으로 사용하는 단 하나의 적재 단위가 있다. 이는 캐릭터의 물품 보관함 슬롯과는 완전히 별개의 단위이다. 따라서, 재화들은 모두 캐릭터의 물품 보관함에는 보관할 수 없는 아이템이 되어야 한다.

◆ B-20-7. 소비 가능 / 불가능한 아이템

- **B-20-7-1.** 아이템을 사용할 때마다 개수가 줄어드는 아이템을 소비성 아이템(Consumable)이라고 한다.

※ 생명력이나 마력을 회복하는 물약(Potion) 종류들은 RPG에서 흔히 등장하는 대표적인 소비성 아이템들이다.
그 외에도 일회성으로 사용하는 각종 마법 주문서, 강화 주문서, 화살 등을 구현했다면, 그것 역시 소비성 아이템으로 볼 수 있다.

- **B-20-7-2.** ~~아이템을 소모하는 타입에는 소모 방식(Consume)과 충전 방식(Charge)이 있다~~.
아이템의 소모 방식은 오직 하나뿐이다. 사용할 때마다 지정한 개수가 차감되는 방식만 존재한다.

※ 기존의 충전 개념은 각 아이템 옵션 별로 적용해야 한다. 이렇게 개념을 변경한 이유는 다음의 두 가지 이유 때문이다.
먼저, (장착하는) 아이템의 능력 옵션은 캐릭터의 기본 능력 향상에 관한 내용만 다룬다고 한정할 수 없다는 점이다.
능력 옵션이 캐릭터의 스킬에 대한 영향력을 주는 경우, 아이템의 능력 옵션은 단지 기본 능력치와 관련한 객체보다 한 단계 더 상위 개념으로 추상화해서, 스킬 옵션에 대한 변경 내역까지 다룰 수 있게 구조를 잡아야 한다.
그리고 두 번째로는, **아이템 옵션마다 충전 개수가 달라야 할 수 있다는 점이다.**
예를 들어, 특정한 아이템 옵션의 기능 A는 충전 개수가 10개, 옵션 기능 B는 충전 개수가 20

개인 아이템을 디자인해야 할 수도 있다.
즉, **충전 기능은 어디까지나 아이템 옵션을 다루는 단계에서 더 적절한 개념**이다. 아이템 그 자체의 성질을 다루는 단계에서 충전 개념이 들어가 버리면, 오히려 아이템의 능력을 표현하는 디자인을 더 한정해버리는 결과를 가져온다.

~~- B-20-7-3. 소모 방식(Consume)~~
~~: 아이템을 다 사용해서 아이템 개수가 0이 되면, 아이템 슬롯이 비워지는 방식이다.~~
~~사용자 입장에서 느끼기에는, 아이템 그 자체가 소모되면서 하나씩 사라져간다고 볼 수 있다.~~

~~- B-20-7-4. 충전 방식(Charge)~~
~~: 아이템에는 일정한 충전 개수가 존재하고, 이 충전 개수가 0이 되면 아이템은 더 이상 충전 개수가 남아 있어야 발휘할 수 있는 능력을 쓰지 못하는 상태가 된다. 하지만 **아이템이 슬롯에서 사라지거나 하지는 않는다.**~~
~~충전 방식이라고 되어 있지만, 정말로 능력의 사용 횟수를 충전하는 게 가능한지 여부는 조건에 따라 다르다.~~

~~※ RPG에서 물약 종류나, 한 번 쓸 때마다 사라지는 두루마리 등의 아이템은 소모 방식으로 소비하는 아이템으로 볼 수 있다.~~
~~만약 번개 마법이 30개 들어 있고, 한 번 번개 마법을 쓸 때마다 30개의 충전이 하나씩 깎여서, 충전 개수가 0이 되면 더 이상 번개 마법을 쓸 수 없는 지팡이가 있다면, 그 지팡이는 충전 방식으로 소비하는 아이템이다. 마법 상점에서 다시 돈을 주고 충전을 허용할지, 아니면 재충전은 불가능하고 다 쓴 지팡이를 버려야 할지는 게임 디자인에 달려 있다.~~

B-21. 장비 아이템

◆ B-21-1. 정의와 제한사항

- **B-21-1-1.** **캐릭터에게 장착할 수 있는 아이템**을 장비 아이템, 혹은 장착 아이템(Equipment Item)이라고 한다.

- **B-21-1-2.** 장비 아이템은 미리 할당된 각 신체 부위에 등록하면, 사용할 수 있는 조건에 부합하는 한, 항상 효능을 발휘한다.

- **B-21-1-3.** 장비 아이템을 착용할 수 있는 부위
: 이 부분은 관련된 기획서의 요구사항을 따른다.

- **B-21-1-4.** 장비 아이템들은, 각각 착용할 수 있는 부위가 미리 정해져 있다.
: 정해진 부위가 아닌, 다른 부위에 장비 아이템을 착용할 수 없다.

- **B-21-1-5.** 무기 아이템은 한 손으로 사용할 수 있는지, 두 손을 사용하는지 등의 여부는 구별하지 않으며, 무조건 장비 슬롯 한 칸을 차지한다.
: 무기가 한손 무기인지, 양손 무기인지, 쌍수 무기인지는, 그 무기의 디자인과 애니메이션 데이터에 의해 구별되어 보일 뿐, 내부 시스템에서는 그에 대한 구분 없이 그냥 무기 장비로만 구분한다.

- **B-21-1-6.** 장비 아이템들은 착용 장비(Wearable Item)와 장신구(Accessory)로 나뉜다.

- **B-21-1-7.** **착용 장비(Wearable Item)**
: 착용 장비는 장비 아이템 중에서, 착용하면 외형이 바뀌는 종류들이다.
이들은 구체적으로 파츠(신체형 파츠와 부착형 파츠) 객체로 표현한다.

- **B-21-1-8.** **장신구(Accesory)**
: 장신구는 장비 아이템 중에서, 착용해도 외형이 바뀌지 않는 종류들이다.
이들은 장착 / 비장착 여부에 따라 캐릭터의 현재 능력 값에 변화를 주지만, 이들은 3D 세계에서 표현하는 별도의 모델 리소스는 없다.

- **B-21-1-9.** **모든 장비 아이템들은, 사용할 수 있는 직업 타입이 각각 정해져 있다.**
: 예를 들어, 전사 직업을 가진 캐릭터가 사용할 수 있는 갑옷 아이템은 마법사 캐릭터는 사용

할 수 없게 제한한다.

※ '모든' 장비 아이템에 대해 직업 타입을 정할 수 있는지에 대한 문제는 숙고가 필요하다.
확실히 착용 장비 아이템이라면, 직업 간 제한을 둔다고 해도 그다지 이상하지 않다. (전사 캐릭터가 마법사 지팡이 휘두르면 모양이 이상하잖아.) 특히, 착용 장비의 경우에는 반드시 애니메이션 정보에 대한 문제가 같이 따라다닌다. 공통으로 사용할 수 있는 종류의 착용 장비가 많을수록, 애니메이션 리소스의 요구량도 직업 전용 방식을 채택할 때보다 몇 배씩 늘어난다.

그런데 장신구는 얘기가 좀 다르다. 장신구는 비록 장비 아이템이긴 해도 외형적으로 드러나는 게 아무 것도 없다. 직업 전용이 아니라고 해서 딱히 리소스 작업이 늘어날 일은 없다는 뜻이다. 하지만 같은 계정 내에서도 아이템을 옮길 방법이 '전혀' 없기 때문에 사실상 어떤 식으로든 직업 제한이 존재하게 되어 버렸다.
장신구까지 직업 제한을 두는 건 확실히 모양이 우스워 보이지만, 실제 현재 시스템 기획 상으로는 그렇게 될 수 밖에 없다. (아이템의 계정 간 공유 및 이동은 전혀 불가능하므로...) 만약 사용자들이 장신구까지 직업 제한이 있는 게 이상하다고 여긴다면, 그래서 플레이어의 다른 캐릭터들에게 장신구를 공유할 수 있는 UI를 요구한다면 어떻게 해야 하는가?

- **B-21-1-10.** 모든 장비 아이템은 반드시 하나의 핵심적인 게임 능력치를 가진다.
: 그러니까, 무기에는 공격력, 방어구에는 방어력이다.
이러한 핵심 능력치는 착용 장비 아이템을 생성하는 경우, 절대로 빠지지 않는 능력치이다.

※ 장신구의 경우에는 직접적인 전투 장비로 보여지지 않기 때문에, 공격력 또는 방어력에 대한 핵심 능력치를 소유하는 게 좀 이상하게 느껴질 수도 있다.
그렇지만 장신구 아이템에도 전투 장비와 마찬가지로 강화 기능을 넣기를 원했고, 강화가 가능하기 위해서는 강화할 수 있는 방향이 미리 정해져 있는 핵심 능력치가 있어야 한다는 결론에 도달했다.
능력치(게임 상태값 Game Status)가 공격력과 방어력만 존재하는 게 아니니만큼, 장신구에 좀 더 어울리는 핵심 능력치의 종류로 대체할 수도 있을 것이다. (추가 생명력, 추가 마력과 같은 종류들 말이다.)

◆ B-21-2. 장비 아이템의 표현

- **B-21-2-1.** 물품 보관함에 있는 아이템들은 아이콘으로 표현한다.
: 아이템 한 종류에 아이템 아이콘 1개로 표현한다.

- **B-21-2-2.** 일반적으로, 캐릭터가 착용이 가능한 아이템들은 아이템에 대한 모델과 FX들을 가진다.
: 장비할 수 있음에도 불구하고 아이템 UI 아이콘의 형태로만 보여지는 경우도 존재한다.

※ 이런 경우는 대개 모델링 데이터가 있어봐야 너무 작아서 플레이어의 눈에 띄지 않는 경우다. 그러나 3D 모델이 있는 아이템이든, UI 아이콘만 존재하는 아이템이든, 기획적으로는 하나의 종류 값에 의해 분류할 수 있어야 한다.

- **B-21-2-3.** 아이템 등급에 따른 아이콘의 변화
: 아이템의 등급에 따라, 물품 보관함에 보관하는 아이템들의 아이콘 테두리 색상을 변경해야 한다.
상세한 내용은 아이템 관련 기획서의 요구사항을 참고한다.

- **B-21-2-4.** 아이템 등급에 따른 외형 변화
: 아이템의 등급에 따라, 아이템을 장착했을 때의 3D 모델 리소스에는 추가로 FX가 더 붙을 수 있다. (더 멋지게 보이도록 하기 위한 장치임)
상세한 내용은 아이템 관련 기획서의 요구사항을 참고한다.

- **B-21-2-5.** 아이템 외형 교체
: 아이템의 외형은 교체할 수 있다. 따라서, 코드를 구현할 때 아이템의 외형이 교체 가능한 방식으로 설계해야 한다.
상세한 내용은 아이템 관련 기획서의 요구사항을 참고한다.

- **B-21-2-6.** 미션, 퀘스트와 관련한 특수한 아이템들을 보관해야 하는 경우, 일반 아이템과의 물품 보관함과 분리해서 보관해야 한다.

※ 물품 보관함의 보관 개수는, 과금 요소를 적용해야 할 가능성이 높다. 하지만 분명히 내 돈 내고 보관함 슬롯 20개를 더 샀는데, 그 중에서 4, 5개 슬롯이 게임 플레이의 전리품과 직접 관련도 없는 퀘스트 아이템 등으로 채워져 있으면 플레이어 입장에서 매우 화가 날 게 분명하다.
아예 물품 보관함 자체가 공짜라면 몰라도, 물품 보관함 슬롯을 과금으로 판매하는 입장이라면, 퀘스트 아이템 등 보상과 직접 관련 없는 요소들이 플레이어의 아이템 보관 영역을 차지하게 만드는 건 매우 안 좋은 판단이다.

- **B-21-2-7.** 게임 플레이 장면을 진행하는 도중에, 적을 물리친 보상으로 땅에 떨어지는 아이템들은 별도의 3D 모델 리소스로 표현한다.

※ 성능을 고려한다면, 아예 땅에 떨어지는 아이템에 대한 모델을 표현하지 않는 방식으로 제작을 하는 게 가장 좋은 방법이다.
하지만 UX를 고려할 때, 보상으로 아이템을 그냥 플레이어의 보관함으로 옮겨주기보다, 땅에 떨어뜨리게 하고 그걸 플레이어의 캐릭터가 '줍도록' 느끼는 방식이 더 실감날 것으로 판단했다.

- **B-21-2-8.** 게임 플레이 장면에서 등장하는 보상용 아이템 모델들(땅에 떨어진 아이템 모델)은

아이템의 큰 분류에 따라 고정된 패턴을 가진 '간단한' 모델들을 공통으로 사용한다.

※ 땅에 떨어진 보상 아이템들을 실제 플레이어가 장착할 때 보이는 고해상도 모델링으로 적용하지 않는다는 뜻이다.

땅에 떨어진 보상 아이템의 개수가 매우 많을 수도 있는데, 각 보상 아이템을 캐릭터가 실제로 사용할 때의 모양처럼 만들어버리면 성능에 부담을 줄 가능성이 높다. 특히, 갑옷 종류들은 신체형 파츠로 분류하는데, 이들은 애니메이션 정보를 포함하기 때문에 최대한 게임에 등장하는 횟수를 줄여야 하는 종류이다. 땅에 떨어진 아이템을 표현하려고 무거운 애니메이션 정보를 포함하는 신체형 파츠를 사용한다? 말이 안 된다.

그러면 결국, 땅에 떨어진 보상용 아이템 모델을 표현하려면 애니메이션 정보가 포함되지 않은 '별도의' 가벼운 모델 데이터를 만들어야 한다는 결론에 도달한다.
문제는 이 모델 데이터를 등장하는 캐릭터의 장비 아이템 모델의 종류만큼 만든다면 심한 낭비라는 점이다. 리소스 데이터를 적재할 때마다 메모리를 소모해야 한다는 점, 그리고 모바일 기기들의 메모리는 PC 와는 다르게 제약이 심하다는 점을 명심하자.

이런 점들을 고려해서 얻을 수 있는 최적의 결론은, 땅에 떨구는 아이템들의 모델은 종류를 최소한으로 한정하고, 성능에 있어 가벼운 모델 리소스를 적용하는 것이다.
Blizzard 의 Warcraft Ⅲ의 경우, 아예 능력치 책을 제외한 모든 아이템들은 땅에 떨어진 모델을 보물상자 형태로 표현하는 극단적인(?) 방법도 사용한다.

◆ B-21-3. 착용 장비 아이템

- **B-21-3-1.** 장비 아이템(장착 아이템) 중에서, 착용시 캐릭터의 외형이 바뀌는 종류는 착용 장비 아이템이다.

※ 언제나, 착용 장비 아이템들은 파츠 데이터가 따라다닌다고 보면 된다.
코드를 설계할 때도 그런 점을 고려해야 한다.

- **B-21-3-2.** **착용 장비 아이템들은 언제나 직업 전용 아이템**들이다.
: 직업마다 캐릭터들의 모델이 다르기 때문에, 직업 공통으로 착용이 가능한 장비 아이템들은 모델 리소스에 대한 작업 과부하가 생긴다.

※ 캐릭터의 직업(비단 직업 뿐만 아니라 종족이 추가되어도 마찬가지다.)에 상관 없이 착용할 수 있는 장비 아이템들의 문제점이라면, 파츠 모델들의 종류가 아~~~~주 많아진다는 점이다.

한 번 생각해보자.

장검을 들었다고 가정한다. 덩치 큰 우락부락한 오크 남성 근육맨 전사와 가냘픈 엘프 여성 마법사가 들고 있는 장검이 똑 같은 파츠 모델일 수 있을까? 전사에게 크기를 맞춘 장검 파츠로 일괄 통일해버리면, 여성 마법사에게는 그 검의 크기는 건물 기둥 정도 되거나, 베르세르크의 가츠가 들고 다니는 드래곤 슬레이어 정도 되어 보일 것이다. 반면, 여성 마법사에게 맞춘다면, 오크 남성 전사에게는 마치 이쑤시개를 들고 휘두르는 격이 되어 버린다.

결국, 그 어느 쪽도 어울리지 않으므로, 다음과 같은 방법을 쓰게 된다.

장검 아이템 하나를 놓고, 오크 남성과 엘프 여성에 대응하는 파츠 모델을 따로 제작해서, 종족과 직업 타입에 대해 파츠 모델을 연결시키는 방식으로 제작을 하는 방식이다.

장검 같은 경우는 그나마 부착형 파츠라서 어떻게든 크기에 대한 타협을 할 수도 있다. 그러나 신체형 파츠(Skinning Mesh를 사용하는)로 구성해야 하는 갑옷들은 캐릭터 고유의 애니메이션 정보가 일치해야 하므로, 위 방법 외에는 완전히 답이 없다.

문제는, 이렇게 '단 한 개의 아이템'임에도 불구하고 파츠 모델의 리소스가 여러 개가 필요하다는 점이다. 물론, 한 개의 아이템을 5개 만드나, 직업 별로 대응하는 5종류의 아이템을 1개씩 만드나 실제 그래픽 작업은 거기서 거기지만, 사용자 입장에서 봤을 때는, 후자 쪽이 좀 더 **'아이템 종류가 풍부해 보인다.'**

※ 착용 장비 아이템들을 직업 전용으로 제한하는 경우의 이득이라면, 아이템의 '재활용' 범위를 축소할 수 있다는 점이다. 플레이어는 하나의 캐릭터도 아무리 오래 플레이 해서 수많은 착용 장비 아이템을 얻었다 하더라도, 그 아이템들은 현재의 캐릭터 직업에게만 적용될 뿐이다.

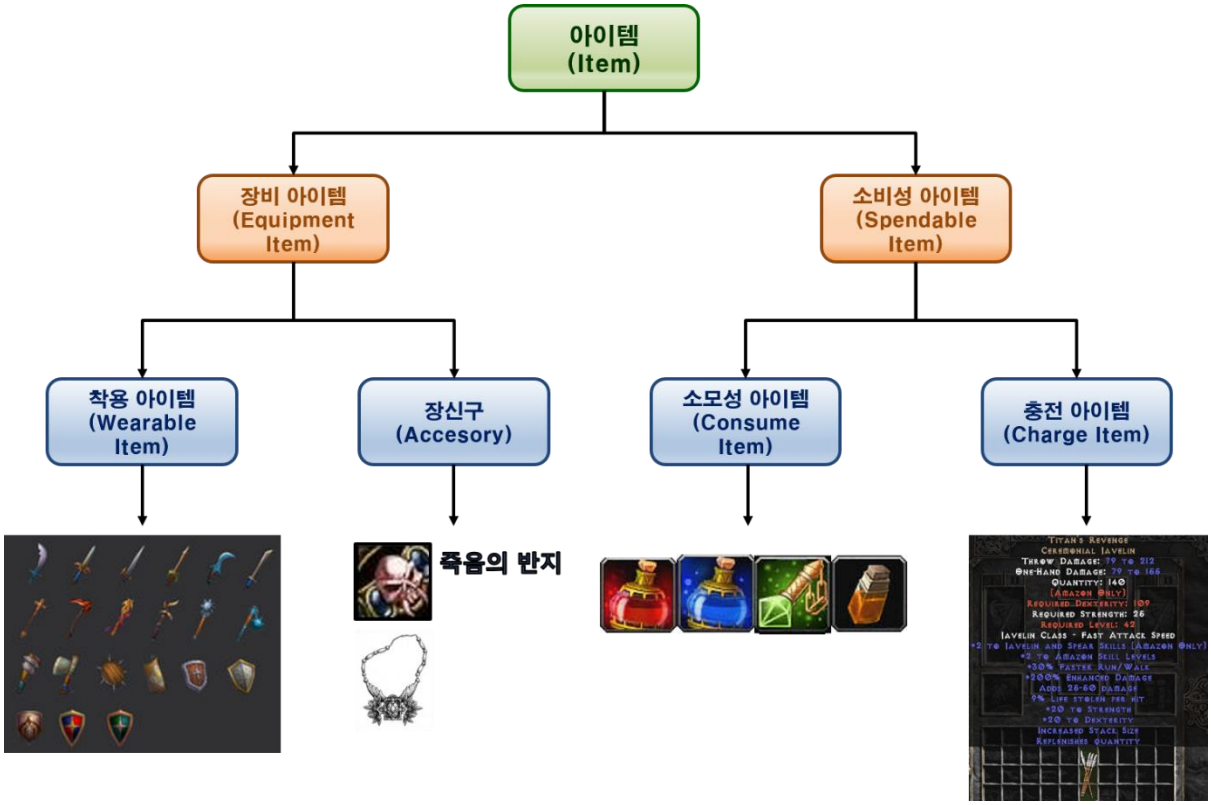
만약, 그 플레이어가 새로운 캐릭터를 키우기로 결심했다면, 기존의 아이템들을 활용할 수 없고 새로운 직업에 아이템들을 열심히 플레이 해서 구하거나, 혹은 **'상점에서 뽑기로 뽑아야 할 것이다.'**

- **B-21-3-3.** 착용 장비 아이템은 다시 무기와 방어구, 장신구로 종류를 나눈다.

- **B-21-3-4.** **무기(Weapon)**
: 핵심 능력치가 공격력인 아이템이다.
무기는 특별히 애니메이션 정보가 필요하지 않은 이상, 대개 부착형 파츠로 만든다.

- **B-21-3-5.** **방어구(Armor)**
: 핵심 능력치가 방어력인 아이템이다.
방어구는 보통 방패 종류를 제외하고는 신체형 파츠로 만든다.

- **B-21-3-6.** 착용 장비 아이템(Wearable Item)들은 캐릭터와 마찬가지로 레벨와 경험치를 가진다.
: 캐릭터처럼 일정한 경험치에 도달하면 다음 레벨로 올라서는 방식으로 성장하게 할 수 있다.



<아이템 분류 모델링>

◆ B-21-4. 형상 장비 아이템(Model Figure Item, 또는 Avatar Item)

- **B-21-4-1.** 외형을 가지고 있는 장비 아이템

B-22. 장비 아이템 보관 / 사용

◆ B-22-1. 아이템의 사용 처리

- **B-22-1-1.** 아이템의 사용은 **반드시 게임 플레이 밖에서만 이루어져야** 한다.
  : 이는, 현재 게임 아키텍처가 클라이언트와 서버의 연결을 지속적으로 유지하지 않는 구조를 전제로 하기 때문이다.

※ 클라이언트의 게임 플레이 과정 내부를 서버에서 모른다는 점이 문제다.
클라이언트에서 처리하는 정보들은 기본적으로 신뢰해서는 안 된다는 점을 생각해본다면, 클라이언트에서 전투를 벌일 때 아이템을 사용한 내역 역시 신뢰할 수는 없다. 그렇다고 해서 서버가 처리하려고 한다면, 클라이언트 – 서버 간 연결 모델 자체를 수정하여야 한다.
클라이언트와 서버는 게임 플레이 도중에 연결을 지속적으로 유지해야 하며, 서버에서는 클라이언트가 플레이하고 있는 게임 방에서의 현재 전투 상황 및 아이템 보유 상황을 점검할 수 있어야 한다. 이건 아예 설계가 달라져 버린다는 의미이다.
클라이언트 – 서버간 지속적인 연결을 필요로 하는 모델은, 필요할 때만 접속해서 처리하고 바로 끊는 방식과 비교해서, 서비스 차원에서 투입해야 하는 자원의 비용도 차원이 다르게 크다.
결국, 비용이라는 중대한 위험 요소를 고려한다면, **클라이언트에서 뭔가를 판정하는 상황 자체를 안 만들도록 디자인하는 쪽으로 가야 합리적**이라고 본다.

- **B-22-1-2.** 게임 플레이에서 아이템을 사용해야 하는 디자인이 필요하다면, 그러한 사용 아이템들은 **그 게임 플레이 방을 빠져나오면 모두 무효화**해야 한다.

※ 쉽게 말하자면, 캐릭터의 생명력을 회복하는 수단을 예로 들어 보자.
많은 RPG 에서는 생명력을 회복하는 물약을 상점에서 판매한다. 그리고 플레이어의 캐릭터들은 상점에서 물약을 구매해서 **'보유한다.'**
서버 측 이슈는 바로 플레이어 캐릭터가 물약을 몇 개 보유했는지에 대한 부분이다. 물약을 보유한 플레이어 캐릭터는 게임을 플레이 하면서, 몬스터와 싸울 때마다 신나게 물약을 소비할 테고... 하지만 이 물약을 소비했는지 아닌지를 판단하는 주체가 서버여야 안전하다는 점이 문제의 핵심이다. 그리고, 그렇게 서버가 일일이 각 플레이어 캐릭터의 물약 개수를 세는 작업을 해주게 만들지 여부는, 설계 상으로 커다란 차이점을 만들어낸다.

물약을 보유한 클라이언트가 물약을 사용했는지를 서버 단계에서 추적하고자 한다면, 클라이언트는 결국 서버와 지속적으로 연결을 유지해야 한다. 그리고, 아이템 사용에 관한 모든 판정을 서버가 해줘야 한다.

이런 상황들이 모두 탐탁지 않고, 피하고 싶다면 그냥 '캐릭터가 보유하는 물약' 같은 아이템이 필요 없는 방식으로 기획하는 것도 좋은 방법이다. 생명력 회복 물약을 게임 플레이 도중에만 획득 / 사용할 수 있고, 현재 게임 플레이 방을 빠져나가자마자 모두 사라지게 한다면, 굳이 캐릭터의 물품 보관함에 있는 생명력 회복 물약이 몇 개인지 서버 단계에서 신경 쓰지 않아도 된다.

- **B-22-1-3.** 반드시 게임 플레이 장면에서도 소비성 아이템을 사용할 수 있어야 게임이 재미있다고 판단한다면, 그러한 소비성 아이템의 재사용 대기 시간을 충분히 길게(일단 5초 이상으로 판단하고 있다.)으로 잡아야 한다.

  또한, 그 경우의 아이템 사용은 **클라이언트가 서버에게 알리고, 서버가 확인을 하는 방식**이어야 한다.

※ 클라이언트 요청 -> 서버 승인 방식은 소비성 아이템을 쓰는 논리 흐름에는 전혀 적합하지 않다.

요청 -> 승인 과정을 네트워크로 하는 행위 자체가 매우 느리기 때문에, 이런 건 반드시 허가가 필요한 중요한 처리 상황에서만 써야 한다. (로그인 처리, 플레이 보상 처리 등) 전투 도중에 물약 하나 드시겠다고 해서 웹 서버로부터의 허가를 4초 동안 기다리는 상황은 말이 안 된다.

결국 최선의 방안은 클라이언트에서 소비성 아이템을 소비하면 즉시 처리하고, 소비 여부를 서버에게 통보하는 방식이다. 클라이언트는 물약을 몇 개 썼는지 통보만 날리고, 잽싸게 다음 게임 논리들을 처리하기 시작한다. 서버는 조금 늦게 통보를 받지만, 통보를 정확히 받았으면 어쨌든 캐릭터가 보유한 소비 아이템을 차감한다.

※ 이런 방식에서는, 결국 클라이언트에게 어느 정도 판단을 맡기는 셈이기 때문에, 보안 문제에 대응하기가 좀 더 까다로울 것이다.

극단적으로 생각하자면, 클라이언트를 잘 요리해서(?) 캐릭터의 실제 물약 보유 수량을 무시하고, 검증 코드들을 회피해서 물약을 사용했음을 알리는 패킷을 계속 서버에 보내게 만들 수가 있다. (가지고 있는 물약은 5갠데, 물약 사용 패킷은 100번이 온다면?)

이런 경우, 서버 측에서는 클라이언트의 이런 조작질을 원천 차단은 못하고(물약의 사용을 판단하는 주체가 일단 클라이언트니까...), 사후 검증을 통해서 클라이언트가 서버의 계산을 벗어나는 이상한 행동을 한다 싶으면 별도 조치를 취하든가 해야 한다.

로그를 남겨서 그 클라이언트의 사용자를 블랙 리스트에 올리든가, 아니면 바로 접속 차단을 해버리든가, 보상을 무시해버리든가 하는 식으로 보안 이슈를 처리해야 할 것이다.

- **B-22-1-4.** 아이템마다 그 아이템의 사용 조건을 지정할 수 있도록 설계해야 한다.
  : 예를 들면, 무기를 사용할 때마다 정령탄이라는 소비성 아이템을 하나씩 소비해야 하는 조건이 붙을 수 있다.

- **B-22-1-5.** 아이템을 사용하는 조건 중에서, 물품 보관함 내에 있는 다른 아이템을 X개 사용하는 조건을 수행할 수 있게 설계해야 한다.
: '이런저런 재료를 X개 소모한다.' 는 식의 조건도 표현해줄 수 있게 하기 위해서다.

- **B-22-1-6.** 1회 사용당 아이템이 몇 개 소모될지에 대해 설정해줄 수 있어야 한다.
: 무작정 한 개씩 소모하겠지 하고 생각해서는 안 된다.

- **B-22-1-7.** 아이템의 사용 처리는 반드시 아이템의 사용 비용이 지불된 이후에 실행해야 한다.
: 그 전에 어떤 이유로든 실패하면, 아이템은 사용해서는 안 되고, 아이템 사용에 따른 효과도 적용하면 안 된다.

- **B-22-1-8.** 플레이어는 어떤 형식으로든, 아이템이 보관되어 있는 위치 또는 순서를 변경할 수 없다.
: 좀 자유도가 떨어져 보이겠지만(?), 아이템의 위치 / 적재 순서 변경조차도 서버와 DB에서의 처리를 필요로 하기 때문이다.

※ 아이템의 위치를 드래그 & 드롭 등의 동작으로 옮기는 행위는 모바일 기기의 터치 기반 인터페이스로는 몹시 불편한 작업 중 하나다. UX의 관점에서도 그다지 좋은 평가를 받기 어렵다.

◆ B-22-2. 물품 보관함(인벤토리, Inventory)

- **B-22-2-1.** 물품 보관함에 플레이어 캐릭터가 획득한 아이템을 보관할 수 있다.

- **B-22-2-2.** 물품 보관함은 여러 개의 슬롯 모음으로 이루어져 있다.
: 슬롯 모음의 모양은 격자 모양이거나, 혹은 1 ~ 2의 열이 여러 줄로 이루어진 목록의 모양일 수 있다.

※ 공간이 부족한 Phone 형식의 모바일 기기의 특성상, 후자의 방식으로 갈 가능성이 높다.
(Tablet 보다 Phone 이 더 많으므로 그렇게 고려하는 건 어쩔 수 없다.)

- **B-22-2-3.** 그렇다고 해서 사용자에게, 물품 보관함의 GUI 외형을 격자 모양과 리스트 모양 중에서 선택하게 할 수 있는 옵션을 제공하지는 않는다.
: 그렇게 정렬방식을 바꾸는 일도 간단한 작업은 아닌데다, 이 게임은 아이템이 물품 보관함 어디에 위치해 있는지가 그렇게 중요하지는 않다.

- **B-22-2-4.** 물품 보관함에 있는 아이템들은 미리 지정되어 있는 기준의 아이템들만 별도로 검색해서 보여주는 탭 인터페이스를 제공한다.

: 구체적으로 어떤 분류를 검색할 수 있는지는 관련된 기획서의 요구사항을 따른다.

<종류 별로 아이템을 검색해서 보여주는 탭 인터페이스의 예>

- **B-22-2-5.** 물품 보관함의 총 슬롯 개수는 무한하지 않으며, 일정한 개수로 제한한다.
  : 무한히 저장할 수 있는 방식의 자료구조가 아니기 때문에 개수를 제한할 수 밖에 없다.

- **B-22-2-1.** 캐릭터의 종류에 관계없이, 물품 보관함에서 사용할 수 있는 최대 슬롯 한계치는 동일하다.
  : 다만... 다 쓰고 싶으면 그만큼 요금을 지불해야 할 뿐이다.

- **B-22-2-2.** **물품 보관함은 플레이어가 생성한 캐릭터마다 별도로 소유**하며, **계정 간에 공유하는 보관함은 없다.**

※ 현재는 '아이템 이동'에 대한 이슈를 최소화하고자 이렇게 정하지만, 향후 사용자들의 반응에 따라, 계정간 공유가 가능한 창고 형식의 물품 보관함을 추가해야 할 가능성이 있다.
만약 그렇게 바뀌게 된다면, 레벨 디자인에도 영향을 주게 된다.
현재의 디자인 상으로는 캐릭터 직업 간 아이템 이동이 '전혀 안 된다.' 그 때문에, 게임 플레이 보상에서 자신의 직업과 관련이 없는 아이템은 전혀 나오지 않아야 한다. (등장한다고 해 봤자, 자신의 다른 캐릭터들에게 옮겨 줄 수도 없으니 말이다.)
그러나, 만약 창고 기능으로 인해 다른 직업 간 아이템 공유가 가능하다면, 사용자들이 게임 플레이 도중에 다른 직업이 사용할 수 있는 아이템도 등장하도록 레벨 디자인을 변경하는 방안도 같이 요구하게 될 수 있다.

- **B-22-2-3.** 슬롯 1개는 1종류의 아이템을 적재할 수 있다.
: 만일 종류가 다른 여러 개의 아이템이 있다면, 종류별로 슬롯을 달리하여 저장해야 한다

- **B-22-2-4.** 물품 보관함이나 퀵 슬롯 슬롯을 사용하는 아이템들의 적재량을 설정할 때는, **슬롯 당 최대 999개**까지만 적재하도록 한계치를 설정해야 한다.

※ 네 자리 수 단위가 넘어가면 숫자가 커지고, 단위를 구분해 줄 쉼표를 붙여야 하기 때문에 UX 관점에서 그다지 좋지 않다고 판단하여 이렇게 결정한다. 더구나, **숫자의 일정한 자리마다 쉼표를 붙이는 문제는 국제화(Internalization, I18N)문제와도 관련이 있기 때문**에, 골치 아픈 상황을 회피하기 위해서라도 세자리 숫자까지만 허용하는 편이 좋다.

P.S – 숫자 자릿수에 대한 국제화 문제를 설명하자면, 자릿수 구분 쉼표는 문화권마다 다르다는 게 문제의 발단이다.

흔히 통용하는 방식(우리나라 포함)은 숫자 세 자리마다 쉼표(,)를 찍는 방식인데, 문화권에 따라 쉼표 대신 마침표를 찍는 곳도 있고, 띄어 쓰는 자릿수가 네 자리인 경우, 심지어 띄어쓰기로 구분하는 곳도 있는 등 '해당 지역 사람에게 자연스럽게 느끼도록 만드는' 처리 자체가 결코 쉬운 문제가 아니다.)
비좁은 아이템 아이콘에 이러한 사항까지 제대로 UX 에 들어맞도록 설계하고 구현하는 부분은 생각보다 까다롭다.
그러니, 굳이 이런 문제까지 신경 쓰지 않도록 하려면 세자리 수에서 최대 한계가 끝나도록 하는 방식이 좋다고 판단한다. ~~(자릿수 두 자리마다 구분하는 문화권에 대한 조사가 필요하다.)~~

- **B-22-2-5.** 슬롯에 적재된 아이템의 개수가 1개가 넘는 경우에는, 해당 슬롯은 **좌측 하단**에 **아라비아 숫자** 방식으로 그 슬롯에 적재되어 있는 아이템의 개수를 표기해야 한다.

- **B-22-2-6.** 같은 종류의 아이템이라도 슬롯 한 개에 적재할 수 있는 최대 용량을 초과한 경우, 다른 슬롯에 저장해야 한다.

- **B-22-2-7.** 한 종류의 아이템이 물품 보관함의 슬롯을 여러 칸 차지할 수 없다.

※ Blizzard 의 Diablo 시리즈라든가, 예전 Sierra 의 Throne Of Darkness 등의 게임들은 물품 보관함 내에서 슬롯을 여러 칸 차지하는 구조였다

◆ B-22-3. 캐릭터 장비 아이템 창

- **B-22-3-1.** 캐릭터에게 장비할 아이템들을 놓는 공간이다.

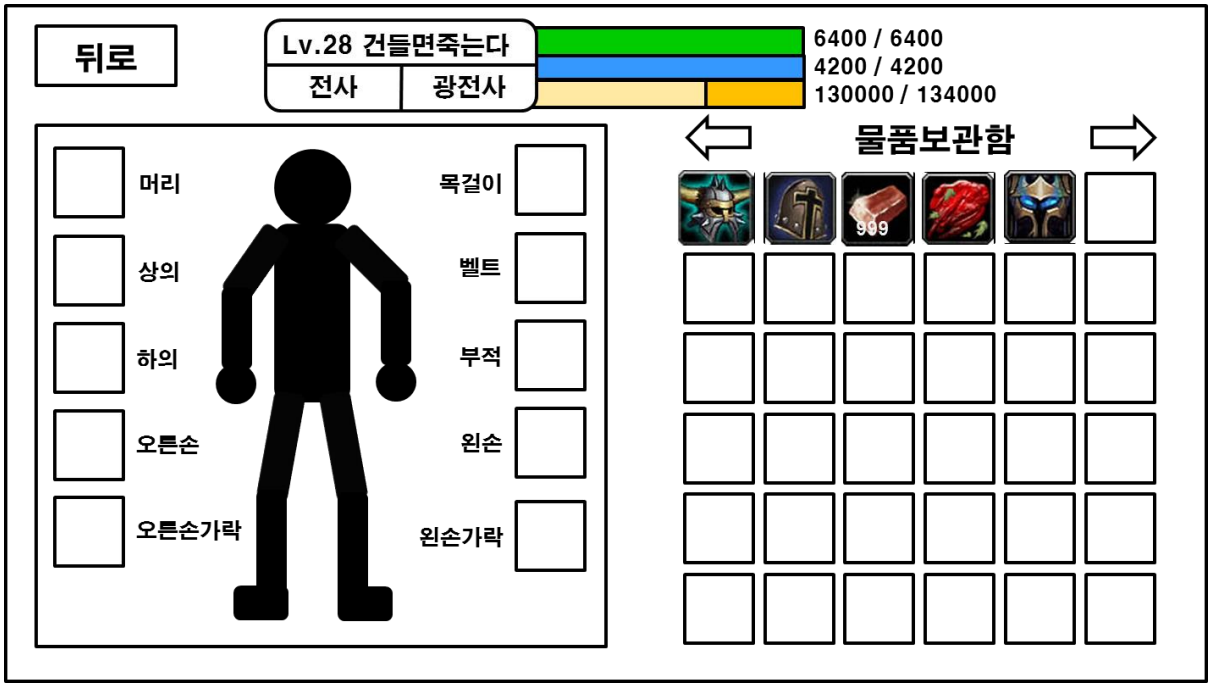
- **B-22-3-2.** 캐릭터의 종류를 불문하고, 플레이어가 조종할 수 있는 캐릭터들의 장비창 구성이나 장비할 수 있는 아이템의 종류 수는 동일하다.

- **B-22-3-3.** 장비 아이템은 장비창에 장착해야만, 장비 아이템의 능력이 캐릭터의 총 능력치에 반영된다.

- **B-22-3-4.** 장비창에 장비 아이템을 (바꿔서) 장착하거나 강화하는 행위는 게임 플레이 장면 밖에서만 가능하다.

- **B-22-3-5.** 게임 플레이 장면 중에서는 장비창 및 장비한 아이템을 볼 수 없다.

※ 게임 플레이 도중에 장비 아이템을 교체하거나 강화하지 못하므로, 굳이 장비창이나 장비 아이템을 보여줄 필요가 없다.
더구나, 그런 GUI 들은 게임 플레이 화면을 덮어버릴 정도로 커야 하는 경우가 대부분이므로 플레이하는 데 방해요소가 될 수 밖에 없으므로 더욱 그렇다.



<물품보관함과 장비창(구체적인 GUI 는 기획에 따라 달라질 수 있다!)>

◆ B-22-4. 회복수단

- **B-22-4-1.** 게임 플레이 도중에 플레이어가 조종하는 캐릭터의 생명력과 마력이 저하되었을 때, 이를 회복할 수 있는 수단을 제공한다.

- **B-22-4-2.** 생명력 회복약
 : 플레이어 캐릭터의 생명력을 회복한다.
  구체적인 회복 방식과 회복량은 관련 기획서의 요구사항을 따른다.

- **B-22-4-3.** 마력 회복약
 : 플레이어 캐릭터의 마력을 회복한다.
  구체적인 회복 방식과 회복량은 관련 기획서의 요구사항을 따른다.

- **B-22-4-4.** 회복수단은 게임 플레이 장면에서만 사용할 수 있다.

- **B-22-4-5.** 회복수단들은 단 하나의 등급만 존재한다.
 : 같은 기능을 여러 등급으로 나눈, 여러 종류의 회복 수단이 존재하지 않는다.

※ '소형 회복 물약', '중형 회복 물약', '대형 회복 물약' 같은 방식으로 나눠 놓고, 각각 100, 200, 300의 체력을 회복한다는 식으로 설정하지 않는다는 뜻이다.
 회복 물약의 존재는 단 하나만 존재하며, '내 캐릭터의 최대 생명력 수치가 몇이든 간에, 복용 시 그 캐릭터는 자신의 최대 생명력의 60%만큼을 회복한다.' 같은 식으로 디자인한다는 뜻이다.

B-23. 장비 아이템 능력 / 강화

◆ B-23-1. 아이템의 생성

- **B-23-1-1.** 아이템 생성 전에, 어떤 종류의 아이템인지 어떤 등급의 아이템인지 결정한다.
  : 이 과정은 아이템의 생성 그 자체와는 직접적인 관련이 없다. 게임 플레이 보상을 어떻게 처리하는지에 더 밀접한 주제다.

- **B-23-1-2.** 서버가 아이템 인스턴스가 사용할 고유 ID를 발급한다.
  : 고유 ID를 어떻게 만드는지에 대해는 관련 항목을 참고할 것.

- **B-23-1-3.** 서버는 자신이 보유한 테이블 데이터에서, 요청이 들어온 고유 코드(Unique Code)의 아이템 종류가 가질 수 있는 상태효과의 종류를 확인한다.

- **B-23-1-4.** **아이템의 고유 코드와 등급에 의해, 아이템이 가질 수 있는 상태효과의 개수가 정해져 있다.** 이 개수만큼 상태효과들을 고른다.
  : 이 작업은 서버가 수행한다.

- **B-23-1-5.** 상태효과들을 여러 번 골라야 할 경우, 같은 종류의 상태효과들을 2번 이상 선택하지 않도록 조정한다.
  : 여러 번 선택한 상태효과는 능력이 중복되어서 한 개의 상태효과로 인식하기 때문이다.

※ 같은 종류의 상태효과들을 중복할 수 있다면, '가장 효율이 좋은' 상태효과들을 여러 번 중복해서, 기획 의도와는 완전히 딴판으로 비정상적으로 강력한 아이템이 등장할 수도 있다.
이런 게 당연히 사용자들에게도 재미있을 수가 없다.

- **B-23-1-6.** 골라낸 상태효과들마다, 아이템의 고유 코드와 등급에 의해 부여할 수 있는 능력치의 최소 ~ 최대 값의 범위가 있다. 이 범위 내에서 무작위 값을 골라온다.
  : 이 작업은 서버가 수행한다.

- **B-23-1-7.** 이제 이 아이템은 아이템의 능력을 구성할 상태효과의 종류 및 적용할 수치가 구체적으로 결정되었다.
  : 데이터베이스는 아이템 인스턴의 고유 ID를 키(Key)로 하여, 아이템에 적용되어 있는 상태효과의 타입들과 적용한 수치를 기록한다.

◆ B-23-2. 아이템의 등급

- **B-23-2-1.** 아이템은 희귀한 정도에 따라 여러 개의 등급으로 나눈다.

- **B-23-2-2.** 높은 등급의 아이템은, 낮은 등급의 아이템에 비해 동일한 종류의 능력에 대해 현재 수치가 더 높고, 아이템이 최고 레벨에 도달했을 때 얻을 수 있는 능력도 더 높다.

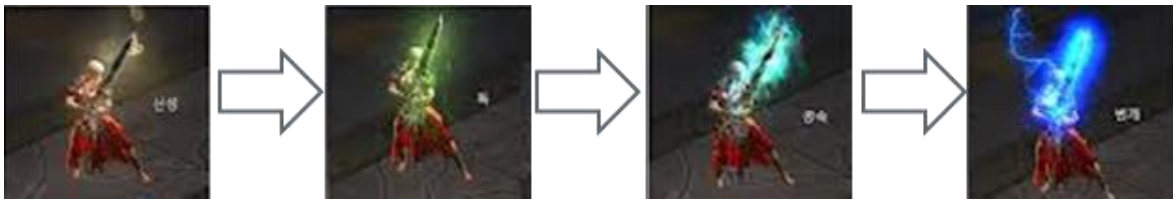
- **B-23-2-3.** 아이템의 세부적인 등급 분류는 관련 기획서의 요구사항을 따른다.

- **B-23-2-4.** 아이템의 외형 중에서 모델 부분은 아이템의 등급에 따라 차별적으로 적용하지 않는다.
  : 하위 등급 아이템이라고 할지라도 상위 등급 아이템에 비교하여, 선택 가능한 모델링 데이터의 종류는 차이가 없다.

※ 아이템의 모델링 데이터까지 아이템 등급에 따라 차별적으로 적용한다면, 제작해야 할 모델도 훨씬 많아지고, 그만큼 메모리에 미리 올려둬야 할 데이터도 더 많아진다. 이는 모바일 기기의 특성을 고려하면 낭비라고 판단했다.

- **B-23-2-5.** 아이템 외형 중에서, 아이템 아이콘 GUI의 테두리 부분의 색상과 아이템 모델에 붙는 FX는 아이템의 등급에 따라 달라진다.
  : 달라지는 상세한 내용은 관련 기획서의 요구사항을 따른다.

※ FX는 상대적으로 아이템 모델링 데이터에 비해 유연하게 사용할 수 있는 리소스이다.
우선 크기와 색상을 조절할 수 있고, 그렇게 조정한 결과를 어디에 적용해도 다 그럴 듯하게 보일 수 있는 범용함이 있기 때문이다.
아이콘 GUI의 테두리 역시 마찬가지로, 그냥 흰색 테두리만 있으면 마음대로 색상 조절해서 쓰면 그만이다. 메모리 점유 면에서 아주 저렴한 방법이라고 볼 수 있다.



<등급에 따라 모양은 같으나 FX는 달라지는 아이템의 예>

- **B-23-2-6.** 더 높은 등급의 아이템일수록 더 희귀하고 얻기 어렵다.
  : 구체적인 등장 확률은 관련 기획서의 요구사항을 따른다.

◆ B-23-3. 아이템의 능력 부여

- **B-23-3-1.** **모든 장비 아이템(Equipment Item)에는 핵심 능력을 필수적으로 부여해야 한다.**
  : 무기는 공격력, 방어구는 방어력을 부여해야 한다.

- **B-23-3-2.** 아이템이 가질 수 있는 상태효과의 최대 개수는 그 아이템의 등급에 따라 달라질 수 있다.
  : 더 높은 등급의 아이템일수록 더 많은 개수의 상태효과를 보유한다.
  등급 별 구체적인 최대 개수는 관련 기획서의 요구사항을 따른다.

- **B-23-3-3.** 같은 종류의 능력을 발휘하는 상태효과라도, 아이템의 등급에 따라 적용할 수치가 더 크거나 작을 수 있다.
  : 아이템 등급이 높으면 더 강력한 상태효과를, 낮으면 더 약한 상태효과를 가지게 한다.

※ RPG 에서 대개 아이템의 등급이 더 높으면 더 강력한 능력을 가진다.
예를 들면, 공격력 10 ~ 20 의 검과 공격력 50 ~ 100 의 검이 있다면, 후자가 더 등급이 높은 아이템이다. 하지만 10 ~ 20, 50 ~ 100 의 수치 차이는 존재하더라도, 두 아이템 모두 공격력의 상태효과를 가지는 점은 동일하다.

- **B-23-3-4.** 장비 아이템의 종류 별 핵심 능력은 다음과 같다.

종류	핵심 능력
머리	공격력
몸통	방어력
무기	방어력
캐릭터 별 특수 장비	방어력
반지	공격력
목걸이	방어력
귀걸이	공격력
팔찌	방어력

◆ B-23-4. 아이템의 강화 레벨 시스템

- **B-23-4-1.** 모든 장비 아이템들은 강화 점수를 모아서, 강화 레벨을 올리는 방식으로 강화할 수 있다.

- **B-23-4-2.** 아이템을 강화하면, 그 아이템의 **핵심 능력치가 상승**한다.

: 무기는 공격력이 상승하고, 방어구들은 방어력이 상승한다.

- **B-23-4-3.** 아이템을 강화하더라도, 그 아이템의 핵심 능력을 제외한 **부가적인 능력들은 향상되지 않는다.**
: 핵심 능력이 아닌 상태 효과들은 더 강해지지 않는다.

※ [얼음 여왕의 지팡이]라는 아이템이 있고, 이 아이템은 공격력 328 ~ 380 에 적중한 적을 10%의 확률로 3 초간 얼어붙게 만들고, 상대방의 체력을 피해의 2%만큼 빼앗아온다고 가정하자.
이 아이템을 강화해도 공격력에 대한 부분만 능력치가 상승하고, 그 외 적을 얼어붙게 만들거나, 체력을 빼앗아오는 부분은 능력이 강화되지 않는다. 아이템의 강화는 무기의 핵심 능력치인 공격력만을 강화하기 때문이다.

- **B-23-4-4.** 핵심 능력치가 상승하는 정도는 별도의 데이터에 의하여 미리 정해져 있다.
: 각 아이템마다 전체 레벨 구간에 능력치 상승 폭을 지정하도록 한다. 공식에 의해 상승할 수치를 정하지 않는다.

※ 공식에 의해 능력치를 상승하게 만든다면, 매우 간단하고 깔끔하게 처리할 수 있는 장점이 있다. 그런 반면에, 게임의 수치적인 균형을 조정하기 위해서 건드리기에는 좋지가 않다.
왜냐하면, **공식이란 물건은, 약간만 수정하더라도 전체 영역에 영향을 미치기 때문**이다.
즉, '이쪽만 조금 조정하는' 행위 자체가 통하지 않는다. 또한, 공식에 의한 결과는 사람이 이해하기에도 직관적이지 않은 경우도 많다. (물론, 이는 상대적으로 훨씬 사소한 문제다. 스프레드시트는 장식으로 쓰라고 있는 물건이 아니다.)
그럴 바에야, 수동으로 데이터를 많이 입력해야 하는 단점이 있더라도, 단계별로 값을 직접 조정할 수 있게 만드는 게 좋을 수가 있다.

◆ B-23-5. 아이템 강화 레벨 초월 시스템

- **B-23-5-1.** 초월 시스템에 의해서 숨겨진 레벨 한계를 돌파하는 기능을 구현하려면, 당연히 처음부터 숨겨진 레벨 한계를 정의해놓아야 한다.

- **B-23-5-2.** 즉, 각 아이템들은 '정상적인' 성장을 했을 때 도달할 수 있는 한계 레벨과, '초월'했을 때 도달할 수 있는 한계 레벨을 각각 가지고 있다.
: 물론, 아이템의 레벨 별 성장 수치들은 처음부터 초월에 의해 도달할 수 있는 한계 레벨까지 정의하고 있어야 한다.

- **B-23-5-3.** 보석이나 룬을 장착하는 기능을 둘 것인지 여부

◆ B-23-6. 아이템 제작 시스템

- **B-23-6-1.** 같은 등급의 아이템을 특정한 조건에 따라 비용을 지불하면, 상위의 아이템으로 합성할 수 있다.

- **B-23-6-2.** 상위 등급의 아이템으로 합성할 경우, **아이템의 능력 옵션들은 전체가 완전히 무작위로 재배열**된다.
  : 사실상, 상위 등급의 아이템을 하나 생성하는 과정이라고 보면 된다.

- **B-23-6-3.** 오직 장착 가능한 아이템들만 상위 등급 아이템으로 제작할 수 있다.
  : 무기, 방어구, 장신구 등이 이에 해당한다.

- **B-23-6-4.** 한 번이라도 상위 등급의 아이템으로 합성할 경우, 이 명령은 취소할 수 없다.
  : 취소할 경우, 더 효율적인 능력 옵션의 조합을 손쉽게 무한정 노릴 수 있게 되므로, 이런 기능을 준다면, 능력 옵션을 교체하는 기능을 가지는 다른 컨텐츠들의 존재 이유가 없어진다.

- **B-23-6-5.** 합성으로 상위 등급을 획득할 수 없는 등급이 존재할 수도 있다.
  : 이건 기획 내용의 방향에 따라 다르다.

◆ B-23-7. 아이템 마법 부여 시스템

- **B-23-7-1.** 일정한 비용을 내고, 아이템의 능력 하나를 다른 것으로 교체할 수 있다.
  : 상세한 내용은 관련 기획서를 참고할 것.

- **B-23-7-2.** 숨겨진 레벨 한계치(초월 시스템)
  : 상세한 내용은 관련 기획서를 참고할 것.

- **B-23-7-3.** 능력치를 교체하는 과정 자체는 아이템을 생성할 때, 아이템이 가질 수 있는 각 상태효과들을 정의하는 과정과 같다.
  : 다만, 플레이어가 교체를 위한 비용을 지불해야 하고, 비용을 지불할 때마다 한 번씩 수행하는 점만 다르다.

◆ B-23-8. 보석을 통한 아이템 강화 시스템

- **B-23-8-1.** 장착 가능한 아이템에는 보석을 끼울 수 있는 홈(Socket)이 존재할 수 있다.

- **B-23-8-2.** 보석 홈은 모든 종류와 등급의 장착 아이템을 불문하고 2개까지 허용한다.

- **B-23-8-3.** 장착 아이템의 부위마다, 끼울 수 있는 보석의 종류가 정해져 있다.

- **B-23-8-4.** 각 부위에 끼울 수 있는 다른 종류의 보석들은, 서로 다른 종류의 캐릭터 능력을 향상시키는 옵션을 아이템에게 부여한다.

- **B-23-8-5.** 보석 홈에 끼운 보석의 등급이 높을수록, 더 높은 수치의 캐릭터 능력을 향상시키는 옵션을 아이템에게 부여할 수 있다.

- **B-23-8-6.** 보석의 능력 중에는 일반 공격을 할 때, 특별한 효과(Effect)를 더해주는 옵션을 부여할 수 있는 보석이 있다.
: 이것을 '발동 보석'이라고 부른다.

- **B-23-8-7.** 발동 보석은 오직 무기에만 장착할 수 있다.

- **B-23-8-8.** 보석 홈의 등장 여부는 아이템의 옵션 중 하나여야 할까? 아니면 무관하게 동작하는 기능이어야 할까?

◆ B-23-9. 셋 아이템(Set Item) 시스템

- **B-23-9-1.** 셋 아이템은, **사전에 정의한 2개 이상의 아이템들을 캐릭터가 장착하면, 아이템에 정의한 능력 외의 추가적인 능력을 더해주는 기능을 가진 아이템 방식**을 말한다.

- **B-23-9-2.** **셋 아이템은 첫 출시에는 포함하지 않으며, 이후에 버전 업데이트를 통해 추가할 컨텐츠이다.**

- **B-23-9-3.** 셋 아이템은 장착 가능한 장비 아이템에만 존재한다.
: 캐릭터에게 장착할 수 없는 아이템은 셋 아이템의 대상이 되지 않는다.

- **B-23-9-4.** 셋 아이템을 구성하기 위해서는 반드시 2개 이상의 아이템을 셋으로 구성해야 한다.
: **아이템 1개로 구성되는 셋 아이템은 존재하지 않는다.**

- **B-23-9-5.** **셋 아이템 효과를 얻기 위해서 필요로 하는 최대 셋 요소의 개수**는, **캐릭터가 장착 가능한 장비 아이템 종류의 개수를 넘을 수 없다.**
: 그러니까, 캐릭터의 장비 아이템이 총 8종류이면, 셋 아이템은 최소 2개에서 ~ 최대 8개의 아이템으로 구성해야 한다. (장착 슬롯이 8개인데 셋이 9개면 영원히 최대 셋을 구성하지 못할

것이다.)

※ 여기에는 보충 설명이 필요하다.
사실, 셋 아이템을 구성하는 모든 아이템 종류의 개수는 캐릭터의 장착 슬롯 개수보다 더 많을 수도 있다. 예를 들면, 장착 슬롯은 8 종류인데, 셋 아이템의 구성 아이템 종류들의 총 개수는 12 개라는 식도 가능하다는 말이다.
다만, 여기서 제한을 두는 점은 다음과 같다.
만약, 셋 아이템의 구성 요소가 되는 아이템들이 전부 합해서 12 개가 있다고 할지라도, 캐릭터가 장착 가능한 종류인 8 개를 넘어서는 셋을 착용해야만 발동하는 아이템 셋의 효과를 정의할 수는 없다는 뜻이다. 즉, 9 개나 11 개의 아이템을 동시에 착용해야 발동하는 셋 효과는 존재할 수 없다.
물론, 2 개, 3 개, 5 개, 8 개의 아이템 셋을 착용할 때 발동하는 효과를 정의하는 건 허용된다.

- **B-23-9-6.** 셋 아이템은 최고 레벨과 등급의 아이템을 기준으로 제작한다.
: 셋 아이템은 다른 아이템들처럼 단품으로만 위력을 내지 않기 때문에 갖추기가 어렵다.
이런 아이템들이 초반 컨텐츠이거나 등급이 낮다면 거의 이용하는 사람이 없기 때문에, 최상위 컨텐츠로 두는 편이 더 적합하다.

- **B-23-9-7.** 셋 아이템의 셋 효과는 그 셋의 구성 아이템을 장착한 개수에 따라 부분적으로 적용할 수 있다.
: 그러니까, 총 5개의 아이템으로 이루어지는 셋 아이템이어도, 그 구성 아이템을 2개 착용할 때의 부분 적용할 효과와, 3개 착용할 때 부분 적용할 효과를 별도로 만들 수 있다는 뜻이다.

~~◆ B-23-10. 아이템의 감정(Identify) 시스템~~

~~- **B-23-10-1.** 희귀 등급 이상의 아이템은 **감정서를 통해서 감정해야 그 아이템의 완전한 능력을 사용**할 수 있다.~~

~~- **B-23-10-2.** 희귀 등급 이상의 아이템은 감정하기 전에는 '감정되지 않은' 상태로 표현하며, 그 아이템의 일반 등급 버전과 같은 능력을 가진다.~~

~~- **B-23-10-3.** 그렇지만, 아이템이 아직 감정되지 않았다고 해서, 그 아이템의 능력이 결정되지 않았다는 의미가 아니다.~~
~~: 어떤 등급의 아이템이건, 아이템이 생성되는 그 시점에 아이템의 능력이 정해진다. 다만, 감정되지 않은 희귀 등급 이상의 아이템들은, 그 능력을 감정하기 전에는 완전하게 발휘하지 못할 뿐이다.~~

~~※ 아이템을 감정하는 시점에 능력 옵션을 정하면 심각한 문제를 야기할 수도 있다.~~

~~아이템의 능력 옵션들은 그 아이템을 획득한 장소의 보물 레벨과, 그 아이템을 획득한 캐릭터의 당시 레벨을 고려해야 한다.~~

~~만약, 아이템을 획득한 시점과 감정한 시점이 극단적으로 달라서, 캐릭터가 아이템을 획득한 뒤에 감정을 하지 않고 내버려둬서, 그 동안 수십 레벨을 올렸다고 해보자. 그러면 초급 지역에 획득한 아이템임에도 불구하고, 이미 높은 레벨이 된 캐릭터의 현재 레벨에 영향을 받아서 원래 얻었어야 하는 능력 옵션보다 훨씬 더 좋은 능력 옵션을 획득할 수도 있다.~~

~~그게 아니더라도, 설계하기에 따라, 아이템을 획득한 지역의 보물 레벨을 반드시 기억하지 않으면 안 될 수도 있다. 사용자가 보물 레벨에 있어 보너스를 받을 수도 있다면? 문제는 점점 더 복잡해질 뿐이다.~~

~~그러므로, 가장 정확하게 반영하려면, 아이템이 처음 생성되는 바로 그 시점에 아이템의 모든 능력도 같이 결정해주는 게 좋다.~~

※ 수정(2015. 03. 16 조수운)

아이템 감정서의 개념을 제거하기로 한다.

이렇게 결정한 이유는, 식별되지 않은 아이템의 개념과, 아이템 제작(등급 상승)의 개념이 충돌하는 면이 있기 때문이다.

기존 설정 상으로는, 희귀 등급부터 아이템 감정서를 사용하도록 되어 있었다. 그런데 만약 마법 등급의 아이템을 희귀 등급으로 등급을 올렸을 경우(즉, 제작했을 경우), 희귀 등급으로 상승한 아이템은 식별되지 않은 상태여야 할까, 아니면 식별된 상태여야 할까?

두 경우에 다 문제가 있다.

전자의 경우, 아이템 제작, 혹은 뽑기를 할 때마다 희귀 등급 이상의 아이템이 최초로 가방에 생성되는 모든 순간마다 식별되지 않은 상태의 희귀 등급 이상의 아이템이 등장하는 점이 문제다. 심지어, 마법 등급 상태일 때는 '식별되어 있었던' 아이템이 희귀 등급이 되자마자 식별되지 않은 상태가 되어야 한다는 점도 문제다.

후자의 경우, 웬만한 상황에서 아이템 식별이 다 되어 있는 상황으로 희귀 등급 이상의 아이템을 제공한다면, 아이템 식별을 할 상황 자체가 별로 없어지게 되고, 아이템 감정서가 딱히 쓸데가 많이 없다. 그냥 없는 거나 그게 그것인 상황이 만들어진다.

아이템 뽑기를 할 때는 어떤가? 아이템 뽑기는 등급이 어떤 게 등장할지 사전에 정해지지 않는다. 만약 희귀 등급 이상의 아이템이 뽑기로 등장했으면, 그 아이템은 식별되지 않은 상태로 등장해야 하나? 하지만, 그보다 하위 등급 아이템은 항상 '식별된' 상태로 등장한다.

이런 애매한 상황들이 많이 생기기 때문에, 아이템 감정서 개념 자체를 삭제하는 게 더 좋겠다고 판단했다.

◆ B-23-11. 아이템 패치 문제

- **B-23-11-1.** 아이템의 능력이 고정되어 있지 않고, 옵션들을 조합해서 생성하기 때문에, 혹여 패치를 통해 옵션의 내부 값 범위와, 값 조합 방식을 변경할 경우, **이미 생성한 아이템에 대해서는 적용할 수 없는 문제가 있다.**

- **B-23-11-2.** 게임을 제작하다 보면, 이전에는 사기적으로 강하게 작용하던 아이템의 옵션을 하향하거나, 지나치게 안 쓰이는 아이템을 위해 옵션의 종류나 값을 조정할 일이 거의 필수적으로 생길 것이기 때문에 문제가 된다.
: 생성한 아이템마다 옵션이 고유하기 때문에, **이를 일괄적으로 패치를 해줄 수가 없다!**

- **B-23-11-3.** 그런 고로, 아이템에 사용하는 옵션의 종류와 값 범위를 조절하는 행위를 한다면, 필연적으로 '구 버전 아이템'과 '신 버전 아이템'에 대한 차이가 발생한다.

※ 이에 대한 가장 직접적인 사례는 Blizzard 의 Diablo Ⅱ이다.
Diablo Ⅱ는 게임 서비스 기간 중에, 아이템에 등장할 수 있는 옵션의 값 범위에 대해 몇 차례 패치를 한 적이 있고, 유니크 아이템에 대해서도 옵션 종류와 값을 조절한 적이 있다.
그 결과, 몇몇 유니크 아이템들은 특정 버전 시점의 이전에 획득한 아이템과 이후에 획득한 아이템 간에 옵션 구성에 차이가 생겼다. (구 발록 스킨 / 신 발록 스킨, 구 샤코 / 신 샤코 등)
또한, 희귀(Rare)등급의 아이템에 나오는 옵션의 값 범위 변화로 인해, Diablo Ⅱ의 오리지널 버전 시절에 등장했던 몇몇 종류의 희귀 등급 아이템들은 확장팩이 나오고 나서도 여전히 값비싸게 거래되었다. 확장 버전에서는 더 이상 등장하지 않는 수치를 가진 옵션을 보유한 아이템이었기 때문이다.

- **B-23-11-4.** 사실 이에 대한 근본적인 해결책은 딱히 없다.(...)
: 몇 가지 제한적으로 통제할 수 있는 방법은 있지만, 완전하지 않다. 근본적으로, 아이템의 종류와 능력을 고정하고 있지 않기 때문에 발생하는 문제다.

※ 이를 통제할 수 있는 방법은 다음과 같은 방식들이 있다.

1. 아이템마다 버전을 두는 방식
: 게임 버전과는 별도로, 아이템마다, 혹은 아이템의 능력을 구성하는 옵션의 종류마다 버전을 두고 이게 변경될 때마다, 버전을 올려주는 방식이다.
이렇게 하면 옵션 구성의 종류 자체가 변경되지 않는다면, 구 버전 시절에 풀린 아이템의 능력 옵션을 패치하는 게 가능하긴 하다.
문제는, 아이템 / 아이템 옵션마다 버전을 관리하는 건 기계적으로 할 수 없는, 순전히 인간적인 영역이기 때문에, 실수하지 않고 올바르게 버전 관리를 해줄 수 있을지는 미지수다.

2. 아이템의 옵션 범위 값을 절대로 조정하지 않는 방식
: 아이템의 능력을 구성하는 옵션 능력들의 수치를 절대로 조정하지 않으면 이런 이슈가 발생할 이유가 없지만, 현실적으로 있을 수가 없다는 게 문제...(...)

3. 아이템의 옵션 범위 값을 상향하는 방향으로만 조정하고, 일괄적으로 컨트롤이 가능한 다른 컨텐츠의 값 조정을 통해서 게임 밸런스를 맞추기

: 그러니까... 소위 '구 버전 아이템을 쓰레기로 만드는'(...) 방식이다.
다른 방식에 비해 장점이라면, '앞으로 만들어낼 아이템'에게만 신경 쓸 수 있다는 부분이다.
구 버전 아이템들은 이제 '중간에 거쳐 가는' 아이템이 되었기 때문에, 그 단계에서 무너진 밸런스는 무시하고, 새롭고 더 강력한 아이템들을 획득하는 시점부터 맞춰준다는 개념이다.
단점은, 새로운 아이템들은 과연 그런 문제가 없겠느냐는 점(...)이다. '새로운 아이템 = 충분히 실전에서 밸런스가 테스트되지 않은 아이템'이기 때문이다.

어쨌든 이 모든 방책들의 공통적인 약점은, 아이템의 옵션 자체를 재설계하는 경우에는 어떻게 해야 하느냐에 대한 부분이다

- **B-23-11-5.** 그래서, 똑같은 아이템, 혹은 아이템의 옵션이 패치에 영향을 받아, **패치 이전과 이후에 생성한 같은 아이템이 옵션에 차이가 발생해도 이를 허용**하기로 한다.

※ 이는 게임이 파티 플레이가 아니고, 아이템 거래를 할 수도 없으므로, 사용자들이 이를 심각하게 인지할 가능성이 별로 없다는 점에 근거한다.
또한 아이템에 자동으로 패치를 적용하게 해주게 만드는 것도 쉬운 일이 아니다.
결국, 어느 정도는 게임의 '고유한 특성'으로 안고 가야 하는 문제라고 할 수 있다. (나도 이러고 싶지는 않았다고...;;)

- **B-23-11-6.** 사실, 어떤 정책을 쓰더라도. 공통적인 약점은, 아이템의 옵션 자체를 재설계하는 경우에는 어떻게 해야 하느냐에 대한 부분이다
: 이 부분은 확실한 해결책은 아직 없는 상태이며, 최대한 레벨 디자인 단계에서 주의하는 방향으로 가닥을 잡고 있다.

※ 구 버전 아이템의 옵션이 너무나도 사기적이어서, 이를 패치로 조정한다고 해도, 이미 이전 버전을 서비스할 때 풀려버린 구 버전 아이템이 있다면, 그것들의 옵션을 어찌할 수 없다는 문제는 사라지지 않는다.

B-24. 화폐 / 재화

◆ B-24-1. 정의 및 제한사항

- **B-24-1-1.** 게임 플레이 내에서 사용할 수 있는 아이템 중에서, **다른 가치로 교환할 수 있는 화폐와 티켓, 일부 아이템**들을 통칭해서 재화로 부르기로 한다.
  : 쉽게 말해, 게임 내 돈으로 통용되는 화폐 단위들과, 이러한 화폐 단위, 또는 다른 가치로 전환할 수 있는 아이템들을 전부 포함한다.

※ 어떻게 생각하면, 다른 뭔가의 '재료'가 될 수 있는 모든 것들은 재화라고 볼 수 있다.
장착 가능한 아이템이나 재료 아이템들처럼, 영속성을 지니고 있을 것 같은 대상들 역시 여기서 예외는 아니다.
더 현실적인 기준으로는, **서비스 운영자가 플레이어에게 '선물할 수 있는' 단위**들은 전부 재화에 속한다고 보면 된다. (운영자가 우편함으로 보낼 수 있는지 여부)

- **B-24-1-2.** 모든 화폐와 티켓은, 축적할 수 있는 한계가 10진수로 **0 ~ 2,000,000,000** 이내의 단위에서 결정해야 한다.
  : 4 바이트(32비트) 부호 있는 정수형의 양수 부분 한계 값 이내(0 ~ 2,147,483,647)여야 한다.

※ 이 부분은 일종의 물리나 수학의 기본 법칙처럼 적용하는 부분이다.
기획적으로 돈 단위를 1000 억 이상 저장할 수 있는 기능이 필요하다면, 이 명세 부분을 반드시 수정해야 한다는 뜻이다. 그렇지 않으려면, 기획적으로 돈의 축적 단위가 4 바이트 정수형의 한계에 머무르도록 수정하든가 해야 한다.
4 바이트 정수형으로는 약 21 억까지만 표현할 수 있으므로, 1000 억을 표현하기 위해서는 아예 숫자를 저장하는 자료형 자체를 8 바이트 정수형 이상으로 바꿔야 한다. (8 바이트 정수형의 값 표현 범위는 -9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,807 이다.)

- **B-24-1-3.** **모든 화폐와 티켓은 절대로 실수형(float, double 등)으로 저장하지 않는다.**
  : 다만, 중간에 계산하는 과정에서 잠시 실수형으로 변환해서 계산에 활용할 수는 있다.
  어쨌든, **최종적인 값 저장은 반드시 정수형으로만 해야 한다.**

※ 컴퓨터에서의 부동소수점 수는 완전히 정확한 값을 나타내지 않는다는 사실을 명심해야 한다.
현재, 프로젝트에서 C#과 .NET 플랫폼을 쓰고 있는 관계로, 굳이 정확한 '소수' 처리를 하고 싶을 경우, decimal 타입의 객체를 이용할 수는 있다.
다만, 이 자료형은 128바이트 자료형이기 때문에, 처리 속도는 사실 좀 느리다고 봐야 한다.

또한, 서비스할 플랫폼들이 이런 자료형을 지원할 수 있는지 여부도 엄밀히 따져야 한다. 모바일 플랫폼들은 제각기 프로그래밍 언어, 플랫폼 환경, 운영체제 등이 다르기 때문이다.

- **B-24-1-4.** 화폐와 티켓의 수량 관련 계산을 할 때, 그 결과 값에 소수 부분이 생기는 경우, **그 소수 부분은 버린다.** (반올림하지 않는다.)

※ 이는 컴퓨터에서의 부동소수점 수를 정수형 수로 변환(Casting)할 때의 동작 그대로를 이용한다는 의미이다.
추가적인 코드나 연산이 필요 없기 때문에, 가장 단순하면서도 확실하게 동작한다.

- **B-24-1-5.** 모든 화폐와 티켓의 구체적인 수량 제한은 관련 기획서의 요구사항을 따른다.

- **B-24-1-6.** 모든 화폐, 티켓, 입장 자원 등의 재화들은 반드시 **사전에 결정되어 있는 단위의 수량으로만 구매할 수 있다.**
: 1개, 5개, 10개, 100개 등으로 사전에 설정한 단위로만 구매해야 한다.

※ 이는 모바일 마켓에서의 과금 거래가 각각 하나의 상품을 구매하는 행위로 취급을 하기 때문이다. 그리고 각 상품은 고유의 ID로 구분을 해야 하며, 1개씩 판매하는 것을 전제로 하고 있다.
그래서 '1 ~ 1000 사이' 따위의 임의 수량으로 구매할 수 없고, 1개씩 판매하는 상품을 여러 개의 종류로, 여러 가격대로 나눠서 판매하는 방식을 취해야 한다. 그리고 이렇게 사전에 정의하고 마켓에 등록한 단위로만 재화를 사고 팔 수 있다.

<사전에 지정한 단위로만 거래할 수 있다.>

- **B-24-1-7.** 아이템의 형태로 보관되지 않는 재화들이라도, 축적할 수 있는 최대량은 미리 정해져 있다.
: 아이템 형태의 재화들은 아이템 슬롯당 축적할 수 있는 양이 이미 정해져 있다. 이런 경우에는 일반적으로 슬롯 1개 당 999개가 가능한 최대 한계치이다.

종류	한계치(Min ~ Max)	설명
과금 화폐	0 ~ 100,000	없음
게임 플레이 화폐	0 ~ 1,000,000	없음
입장 자원	0 ~ 1,000	• 시간이 지남에 따라, 자동으로 회복되는 양의 최대치와는 다른 개념이다.
소탕권	0 ~ 100,000	없음
아이템 뽑기 횟수	0 ~ 100	없음
스킬 룬 뽑기 횟수	0 ~ 100	없음
무작위 뽑기 횟수	0 ~ 100	없음
아이템 형상 변환권	0 ~ 100	없음
스킬 형상 변환권	0 ~ 100	없음
초월 모드 코인	0 ~ 100,000	없음
결투장 코인	0 ~ 100,000	없음
길드 코인	0 ~ 100,000	없음

※ 각 재화마다 한계값을 사전에 정해두고, 재화의 수치에 변동이 생길 때, 항상 값 범위를 점검하는 동작이 필요하다.
특히, 컴퓨터에서는 0 미만의 값이 부호 없는 타입에서는 양의 정수로 작동하기 때문에, 보통 0으로 쓰는 최소 값에 대한 점검도 반드시 해야 한다.

- **B-24-1-8.** 아이템의 형태로 보관되지 않는 재화들은, 획득하는 즉시 각 재화 단위를 저장하는 변수에 축적한다.
: 아이템의 형태로 물품 보관함 등에 보관할 수 없다.

※ 아이템의 형대로 보관되지 않는 재화들은 주로 화폐의 기능을 하는 재화들이다.
과금 화폐, 게임 플레이 화폐, 입장 자원, 뽑기 횟수 등이 이에 해당한다. 모두, 아이템의 형태로 물품 보관함에 보관되지 않고, 사전에 정해진 재화 단위로 수치가 축적되는 특성이 있다.

- **B-24-1-9.** 아이템의 형태로 보관되지 않는 재화들은 최대 한계량을 넘어서서 획득할 수 없다.
: **한계량을 초과한 나머지 값들은 전부 버려진다.**

※ 그러므로, 축적 한계량에 도달하리라고 예상하는 구간에서 사용자에게 알림을 주거나, 아니면 최대 한계량 자체를 획득량에 비해 거의 도달할 수 없을 정도로 높게 설정하는 등의 장치를 둬야 할 것이다.

◆ B-24-2. 재화의 귀속 정책

- **B-24-2-1.** 어떤 재화가 특정한 플레이 단위에 '귀속'된다고 함은, 그 플레이 단위에서만 이용할 수 있는 전용 재화가 된다는 뜻이다.

- **B-24-2-2.** 귀속된 재화들은, 원칙적으로 **귀속되는 단위를 벗어나서 유통하지 못한다.**

※ 사용자 계정에 귀속되는 재화들은 사용자 내부의 캐릭터들은 공유할 수 있으나, 다른 계정으로 옮기지 못한다.
사용자 계정 내 캐릭터에 귀속되는 재화들은, 같은 사용자 계정이 소유하는 캐릭터들일지라도, 재화가 따로 관리되며, 캐릭터 간 재화의 이동이 불가능하다.

- **B-24-2-3.** 사용자 계정 단위에 귀속하는 재화의 종류는 다음과 같다.

- 과금 화폐(Game Cash Money)
- 게임 플레이 화폐(Game Play Money)

- **B-24-2-4.** 사용자 계정 내 캐릭터 단위에 귀속하는 재화는 다음과 같다.

- 스테이지 입장 자원
- 스테이지 소탕권
- (장비)아이템 뽑기권
- 스킬 룬 뽑기권
- (장비)아이템 강화석
- (장비)아이템 초월석
- 스킬 강화석
- 제작 재료
- (장비)아이템 강화 보석
- (장비)아이템 마법 부여석
- (장비)아이템 형상 변환권
- 스킬 효과 형상 변환권
- ~~(장비)아이템 감정서~~
- P vs P 코인
- 길드 코인

※ 사실상, 과금 화폐와 게임 플레이 화폐를 제외한 모든 재화들은 플레이어 캐릭터 단위로 귀속하는 재화들이다.

◆ B-24-3. 게임 내 결제를 통해 구매하는 화폐(과금 화폐, Game Cash Money)

- **B-24-3-1.** 재화의 단위는 **큐빅(Cubic)**로 칭한다.
  : 향후, 큐빅 단위가 등장한다면, 과금 화폐를 의미한다고 보면 된다.

※ 보통 게임들이 과금 화폐를 표현할 때, 보석(Gem) 단위를 많이 사용하는데, 이 게임에서는 이미 보석이 장비 아이템 능력을 향상시키는 부가 아이템 종류로써 등장한다. 물론, 다른 종류의 보석 이름을 사용해도 되긴 하겠지만, 혼란이 올 여지가 있기 때문에, 아예 보석 종류로 한 번에 인식되지 않는 별도의 단위를 사용하는 게 좋겠다고 판단했다.

※ 과금 화폐의 단위 이름은 기획에 따라 변경될 수 있다.
큐빅 외에 몇 가지 적당한 후보들로 생각한 이름들은 다음과 같다.

- 영혼석(Soulstone) : 의외로 많이 씀
- 마력석(Mojo) : ...
- 크리스탈(Crystal) : 상대적으로 덜 혼동되는 광물 종류로...
- 옥(Jade) : 고대 중국에서는 금보다 옥을 더 높이 쳤다. (금새가 아니고 옥새인 이유는?)

- **B-24-3-2.** 실제 세계의 통화(프로그램에서는 통화의 구분을 위해 Cash로 부른다.)를 통해 환전하는 단위의 화폐이다.
  : **실제 세계의 화폐는 오직 이 화폐 단위로만 '환전'할 수 있다.**

※ 기술적인 의미에서는, 실제 세계의 돈을 지불하고, 화폐 용도를 하는 아이템을 일정 수량 '구입'하는 방식이다.
실질적인 사용 목적이나 방식은 현실 세계의 돈과 유사하지만, 완전히 같은 건 아니다.
그 중 가장 대표적인 특징은, 현실 세계의 환율을 전혀 고려하지 않는다는 점이다.

※ 현실 세계의 화폐는 게임 프로그램에서 그대로 적용하기에 문제가 많다.
왜냐하면 세계 각국에서 서로 다른 단위의 화폐를 사용하고 있으며, 환율 역시 시시각각 변하기 때문에, 가치가 늘 조금씩이라도 달라진다. 이런 문제는 게임을 국제적으로 서비스할 때 특히 문제가 되기 쉽다.
그래서 현실 세계의 지역에 상관없이 게임 서비스 내에서 현실 세계의 돈을 대체할만한 통화 단위가 필요하기 때문에 이를 위한 별도의 화폐가 있어야 한다.

- **B-24-3-3.** 이 재화를 획득할 수 있는 컨텐츠들은 다음과 같다.

종류	설명
스토리 모드(일반) 완료	• 이 과정으로 획득할 수 없음
스토리 모드(정예) 완료	• 이 과정으로 획득할 수 없음
~~일일 던전 완료~~	~~• 이 과정으로 획득할 수 없음~~
주간 던전 완료	• 이 과정으로 획득할 수 없음
혼돈 던전 완료	• 이 과정으로 획득할 수 없음
초월 던전 완료	• 이 과정으로 획득할 수 없음
숨겨진 던전 완료	• 이 과정으로 획득할 수 없음
결투장 승리	• 이 과정으로 획득할 수 없음
길드 던전 완료	• 이 과정으로 획득할 수 없음
길드 전쟁 승리	• 이 과정으로 획득할 수 없음
월드 보스 협공 보상	• 이 과정으로 획득할 수 없음
수련장	• 이 과정으로 획득할 수 없음
용병 활동	• 이 과정으로 획득할 수 없음
보유한 재화 판매 (장비, 소모품, 재료 등)	• 이 과정으로 획득할 수 없음
아이템 강화 결과	• 이 과정으로 획득할 수 없음
아이템 제작 결과	• 이 과정으로 획득할 수 없음
일일 미션 달성 보상	• 미션 목표를 달성했을 때, 그 보상으로 주어질 수 있다.
업적 달성 보상	• 업적 목표를 달성했을 때, 그 보상으로 주어질 수 있다.
일일 접속 보상	• 게임에 일정 기간 동안 반복적으로 접속했을 때 보상하는 재화의 종류에 포함할 수 있다.
서비스 운영에 의한 비정기적 지급	• 그 외 비정기적인 방식으로 서비스 운영 단계에서 사용자에게 공급하는 경우 • 예를 들면, 장기 미접속자가 오랜만에 다시 접속했을 때, 플레이 유도를 위해 지급하는 경우를 들 수 있다.
VIP 레벨에 의한 지급 방식 변화	• VIP 레벨을 올렸을 때의 혜택으로 제공할 수 있다. • 게임 플레이 진행 중, 특정한 이벤트가 발생했을 때, 추가적인 양을 획득하게 하거나, 지급하는 양을 지정한 배수만큼 늘려주는 방식 등이다. • VIP 레벨에 따라 지급하는 양이 더 많아질 수 있다.
상설 상점 구매	• 현실의 화폐를 이용해 구입할 수 있다. : 현실 화폐의 종류는 서비스 지역에 따라 다를 수 잇다. • 교환할 양에 대해 미리 정해진 상품 단위가 몇 가지 있고, 플레이어는 이 중에서 선택하는 방식이다. • 구입 횟수나 재사용 대기 시간을 두지 않는다.

	: 이건 전형적인 다다익선(多多益善)...
숨겨진 상점 구매	• 이 과정으로 획득할 수 없음
무작위 뽑기	• 어떤 재화라도 나올 수 있기 때문에, 그 중에 한 종류로써 등장할 수 있다.

◆ B-24-4. 게임 플레이 화폐(Game Play Money)

- **B-24-4-1.** 화폐의 단위는 **골드(Gold)**를 사용한다.
: 만만한 단위가 이거지 뭐...

※ 일반적으로 가장 많이 쓰이는 명칭이다. 매우 직관적이고 목적을 알기 쉽다.
특히, 게임을 많이 해보지 않은 사용자 층에게도 손쉽게 화폐의 목적을 인식시킬 수 있다는 점이 가장 큰 장점이다. (그러니 흔해 빠진 명칭이어도 계속 우려 먹는 거겠지...)

- **B-24-4-2.** 게임 플레이 도중에 지속적으로 획득할 수 있는 화폐이다.
: 일반적인 보상으로 주어지는 화폐의 단위이다.

※ 게임 플레이 중에 거의 항상 획득하는 보상이기 때문에, 일반적으로 다른 단위의 재화보다 큰 단위로 다루어지는 경우가 많다.
예를 들면, 대개의 과금 화폐는 수 천 ~ 수 만 단위의 숫자들을 넘나드는 반면, 골드는 수 십만 ~ 수 천만 단위로 모일 수가 있다.
언제나 등장하는 흔한 화폐이니만큼 가치가 상대적으로 낮을 수 밖에 없다. (화폐 단위 이름과는 다르게, 가치가 낮다...;;)

- **B-24-4-3.** 이 재화를 획득할 수 있는 컨텐츠들은 다음과 같다.

종류	설명
스토리 모드(일반) 완료	• 스테이지를 돌파하는데 실패하더라도, 보상 자체는 이루어질 수 있다. • 스테이지 단계와 난이도에 따라 획득하는 양에 차이가 날 수 있다
스토리 모드(정예) 완료	• 스테이지를 돌파하는데 실패하더라도, 보상 자체는 이루어질 수 있다. • 스테이지 단계와 난이도에 따라 획득하는 양에 차이가 날 수 있다
~~일일 던전 완료~~	~~• 스테이지를 돌파하는데 실패하더라도, 보상 자체는 이루어질 수 있다.~~ ~~• 스테이지 단계와 난이도에 따라 획득하는 양에 차이가 날 수 있다~~
주간 던전 완료	• 스테이지를 돌파하는데 실패하더라도, 보상 자체는 이루어질 수 있다.

	• 스테이지 단계와 난이도에 따라 획득하는 양에 차이가 날 수 있다
혼돈 던전 완료	• 스테이지를 돌파하는데 실패하더라도, 보상 자체는 이루어질 수 있다. • 스테이지 단계와 난이도에 따라 획득하는 양에 차이가 날 수 있다
초월 던전 완료	• 스테이지를 돌파하는데 실패하더라도, 보상 자체는 이루어질 수 있다. • 스테이지 단계와 난이도에 따라 획득하는 양에 차이가 날 수 있다
숨겨진 던전 완료	• 스테이지를 돌파하는데 실패하더라도, 보상 자체는 이루어질 수 있다. • 스테이지 단계와 난이도에 따라 획득하는 양에 차이가 날 수 있다
결투장 승리	• 이 과정으로 획득할 수 없음
길드 던전 완료	• 이 과정으로 획득할 수 없음
길드 전쟁 승리	• 이 과정으로 획득할 수 없음
월드 보스 협공 보상	• 공헌도에 따라 보상받는 양이 달라진다.
수련장	• 이 과정으로 획득할 수 없음
용병 활동	• 용병 활동의 성과로 지급받는다. • 일부는 다른 플레이어의 습격 활동에 의해 약탈당할 수 있다.
보유한 재화 판매 (장비, 소모품, 재료 등)	• 아이템, 소모품, 재료 등의 판매 가격은 전부 게임 플레이 화폐로 처리한다. • 판매 가격은 해당 재화의 가치에 따라 높거나 낮을 수 있다.
아이템 강화 결과	• 이 과정으로 획득할 수 없음
아이템 제작 결과	• 이 과정으로 획득할 수 없음
일일 미션 달성 보상	• 미션 목표를 달성했을 때, 그 보상으로 주어질 수 있다.
업적 달성 보상	• 달성한 업적의 가치에 따라, 보상받는 양이 달라질 수 있다.
일일 접속 보상	• 접속을 연속으로 유지한 횟수에 따라, 보상받는 양이 달라질 수 있다.
서비스 운영에 의한 비정기적 지급	• 그 외 비정기적인 방식으로 서비스 운영 단계에서 사용자에게 공급하는 경우 • 예를 들면, 장기 미접속자가 오랜만에 다시 접속했을 때, 플레이 유도를 위해 지급하는 경우를 들 수 있다.
VIP 레벨에 의한 지급 방식 변화	• VIP 레벨을 올렸을 때의 혜택으로 제공할 수 있다. • 게임 플레이 진행 중, 특정한 이벤트가 발생했을 때, 추가적인 양을 획득하게 하거나, 지급하는 양을 지정한 배수만큼 늘려주는 방식 등이다. • VIP 레벨에 따라 지급하는 양이 더 많아질 수 있다.
상설 상점 구매	• 과금 화폐(큐빅, Cubic)를 이용해 구입할 수 있다. • 교환하는 양에 따라 미리 정해진 상품 단위가 몇 가지 있고, 플레이어는 이 중에서 선택하는 방식이다. • 실제 시간 간격으로 구입할 수 있는 횟수의 제한이 있을 수 있다.

숨겨진 상점 구매	• 이 과정으로 획득할 수 없음
무작위 뽑기	• 어떤 재화라도 나올 수 있기 때문에, 그 중에 한 종류로써 등장할 수 있다.

- **B-24-4-4.** 플레이어가 **게임을 플레이 했을 때, 가장 처음부터 획득할 수 있는 화폐**이다.
: 그래서 그만큼 다른 화폐에 비해 가치가 상대적으로 더 낮다. 고위 컨텐츠로 갈수록 골드의 단위는 커질 수 밖에 없다.

- **B-24-4-5.** 게임 플레이 화폐를 다른 화폐 단위, 즉, 보관함에 아이템 방식으로 보관하지 않는 재화 종류로 교환할 수 없다.
: 게임 플레이 화폐 -> 과금 화폐는 물론이고, 입장 자원, 뽑기 횟수, 형상 변환 횟수 등 숫자로만 축적하는 화폐 기능 단위로의 변환이 불가능하다.

※ 즉, 골드(게임 플레이 화폐)는 매수만 가능하고, 매도가 안 되는 화폐로 간주하면 된다.
획득 경로가 가장 다양하고, 가장 많은 횟수로 획득하는 화폐이기 때문에, 가치를 다른 단위로 전용할 수 있는 경우의 수가 가장 적어야 한다.

◆ B-24-5. 스테이지 입장 자원

- **B-24-5-1.** 단위는 뭘로 하지????

- **B-24-5-2.** 스테이지에 입장하기 위해 소모하는 대표적인 재화 중 하나이다.

※ 모든 스테이지의 진입이 입장 자원만 가지고 있다고 해서 가능하지 않다.
일일 횟수 제한이 있는 경우라든가, 특정한 자격(길드 가입, 점수 등)이 있는지 여부는 별도로 점검하여 입장 자격을 가린다. 스테이지 입장 자원은 해당 스테이지에 정말로 입장할 수 있는지 여부를 가리는 방법 중 하나일 뿐이다.

- **B-24-5-3.** 스테이지의 특성과 난이도에 따라, 입장 자원을 소모하는 양이 달라질 수 있다.

- **B-24-5-4.** 플레이어는 입장 자원을 요구하는 스테이지의 임무에 실패했을 때, 입장 자원의 일부를 반환 받을 수 있다.

스테이지	소모량	실패 반환	설명
시나리오 (난이도 일반)	5	1	• 반복 플레이 횟수 제한 없음
시나리오 (난이도 정예)	10	2	• 반복 플레이 횟수 제한 없음
~~일일 던전~~	~~5~~	~~1~~	~~• 1 일 당 5 회 제한~~ ~~• 매일 0 시에 입장 제한 횟수 초기화~~

주간 던전	5	1	・1 일 당 2 회 입장 제한 ・다음 해당 요일의 0 시에 입장 제한 횟수 초기화
혼돈 던전	5	1	・1 일 당 3 회 입장 제한 ・ 매일 0 시에 입장 제한 횟수 초기화
초월 던전	5	1	・1 일 당 3 회 입장 제한 ・ 매일 0 시에 입장 제한 횟수 초기화
숨겨진 던전	5	1	・스테이지 플레이의 결과 보상 도중에 확률에 의해 등장한다. ・제공하는 보상이 많다. ・플레이어에게 재화 보너스를 제공하는 느낌으로 설계한다.

※ 스테이지에 한 번에 입장할 때, 소모 단위가 1, 2 단위가 아니라 5 이상인 이유는, 실패했을 때 반환해주는 입장 자원의 '거스름돈' (개평?;;) 때문이다.
입장 자원의 소모 단위가 1 이나 2 단위이면, 스테이지 공략을 실패했을 때 환불해주는 방식이 아예 불가능하거나,(소모 단위가 1 인 경우) 지나치게 환불하는 단위의 가치가 커진다.(소모 단위가 2 인 경우, ~~0.1 장 반환하면 되지...~~)

- **B-24-5-5.** 이 재화를 획득할 수 있는 컨텐츠들은 다음과 같다.

종류	설명
스토리 모드(일반) 완료	・이 과정으로 획득할 수 없음
스토리 모드(정예) 완료	・이 과정으로 획득할 수 없음
~~일일 던전 완료~~	~~・이 과정으로 획득할 수 없음~~
주간 던전 완료	・이 과정으로 획득할 수 없음
혼돈 던전 완료	・이 과정으로 획득할 수 없음
초월 던전 완료	・이 과정으로 획득할 수 없음
숨겨진 던전 완료	・이 과정으로 획득할 수 없음
결투장 승리	・이 과정으로 획득할 수 없음
길드 던전 완료	・이 과정으로 획득할 수 없음
길드 전쟁 승리	・이 과정으로 획득할 수 없음
월드 보스 협공 보상	・공헌도에 따라 보상받는 양이 달라진다.
수련장	・이 과정으로 획득할 수 없음
용병 활동	・용병 활동의 성과로 지급받는다.
보유한 재화 판매 (장비, 소모품, 재료 등)	・이 과정으로 획득할 수 없음
아이템 강화 결과	・이 과정으로 획득할 수 없음

아이템 제작 결과	• 이 과정으로 획득할 수 없음
일일 미션 달성 보상	• 미션 목표를 달성했을 때, 그 보상으로 주어질 수 있다.
업적 달성 보상	• 업적 목표를 달성했을 때, 그 보상으로 주어질 수 있다.
일일 접속 보상	• 게임에 일정 기간 동안 반복적으로 접속했을 때 보상하는 재화의 종류에 포함할 수 있다.
서비스 운영에 의한 비정기적 지급	• 그 외 비정기적인 방식으로 서비스 운영 단계에서 사용자에게 공급하는 경우 • 예를 들면, 장기 미접속자가 오랜만에 다시 접속했을 때, 플레이 유도를 위해 지급하는 경우를 들 수 있다.
VIP 레벨에 의한 지급 방식 변화	• VIP 레벨을 올렸을 때의 혜택으로 제공할 수 있다. • 게임 플레이 진행 중, 특정한 이벤트가 발생했을 때, 추가적인 양을 획득하게 하거나, 지급하는 양을 지정한 배수만큼 늘려주는 방식 등이다. • VIP 레벨에 따라 지급하는 양이 더 많아질 수 있다.
상설 상점 구매	• 과금 화폐(큐빅, Cubic)를 이용해 구입할 수 있다. • 교환하는 양에 따라 미리 정해진 상품 단위가 몇 가지 있고, 플레이어는 이 중에서 선택하는 방식이다. • 실제 시간 간격으로 구입할 수 있는 횟수의 제한이 있을 수 있다.
숨겨진 상점 구매	• 이 과정으로 획득할 수 없음
무작위 뽑기	• 어떤 재화라도 나올 수 있기 때문에, 그 중에 한 종류로써 등장할 수 있다.

◆ B-24-6. 스테이지 소탕권

- **B-24-6-1.** 스테이지를 직접 플레이 하지 않고, 보상만을 획득하기 위해 소모하는 재화이다.

- **B-24-6-2.** 스테이지 소탕권을 사용하는 경우, **스테이지의 보상은 얻을 수 있지만, 스테이지 돌파에 의한 캐릭터 경험치는 획득할 수 없다.**

※ 스테이지 소탕권을 사용 결과와 직접 플레이 하는 결과에 차이가 전혀 없다면, 소탕권을 사용하는 쪽이 훨씬 효율이 더 좋기 때문에, 직접 플레이는 완전히 사장될 수 밖에 없다.
그렇기 때문에, 편리함과 맞바꾸는 반대 급부가 필요하다.

- **B-24-6-3.** 스테이지 소탕권은 1회 사용하는 GUI 명령과 연속적으로 X회 사용하는 GUI 명령이 공존한다.
: 단, 연속 사용의 횟수 자체는 고정되어 있고, 플레이어가 연속 횟수를 조절하지는 못한다.

※ 이건 그냥 전적으로 사용성에 관한 문제다.

플레이어가 소탕권을 몇 장이나 가지고 있는지 모르는데, 이걸 수동으로 조절하게 하려면, 아무리 따져봐도 **최소한 '버튼 한 번 터치'보다는 훨씬 복잡은 GUI 가 필요**하다. 이런 복잡성은 우리가 원하는 바가 아니다.

그냥 5 장이면 5 장, 10 장이면 10 장으로, 적당히 고정된 숫자를 제공해주는 게 차라리 더 낫다.

만약 스테이지의 소탕 가능 횟수가 정해져 있고, 소탕권의 1 회당 연속 사용 횟수보다 적게 남았다면(1 회당 연속 5 회 소탕 가능한데, 남은 소탕 횟수는 2 회라든가...), 자동으로 가능한 남은 소탕 횟수에 맞춰주는 정도면 충분하리라고 본다.

이런 상황에서 연속 소탕을 원하지 않는다면, 그 플레이어는 그냥 1 회씩 원하는 만큼 소탕권을 쓰면 된다.

- **B-24-6-4.** 스테이지 소탕권의 연속적인 사용은 VIP 레벨이 일정 이상이어야 가능하도록 지정할 수 있다.

- **B-24-6-5.** 이 재화를 획득할 수 있는 컨텐츠들은 다음과 같다.

종류	설명
스토리 모드(일반) 완료	• 이 과정으로 획득할 수 없음
스토리 모드(정예) 완료	• 이 과정으로 획득할 수 없음
~~일일 던전 완료~~	~~• 이 과정으로 획득할 수 없음~~
주간 던전 완료	• 이 과정으로 획득할 수 없음
혼돈 던전 완료	• 이 과정으로 획득할 수 없음
초월 던전 완료	• 이 과정으로 획득할 수 없음
숨겨진 던전 완료	• 이 과정으로 획득할 수 없음
결투장 승리	• 이 과정으로 획득할 수 없음
길드 던전 완료	• 이 과정으로 획득할 수 없음
길드 전쟁 승리	• 이 과정으로 획득할 수 없음
월드 보스 협공 보상	• 공헌도에 따라 보상받는 양이 달라진다.
수련장	• 이 과정으로 획득할 수 없음
용병 활동	• 이 과정으로 획득할 수 없음
보유한 재화 판매 (장비, 소모품, 재료 등)	• 이 과정으로 획득할 수 없음
아이템 강화 결과	• 이 과정으로 획득할 수 없음
아이템 제작 결과	• 이 과정으로 획득할 수 없음
일일 미션 달성 보상	• 미션 목표를 달성했을 때, 그 보상으로 주어질 수 있다.
업적 달성 보상	• 업적 목표를 달성했을 때, 그 보상으로 주어질 수 있다.
일일 접속 보상	• 게임에 일정 기간 동안 반복적으로 접속했을 때 보상하는 재화의 종류에 포함할 수 있다.

서비스 운영에 의한 비정기적 지급	・그 외 비정기적인 방식으로 서비스 운영 단계에서 사용자에게 공급하는 경우 ・예를 들면, 장기 미접속자가 오랜만에 다시 접속했을 때, 플레이 유도를 위해 지급하는 경우를 들 수 있다.
VIP 레벨에 의한 지급 방식 변화	・ VIP 레벨을 올렸을 때의 혜택으로 제공할 수 있다. ・게임 플레이 진행 중, 특정한 이벤트가 발생했을 때, 추가적인 양을 획득하게 하거나, 지급하는 양을 지정한 배수만큼 늘려주는 방식 등이다. ・ VIP 레벨에 따라 지급하는 양이 더 많아질 수 있다.
상설 상점 구매	・과금 화폐(큐빅, Cubic)를 이용해 구입할 수 있다. ・교환하는 양에 따라 미리 정해진 상품 단위가 몇 가지 있고, 플레이어는 이 중에서 선택하는 방식이다. ・실제 시간 간격으로 구입할 수 있는 횟수의 제한이 있을 수 있다.
숨겨진 상점 구매	・이 과정으로 획득할 수 없음
무작위 뽑기	・어떤 재화라도 나올 수 있기 때문에, 그 중에 한 종류로써 등장할 수 있다.

- **B-24-6-6.** 스테이지 소탕권은 **스토리 모드의 던전에서만 사용**할 수 있다.
: 던전의 난이도에는 상관 없이 사용할 수 있지만, ~~일일 던전~~, 주간 던전, 혼돈 던전, 초월 던전 등의 특수한 던전 스테이지에서는 사용할 수 없다.

- **B-24-6-7.** 한 번이라도 완료한 스테이지에서만 사용할 수 있다.
: 플레이 결과를 최고 등급을 받지 못했더라도, 일단 스테이지를 완료하기만 했으면 스테이지 소탕권은 문제 없이 사용할 수 있다.

◆ B-24-7. (장비)아이템 뽑기권

- **B-24-7-1.** 아이템 뽑기 상점에서 무작위한 능력과 등급을 가진 아이템을 하나 획득할 수 있는 권한을 주는 티켓 방식의 재화이다.

- **B-24-7-2.** 이 재화를 얻으면, 얻은 개수만큼 무료로 장비 아이템 뽑기를 할 수 있는 횟수가 늘어난다.

- **B-24-7-3.** 뽑기권 자체는 보관 가능한 형태의 아이템으로 등장하지 않는다.
: 수령하는 즉시, 뽑기권을 '사용'한 것으로 간주하여, 아이템 뽑기 상점에서 무료로 뽑을 수 있는 횟수가 늘어나는 방식이다.

- **B-24-7-4.** 장비 아이템 뽑기 1회 당 뽑기 횟수를 1 소모한다.

: 스킬 룬 뽑기 1회에 뽑기 횟수를 여러 개 소모하는 방식은 고려하고 있지 않다.

- **B-24-7-5.** 장비 아이템 뽑기 횟수를 1회 사용하는 GUI 명령과 연속적으로 X회 사용하는 GUI 명령이 공존한다.
: 단, 연속 사용의 횟수 자체는 고정되어 있고, 플레이어가 연속 횟수를 조절하지는 못한다.

- **B-24-7-6.** 이 재화를 획득할 수 있는 컨텐츠들은 다음과 같다.

종류	설명
스토리 모드(일반) 완료	• 이 과정으로 획득할 수 없음 : 아이템으로 직접 획득하기 이다.
스토리 모드(정예) 완료	• 이 과정으로 획득할 수 없음 : 아이템으로 직접 획득하기 때문이다.
~~일일 던전 완료~~	~~• 이 과정으로 획득할 수 없음~~ ~~: 아이템으로 직접 획득하기 때문이다.~~
주간 던전 완료	• 이 과정으로 획득할 수 없음 : 아이템으로 직접 획득하기 때문이다.
혼돈 던전 완료	• 이 과정으로 획득할 수 없음 : 아이템으로 직접 획득하기 때문이다.
초월 던전 완료	• 이 과정으로 획득할 수 없음 : 아이템으로 직접 획득하기 때문이다.
숨겨진 던전 완료	• 이 과정으로 획득할 수 없음 : 아이템으로 직접 획득하기 때문이다.
결투장 승리	• 이 과정으로 획득할 수 없음
길드 던전 완료	• 이 과정으로 획득할 수 없음
길드 전쟁 승리	• 이 과정으로 획득할 수 없음
월드 보스 협공 보상	• 이 과정으로 획득할 수 없음 : 아이템으로 직접 획득하기 때문이다.
수련장	• 이 과정으로 획득할 수 없음
용병 활동	• 이 과정으로 획득할 수 없음
보유한 재화 판매 (장비, 소모품, 재료 등)	• 이 과정으로 획득할 수 없음
아이템 강화 결과	• 이 과정으로 획득할 수 없음
아이템 제작 결과	• 이 과정으로 획득할 수 없음
일일 미션 달성 보상	• 미션 목표를 달성했을 때, 그 보상으로 주어질 수 있다.
업적 달성 보상	• 업적 목표를 달성했을 때, 그 보상으로 주어질 수 있다.
일일 접속 보상	• 게임에 일정 기간 동안 반복적으로 접속했을 때 보상하는 재화의 종류에 포함할 수 있다.

서비스 운영에 의한 비정기적 지급	• 그 외 비정기적인 방식으로 서비스 운영 단계에서 사용자에게 공급하는 경우 • 예를 들면, 장기 미접속자가 오랜만에 다시 접속했을 때, 플레이 유도를 위해 지급하는 경우를 들 수 있다.
VIP 레벨에 의한 지급 방식 변화	• 이 과정으로 획득할 수 없음
상설 상점 구매	• 과금 화폐(큐빅, Cubic)를 이용해 구입할 수 있다. • 명시적인 교환권이 따로 있는 건 아니고, 제작 메뉴에 과금 화폐를 이용한 가격이 매겨져 있다. • 구입 횟수나 재사용 시간에 대한 제한 사항이 없다.
숨겨진 상점 구매	• 이 과정으로 획득할 수 없음
무작위 뽑기	• 어떤 재화라도 나올 수 있기 때문에, 그 중에 한 종류로써 등장할 수 있다.

◆ B-24-8. 스킬 룬 뽑기권

- **B-24-8-1.** 스킬 룬 뽑기 상점에서 무작위한 능력과 등급을 가진 스킬 룬을 하나 획득할 수 있는 권한을 주는 티켓 재화이다.

- **B-24-8-2.** 이 재화를 얻으면, 얻은 개수만큼 무료로 스킬 룬 뽑기를 할 수 있는 횟수가 늘어난다.

- **B-24-8-3.** 뽑기권 자체는 보관 가능한 형태의 아이템으로 등장하지 않는다.
  : 수령하는 즉시, 뽑기권을 '사용'한 것으로 간주하여, 아이템 뽑기 상점에서 무료로 뽑을 수 있는 횟수가 늘어나는 방식이다.

- **B-24-8-4.** 스킬 룬 뽑기 1회 당 뽑기 횟수를 1 소모한다.
  : 스킬 룬 뽑기 1회에 뽑기 횟수를 여러 개 소모하는 방식은 고려하고 있지 않다.

- **B-24-8-5.** 스킬 룬 뽑기 횟수를 1회 사용하는 GUI 명령과 연속적으로 X회 사용하는 GUI 명령이 공존한다.
  : 단, 연속 사용의 횟수 자체는 고정되어 있고, 플레이어가 연속 횟수를 조절하지는 못한다.

- **B-24-8-6.** 이 재화를 획득할 수 있는 컨텐츠들은 다음과 같다.

종류	설명
스토리 모드(일반) 완료	• 이 과정으로 획득할 수 없음 : 스킬 룬으로 직접 획득하기 때문이다.

스토리 모드(정예) 완료	• 이 과정으로 획득할 수 없음 : 스킬 룬으로 직접 획득하기 때문이다.
~~일일 던전 완료~~	~~• 이 과정으로 획득할 수 없음~~ ~~: 스킬 룬으로 직접 획득하기 때문이다.~~
주간 던전 완료	• 이 과정으로 획득할 수 없음 : 스킬 룬으로 직접 획득하기 때문이다.
혼돈 던전 완료	• 이 과정으로 획득할 수 없음 : 스킬 룬으로 직접 획득하기 때문이다.
초월 던전 완료	• 이 과정으로 획득할 수 없음 : 스킬 룬으로 직접 획득하기 때문이다.
숨겨진 던전 완료	• 이 과정으로 획득할 수 없음 : 스킬 룬으로 직접 획득하기 때문이다.
결투장 승리	• 이 과정으로 획득할 수 없음
길드 던전 완료	• 이 과정으로 획득할 수 없음
길드 전쟁 승리	• 이 과정으로 획득할 수 없음
월드 보스 협공 보상	• 이 과정으로 획득할 수 없음 : 스킬 룬으로 직접 획득하기 때문이다.
수련장	• 이 과정으로 획득할 수 없음
용병 활동	• 용병 활동의 성과로 지급받는다.
보유한 재화 판매 (장비, 소모품, 재료 등)	• 이 과정으로 획득할 수 없음
아이템 강화 결과	• 이 과정으로 획득할 수 없음
아이템 제작 결과	• 이 과정으로 획득할 수 없음
일일 미션 달성 보상	• 미션 목표를 달성했을 때, 그 보상으로 주어질 수 있다.
업적 달성 보상	• 업적 목표를 달성했을 때, 그 보상으로 주어질 수 있다.
일일 접속 보상	• 게임에 일정 기간 동안 반복적으로 접속했을 때 보상하는 재화의 종류에 포함할 수 있다.
서비스 운영에 의한 비정기적 지급	• 그 외 비정기적인 방식으로 서비스 운영 단계에서 사용자에게 공급하는 경우 • 예를 들면, 장기 미접속자가 오랜만에 다시 접속했을 때, 플레이 유도를 위해 지급하는 경우를 들 수 있다.
VIP 레벨에 의한 지급 방식 변화	• VIP 레벨을 올렸을 때의 혜택으로 제공할 수 있다. • 게임 플레이 진행 중, 특정한 이벤트가 발생했을 때, 추가적인 양을 획득하게 하거나, 지급하는 양을 지정한 배수만큼 늘려주는 방식 등이다. • VIP 레벨에 따라 지급하는 양이 더 많아질 수 있다.
상설 상점 구매	• 과금 화폐(큐빅, Cubic)를 이용해 구입할 수 있다.

	・명시적인 교환권이 따로 있는 건 아니고, 제작 메뉴에 과금 화폐를 이용한 가격이 매겨져 있다. ・실제 시간 간격으로 구입할 수 있는 횟수의 제한이 있을 수 있다.
숨겨진 상점 구매	・이 과정으로 획득할 수 없음
무작위 뽑기	・어떤 재화라도 나올 수 있기 때문에, 그 중에 한 종류로써 등장할 수 있다.

◆ B-24-9. 장착 아이템 강화석

- **B-24-9-1.** 아이템의 강화 점수를 올려서, 강화 단계를 더 높이는데 사용한다.

※ 아이템 강화 항목의 명세에서도 설명하겠지만, 장착 아이템의 강화석을 사용하지 않더라도, 다른 장착 가능한 아이템들과 게임 플레이 화폐를 더해서 아이템의 강화 점수를 올릴 수 있다. 장착 아이템 강화석도 같은 역할을 하는데, 오직 장착 가능 아이템의 강화 점수를 올리기 위한 용도로만 사용하기 위해서 고안된 재화이다.

- **B-24-9-2.** 1회 사용시 강화 점수를 얼마나 높일 수 있는지에 따라, 종류는 여러 가지가 존재한다.

종류	가격(예시)	설명
일반	10	・사용 대상이 된 아이템 아이템 강화 점수를 조금 올려준다.
고급	30	・사용 대상이 된 아이템 아이템 강화 점수를 많이 올려준다.
최고급	100	・사용 대상이 된 아이템 아이템 강화 점수를 대량으로 올려준다.

- **B-24-9-3.** 모든 아이템 강화석들은 사용할 때 1개씩 소모된다.
: 즉, 아이템 강화석들은 소모성 아이템들이다.

- **B-24-9-4.** 아이템 강화석은 보관함 슬롯 한 개에 최대 999개까지 축적할 수 있다.

- **B-24-9-5.** 이 재화를 획득할 수 있는 컨텐츠들은 다음과 같다.

종류	설명
스토리 모드(일반) 완료	・이 과정으로 획득할 수 없음
스토리 모드(정예) 완료	・이 과정으로 획득할 수 없음
~~일일 던전 완료~~	~~・이 과정으로 획득할 수 없음~~
주간 던전 완료	・이 과정으로 획득할 수 없음
혼돈 던전 완료	・획득 가능

초월 던전 완료	• 이 과정으로 획득할 수 없음
숨겨진 던전 완료	• 이 과정으로 획득할 수 없음
결투장 승리	• 이 과정으로 획득할 수 없음
길드 던전 완료	• 이 과정으로 획득할 수 없음
길드 전쟁 승리	• 이 과정으로 획득할 수 없음
월드 보스 협공 보상	• 획득 가능
수련장	• 이 과정으로 획득할 수 없음
용병 활동	• 이 과정으로 획득할 수 없음
보유한 재화 판매 (장비, 소모품, 재료 등)	• 이 과정으로 획득할 수 없음
아이템 강화 결과	• 이 과정으로 획득할 수 없음
아이템 제작 결과	• 이 과정으로 획득할 수 없음
일일 미션 달성 보상	• 미션 목표를 달성했을 때, 그 보상으로 주어질 수 있다.
업적 달성 보상	• 업적 목표를 달성했을 때, 그 보상으로 주어질 수 있다.
일일 접속 보상	• 게임에 일정 기간 동안 반복적으로 접속했을 때 보상하는 재화의 종류에 포함할 수 있다.
서비스 운영에 의한 비정기적 지급	• 이 과정으로 획득할 수 없음
VIP 레벨에 의한 지급 방식 변화	• 이 과정으로 획득할 수 없음
상설 상점 구매	• 이 과정으로 획득할 수 없음
숨겨진 상점 구매	• 구매 가능한 재화의 종류 중 하나로 등장할 수 있다.
무작위 뽑기	• 어떤 재화라도 나올 수 있기 때문에, 그 중에 한 종류로써 등장할 수 있다.

◆ B-24-10. 장착 아이템 초월석

- **B-24-10-1.** 아이템을 강화할 수 있는 최대 강화 레벨을 더 높이는 데 사용한다.
  : 1 회 사용시 강화 레벨을 몇 단계까지 높일 수 있는지에 따라, 종류는 여러 가지가 존재한다.

※ 실제로는 초월석만으로 장착 아이템의 최대 강화 레벨을 높일 수는 없다.
최대 강화 레벨을 높이는 목적을 달성하기 위해 구해야 하는 '상대적으로 더 어려운 재료'이면서, 실제로 최대 강화 레벨을 얼마까지 높여줄 것인지 결정하는 재료 요소이다.
실제 아이템의 최대 레벨 강화에는 초월석 말고도 더 많은 재료 아이템들과 재화들이 소모된다. 최대 레벨을 더 많이 높일 수 있는 초월석일수록 더욱 그렇다.

※ 혹시라도 혼동할까 적어두자면, 아이템 강화석은 아이템의 현재까지 쌓인 강화 수치를 높이는데 쓰이고, 아이템 초월석은 그 아이템을 강화할 수 있는 레벨의 최대 한계치를 늘리는 데 쓰인다.
즉, 어떤 장검 아이템의 강화 레벨이 현재 3 이고, 최대 10 까지 강화할 수 있다고 하자,(3 / 10)
아이템 강화석은 현재 레벨이 3 레벨의 강화 수치 부분을 높인다.
아이템 초월석은 아이템의 최대 강화 레벨인 10 레벨의 수치 부분을 더 늘린다.

- **B-24-10-2.** 이 재화를 획득할 수 있는 컨텐츠들은 다음과 같다.

종류	설명
스토리 모드(일반) 완료	• 이 과정으로 획득할 수 없음
스토리 모드(정예) 완료	• 이 과정으로 획득할 수 없음
~~일일 던전 완료~~	~~• 이 과정으로 획득할 수 없음~~
주간 던전 완료	• 이 과정으로 획득할 수 없음
혼돈 던전 완료	• 획득 가능
초월 던전 완료	• 이 과정으로 획득할 수 없음
숨겨진 던전 완료	• 이 과정으로 획득할 수 없음
결투장 승리	• 이 과정으로 획득할 수 없음
길드 던전 완료	• 이 과정으로 획득할 수 없음
길드 전쟁 승리	• 이 과정으로 획득할 수 없음
월드 보스 협공 보상	• 획득 가능
수련장	• 이 과정으로 획득할 수 없음
용병 활동	• 이 과정으로 획득할 수 없음
보유한 재화 판매 (장비, 소모품, 재료 등)	• 이 과정으로 획득할 수 없음
아이템 강화 결과	• 이 과정으로 획득할 수 없음
아이템 제작 결과	• 이 과정으로 획득할 수 없음
일일 미션 달성 보상	• 미션 목표를 달성했을 때, 그 보상으로 주어질 수 있다.
업적 달성 보상	• 업적 목표를 달성했을 때, 그 보상으로 주어질 수 있다.
일일 접속 보상	• 게임에 일정 기간 동안 반복적으로 접속했을 때 보상하는 재화의 종류에 포함할 수 있다.
서비스 운영에 의한 비정기적 지급	• 이 과정으로 획득할 수 없음
VIP 레벨에 의한 지급 방식 변화	• 이 과정으로 획득할 수 없음
상설 상점 구매	• 이 과정으로 획득할 수 없음
숨겨진 상점 구매	• 구매 가능한 재화의 종류 중 하나로 등장할 수 있다.

무작위 뽑기	• 어떤 재화라도 나올 수 있기 때문에, 그 중에 한 종류로써 등장할 수 있다.

◆ B-24-11. 스킬 강화석

- **B-24-11-1.** 강화하려고 하는 스킬의 숙련도 수치를 높여서, 스킬의 숙련 레벨을 더 높이는 데 사용한다.
  : 1회 사용시 숙련도 수치를 얼마나 높일 수 있는지에 따라, 종류는 여러 가지가 존재한다.

※ 관련 명세 항목에서 더 자세하게 설명하겠지만, 스킬 강화석을 사용하지 않더라도, 스킬 룬과 게임 플레이 화폐를 더해서 스킬의 숙련 점수를 올릴 수 있다. 그 밖에도, 수련장을 통해서도 스킬의 숙련 점수를 올릴 수 있다.
스킬 강화석도 같은 역할을 하는데, 오직 스킬의 숙련 점수를 올리기 위한 용도로만 사용하기 위해서 고안된 재화이다.

- **B-24-11-2.** 이 재화를 획득할 수 있는 컨텐츠들은 다음과 같다.

종류	설명
스토리 모드(일반) 완료	• 이 과정으로 획득할 수 없음
스토리 모드(정예) 완료	• 이 과정으로 획득할 수 없음
~~일일 던전 완료~~	~~• 이 과정으로 획득할 수 없음~~
주간 던전 완료	• 이 과정으로 획득할 수 없음
혼돈 던전 완료	• 획득 가능
초월 던전 완료	• 이 과정으로 획득할 수 없음
숨겨진 던전 완료	• 이 과정으로 획득할 수 없음
결투장 승리	• 이 과정으로 획득할 수 없음
길드 던전 완료	• 획득 가능
길드 전쟁 승리	• 이 과정으로 획득할 수 없음
월드 보스 협공 보상	• 획득 가능
수련장	• 이 과정으로 획득할 수 없음
용병 활동	• 이 과정으로 획득할 수 없음
보유한 재화 판매 (장비, 소모품, 재료 등)	• 이 과정으로 획득할 수 없음
아이템 강화 결과	• 이 과정으로 획득할 수 없음
아이템 제작 결과	• 이 과정으로 획득할 수 없음
일일 미션 달성 보상	• 미션 목표를 달성했을 때, 그 보상으로 주어질 수 있다.
업적 달성 보상	• 업적 목표를 달성했을 때, 그 보상으로 주어질 수 있다.

일일 접속 보상	・게임에 일정 기간 동안 반복적으로 접속했을 때 보상하는 재화의 종류에 포함할 수 있다.
서비스 운영에 의한 비정기적 지급	・이 과정으로 획득할 수 없음
VIP 레벨에 의한 지급 방식 변화	・이 과정으로 획득할 수 없음
상설 상점 구매	・과금 화폐(큐빅, Cubic)를 이용해 구입할 수 있다. ・명시적인 교환권이 따로 있는 건 아니고, 제작 메뉴에 과금 화폐를 이용한 가격이 매겨져 있다. ・실제 시간 간격으로 구입할 수 있는 횟수의 제한이 있을 수 있다.
숨겨진 상점 구매	・구매 가능한 재화의 종류 중 하나로 등장할 수 있다.
무작위 뽑기	・어떤 재화라도 나올 수 있기 때문에, 그 중에 한 종류로써 등장할 수 있다.

◆ B-24-12. (장착) 아이템 제작 재료

- **B-24-12-1.** 아이템의 등급의 향상을 위한 비용으로 사용하는 재화 중 하나이다.

※ 아이템의 강화와 아이템의 승급은 다른 개념이다.
아이템을 강화할 때는, 그 아이템의 등급을 바꾸지 않고, 강화 레벨만 올리는 것을 말한다.
아이템의 합성은, 그 아이템의 등급 자체를 한 단계 격상시키는 것이다. 이 과정에서 아이템의 모든 능력 옵션들은 무작위로 재부여한다. 또한, 현재까지의 강화 수치에 의해 강화 레벨이 다시 결정된다. (등급이 높은 아이템일수록 강화 레벨을 올리기 위해 필요한 강화 수치가 더 많이 필요하다.)

- **B-24-12-2.** 합성 재료 아이템들끼리는 합성하거나 승급하는 과정이 없다.
: 승급할 단계가 높을수록 더 많은 재료가 필요할 뿐이고, 더 '상위 단계의' 재료를 필요로 하는 일은 없다.

- **B-24-12-3.** 이 재화를 획득할 수 있는 컨텐츠들은 다음과 같다.

종류	설명
스토리 모드(일반) 완료	・이 과정으로 획득할 수 없음
스토리 모드(정예) 완료	・이 과정으로 획득할 수 없음
~~일일 던전 완료~~	~~・획득 가능~~
주간 던전 완료	・이 과정으로 획득할 수 없음
혼돈 던전 완료	・전설 아이템 승급에 필요한 특수 재료인 '악의 정수'를 획득할 수

	있다. ~~・나머지 재료들을 일일 던전을 통해서 획득해야 한다.~~
초월 던전 완료	・이 과정으로 획득할 수 없음
숨겨진 던전 완료	・이 과정으로 획득할 수 없음
결투장 승리	・이 과정으로 획득할 수 없음
길드 던전 완료	・획득 가능
길드 전쟁 승리	・이 과정으로 획득할 수 없음
월드 보스 협공 보상	・획득 가능
수련장	・이 과정으로 획득할 수 없음
용병 활동	・이 과정으로 획득할 수 없음
보유한 재화 판매 (장비, 소모품, 재료 등)	・이 과정으로 획득할 수 없음
아이템 강화 결과	・이 과정으로 획득할 수 없음
아이템 제작 결과	・이 과정으로 획득할 수 없음
일일 미션 달성 보상	・미션 목표를 달성했을 때, 그 보상으로 주어질 수 있다.
업적 달성 보상	・업적 목표를 달성했을 때, 그 보상으로 주어질 수 있다.
일일 접속 보상	・게임에 일정 기간 동안 반복적으로 접속했을 때 보상하는 재화의 종류에 포함할 수 있다.
서비스 운영에 의한 비정기적 지급	・그 외 비정기적인 방식으로 서비스 운영 단계에서 사용자에게 공급하는 경우 ・예를 들면, 장기 미접속자가 오랜만에 다시 접속했을 때, 플레이 유도를 위해 지급하는 경우를 들 수 있다.
VIP 레벨에 의한 지급 방식 변화	・ VIP 레벨을 올렸을 때의 혜택으로 제공할 수 있다. ・게임 플레이 진행 중, 특정한 이벤트가 발생했을 때, 추가적인 양을 획득하게 하거나, 지급하는 양을 지정한 배수만큼 늘려주는 방식 등이다. ・ VIP 레벨에 따라 지급하는 양이 더 많아질 수 있다.
상설 상점 구매	・이 과정으로 획득할 수 없음
숨겨진 상점 구매	・구매 가능한 재화의 종류 중 하나로 등장할 수 있다.
무작위 뽑기	・어떤 재화라도 나올 수 있기 때문에, 그 중에 한 종류로써 등장할 수 있다.

◆ B-24-13. 아이템 강화 보석

- **B-24-13-1.** 장비하는 아이템들(Equipment Item)은 능력 옵션에 따라, 1개의 보석 소켓을 가질 수 있다.

※ 아이템이 1 개를 초과하는 개수의 보석 소켓을 가질 수 있는지는 고민 중이기는 하지만, 일단은 아이템마다 최대 1 개로 고정하기로 했다.
이유는, 보석의 능력으로 인한 밸런스 붕괴, 모바일 환경에서의 GUI 등을 고려해서이다.

- **B-24-13-2.** **일반 보석**
: 종류에 따라 **장착 아이템 1개, 장신구 1개로 총 2가지 장착 부위에 장착**할 수 있는 보석이다.
보석을 끼운 아이템에 특정한 능력 수치를 더해준다.

- **B-24-13-3.** **(스킬)발동 보석**
: 장착 부위 중, **무기 부위에 들어갈 아이템에만 끼울 수 있는 보석이다.**
아이템을 사용할 때, 보석의 능력으로 지정되어 있는 스킬을 발동시킬 수 있다.

※ 스킬을 발동시키기 때문에, 캐릭터마다 1 개를 초과해서 허용하지 않도록, 무기에만 장착 가능하도록 제한한 것이다.
이런 보석을 여러 개 장비할 수 있다면, 경우에 따라 심각하게 게임의 밸런스를 무너뜨릴 수 있다.

- **B-24-13-4.** 일반 보석들은 다음의 종류와 특징을 가지고 있다.

종류	장착 / 강화 가능한 부위	설명
에메랄드	무기, 반지	• 공격력을 늘려준다.
자수정	몸통, 목걸이	• 방어력을 늘려준다.
루비	투구, 팔찌	• 생명력을 늘려준다.
다이아몬드	캐릭터 별 특수 파츠, 귀걸이	• 마력을 늘려준다.

- **B-24-13-5.** 발동 보석은 10개의 종류가 있으며, 각각 10가지의 다른 스킬 능력들을 하나씩 가지고 있다.

- **B-24-13-6.** 일반 보석들은 각자 등급을 가지고 있다.
: 등급은 총 X 단계까지 있고, 높은 단계로 갈수록 획득할 수 있는 확률이 더 낮고, 더 어려운 스테이지에서 등장한다.

- **B-24-13-7.** 일반 아이템 강화 보석은, 아이템 제작(아이템의 등급을 상향함)의 재료 중 하나로 사용하는 용도도 있다.
: 아이템 제작에 보석을 사용할 때는, 장착할 수 있는 부위의 장비 아이템에 사용해야 한다.

※ 무기와 반지의 강화에는 에메랄드가 필요하고, 투구와 팔찌에는 루비가 필요하다.

- **B-24-13-8.** 이 재화를 획득할 수 있는 컨텐츠들은 다음과 같다.

종류	설명
스토리 모드(일반) 완료	• 이 과정으로 획득할 수 없음
스토리 모드(정예) 완료	• 이 과정으로 획득할 수 없음
~~일일 던전 완료~~	~~• 획득 가능~~
주간 던전 완료	• 이 과정으로 획득할 수 없음
혼돈 던전 완료	• 이 과정으로 획득할 수 없음
초월 던전 완료	• 이 과정으로 획득할 수 없음
숨겨진 던전 완료	• 이 과정으로 획득할 수 없음
결투장 승리	• 이 과정으로 획득할 수 없음
길드 던전 완료	• 획득 가능
길드 전쟁 승리	• 이 과정으로 획득할 수 없음
월드 보스 협공 보상	• 획득 가능
수련장	• 이 과정으로 획득할 수 없음
용병 활동	• 이 과정으로 획득할 수 없음
보유한 재화 판매 (장비, 소모품, 재료 등)	• 이 과정으로 획득할 수 없음
아이템 강화 결과	• 이 과정으로 획득할 수 없음
아이템 제작 결과	• 하위 보석끼리 합성한 결과로 상위 보석을 얻을 수 있다. : 물론, 합성 조건을 만족해야 한다.
일일 미션 달성 보상	• 미션 목표를 달성했을 때, 그 보상으로 주어질 수 있다.
업적 달성 보상	• 업적 목표를 달성했을 때, 그 보상으로 주어질 수 있다.
일일 접속 보상	• 게임에 일정 기간 동안 반복적으로 접속했을 때 보상하는 재화의 종류에 포함할 수 있다.
서비스 운영에 의한 비정기적 지급	• 그 외 비정기적인 방식으로 서비스 운영 단계에서 사용자에게 공급하는 경우 • 예를 들면, 장기 미접속자가 오랜만에 다시 접속했을 때, 플레이 유도를 위해 지급하는 경우를 들 수 있다.
VIP 레벨에 의한 지급 방식 변화	• VIP 레벨을 올렸을 때의 혜택으로 제공할 수 있다. • 게임 플레이 진행 중, 특정한 이벤트가 발생했을 때, 추가적인 양을 획득하게 하거나, 지급하는 양을 지정한 배수만큼 늘려주는 방식 등이다. • VIP 레벨에 따라 지급하는 양이 더 많아질 수 있다.
상설 상점 구매	• 이 과정으로 획득할 수 없음
숨겨진 상점 구매	• 구매 가능한 재화의 종류 중 하나로 등장할 수 있다.
무작위 뽑기	• 어떤 재화라도 나올 수 있기 때문에, 그 중에 한 종류로써 등장할 수 있다.

◆ B-24-14. 아이템 형상 변환권

- **B-24-14-1.** 아이템의 외형을 무료로 변경할 수 있는 횟수를 늘려주는 재화이다.

- **B-24-14-2.** 형상을 변환하더라도, 아이템의 능력에는 전혀 변화가 없다.
: 오직, 겉으로 보이는 모델링, 이펙트 등만 바뀔 뿐이다. **성능은 형상을 변화하기 이전과 다름없이 똑같다.**

※ 이 게임에서는 캐릭터의 레벨이 낮을 때 사용하는 아이템이라고 해서 생김새가 더 단순하고 소박하게 생기거나, 높은 레벨의 캐릭터가 사용하는 아이템이라고 해서 생김새가 더 화려하고 웅장한 방식을 사용하지 않는다.
즉, **아이템의 외형에는 어떤 내적이거나 외적인 등급도 없다.**
고레벨 지역에서 등장한 아이템일지라도, 자신은 그 아이템의 저레벨 지역 모델링이 더 마음에 든다면, 그렇게 바꿔서 쓰면 되는 개념이다.

- **B-24-14-3.** 이 재화를 얻으면, 얻은 수만큼 무료로 아이템의 형상 변환을 할 수 있는 횟수가 늘어난다.

- **B-24-14-4.** 아이템 형상 변환권 자체는 보관 가능한 형태의 아이템으로 등장하지 않는다.
: 수령하는 즉시, 뽑기권을 '사용'한 것으로 간주하여, 아이템 형상 변환이 가능한 상점에서 무료로 형상 변환을 할 수 있는 횟수가 늘어나는 방식이다.

- **B-24-14-5.** 아이템의 형상 변환 시도 1회 당 아이템 형상 변환 가능 횟수를 1 소모한다.
: 아이템의 형상 변환 시도 1회에 가능 횟수를 여러 개 소모하는 방식은 고려하고 있지 않다.

- **B-24-14-6.** 아이템의 변상 변환을 1회 사용하는 GUI 명령과 연속적으로 X회 사용하는 GUI 명령이 공존한다.
: 단, 연속 사용의 횟수 자체는 고정되어 있고, 플레이어가 연속 횟수를 조절하지는 못한다.

- **B-24-14-7.** 이 재화를 획득할 수 있는 컨텐츠들은 다음과 같다.

종류	설명
스토리 모드(일반) 완료	• 이 과정으로 획득할 수 없음
스토리 모드(정예) 완료	• 이 과정으로 획득할 수 없음
~~일일 던전 완료~~	~~• 이 과정으로 획득할 수 없음~~
주간 던전 완료	• 이 과정으로 획득할 수 없음
혼돈 던전 완료	• 이 과정으로 획득할 수 없음
초월 던전 완료	• 이 과정으로 획득할 수 없음

숨겨진 던전 완료	• 이 과정으로 획득할 수 없음
결투장 승리	• 이 과정으로 획득할 수 없음
길드 던전 완료	• 이 과정으로 획득할 수 없음
길드 전쟁 승리	• 이 과정으로 획득할 수 없음
월드 보스 협공 보상	• 이 과정으로 획득할 수 없음
수련장	• 이 과정으로 획득할 수 없음
용병 활동	• 이 과정으로 획득할 수 없음
보유한 재화 판매 (장비, 소모품, 재료 등)	• 이 과정으로 획득할 수 없음
아이템 강화 결과	• 이 과정으로 획득할 수 없음
아이템 제작 결과	• 이 과정으로 획득할 수 없음
일일 미션 달성 보상	• 이 과정으로 획득할 수 없음
업적 달성 보상	• 업적 목표를 달성했을 때, 그 보상으로 주어질 수 있다.
일일 접속 보상	• 이 과정으로 획득할 수 없음
서비스 운영에 의한 비정기적 지급	• 이 과정으로 획득할 수 없음 : 플레이 장려와는 크게 관계가 없는 사항이라서 그렇다.
VIP 레벨에 의한 지급 방식 변화	• 이 과정으로 획득할 수 없음 : 플레이 편의를 더해주는 것과 별 관계가 없는 사항이다.
상설 상점 구매	• 과금 화폐(큐빅, Cubic)를 이용해 구입할 수 있다.
숨겨진 상점 구매	• 구매 가능한 재화의 종류 중 하나로 등장할 수 있다.
무작위 뽑기	• 어떤 재화라도 나올 수 있기 때문에, 그 중에 한 종류로써 등장할 수 있다.

◆ B-24-15. (장비)아이템 마법 부여석

- **B-24-15-1.** 마법 부여 상점에서, 비용을 내지 않고 무료로 캐릭터가 가지고 있는 장비 아이템의 능력을 다른 것으로 교체할 수 있는 권한을 주는 티켓 방식의 재화이다.

- **B-24-15-2.** 이 재화를 얻으면, 얻은 수만큼 무료로, 선택한 장비 아이템에 마법 부여를 할 수 있는 횟수가 늘어난다.

- **B-24-15-3.** 이 재화 자체는 보관 가능한 형태의 아이템으로 등장하지 않는다.
: 수령하는 즉시, 마법 부여석들은 '사용'한 것으로 간주하여 사라지고, 아이템 마법 부여 상점에서 해당하는 마법 부여를 무료로 할 수 있는 횟수가 그만큼 늘어난다.

- **B-24-15-4.** 마법 부여를 1회 수행할 때마다, 마법 부여석으로 획득한 수행 횟수를 1회씩 소모한다.

: 1회의 마법 부여마다 마법 부여석에 의한 수행 횟수를 여러 개 소모하는 방식은 고려하고 있지 않다.

- **B-24-15-5.** 마법 부여석의 종류는 다음과 같다.

종류	설명
무작위 마법 부여석	• 마법 부여 상점에서, 선택한 장비 아이템의 능력 옵션 전체를 무작위로 바꾸는 마법 부여를 무료로 할 수 있는 횟수를 1 회 늘려준다.
선택적 마법 부여석	• 마법 부여 상점에서, 선택한 장비 아이템의 능력 옵션 중 선택한 1 개의 옵션을 무작위로 바꾸는 마법 부여를 무료로 할 수 있는 횟수를 1 회 늘려준다.

- **B-24-15-6.** 이 재화를 획득할 수 있는 컨텐츠들은 다음과 같다.

종류	설명
스토리 모드(일반) 완료	• 이 과정으로 획득할 수 없음
스토리 모드(정예) 완료	• 이 과정으로 획득할 수 없음
~~일일 던전 완료~~	~~• 이 과정으로 획득할 수 없음~~
주간 던전 완료	• 이 과정으로 획득할 수 없음
혼돈 던전 완료	• 이 과정으로 획득할 수 없음
초월 던전 완료	• 이 과정으로 획득할 수 없음
숨겨진 던전 완료	• 이 과정으로 획득할 수 없음
결투장 승리	• 이 과정으로 획득할 수 없음
길드 던전 완료	• 이 과정으로 획득할 수 없음
길드 전쟁 승리	• 이 과정으로 획득할 수 없음
월드 보스 협공 보상	• 이 과정으로 획득할 수 없음
수련장	• 이 과정으로 획득할 수 없음
용병 활동	• 이 과정으로 획득할 수 없음
보유한 재화 판매 (장비, 소모품, 재료 등)	• 이 과정으로 획득할 수 없음
아이템 강화 결과	• 이 과정으로 획득할 수 없음
아이템 제작 결과	• 이 과정으로 획득할 수 없음
일일 미션 달성 보상	• 이 과정으로 획득할 수 없음
업적 달성 보상	• 업적 목표를 달성했을 때, 그 보상으로 주어질 수 있다.
일일 접속 보상	• 이 과정으로 획득할 수 없음
서비스 운영에 의한 비정기적 지급	• 이 과정으로 획득할 수 없음 : 플레이 장려와는 크게 관계가 없는 사항이라서 그렇다.

VIP 레벨에 의한 지급 방식 변화	• 이 과정으로 획득할 수 없음 : 플레이 편의를 더해주는 것과 별 관계가 없는 사항이다.
상설 상점 구매	• 과금 화폐(큐빅, Cubic)를 이용해 구입할 수 있다.
숨겨진 상점 구매	• 구매 가능한 재화의 종류 중 하나로 등장할 수 있다.
무작위 뽑기	• 어떤 재화라도 나올 수 있기 때문에, 그 중에 한 종류로써 등장할 수 있다.

◆ B-24-16. 아이템 감정서

- B-24-16-1. 희귀 등급 이상의 아이템은 감정서를 통해서 감정해야 그 아이템의 완전한 능력을 사용할 수 있다.

※ 아이템을 감정하기 전에는 '감정되지 않은' 상태로 표현하며, 그 아이템의 일반 등급 버전과 같은 능력을 가진다.

- B-24-16-2. 아이템 감정서의 종류는 다음과 같다.

종류	가격(예시)	설명
일반	10	• 희귀 등급의 아이템만을 감정할 수 있다. • 상대적으로 흔하다.
고급	50	• 전설 등급의 아이템만을 감정할 수 있다. • 사용자의 실수를 막기 위해, 희귀 등급의 아이템을 감정할 수 없게 한다. • 일반적으로 판매하지 않으며, 일반 아이템 감정서와 과금 화폐를 이용해 합성으로 '제작'할 수는 있다.

- B-24-16-3. 이 재화를 획득할 수 있는 컨텐츠들은 다음과 같다.

종류	설명
스토리 모드(일반) 완료	• 획득 가능
스토리 모드(정예) 완료	• 획득 가능
일일 던전 완료	• 획득 가능
주간 던전 완료	• 획득 가능
혼돈 던전 완료	• 획득 가능
초월 던전 완료	• 획득 가능
숨겨진 던전 완료	• 획득 가능
결투장 승리	• 이 과정으로 획득할 수 없음

길드 던전 완료	• 이 과정으로 획득할 수 없음
길드 전쟁 승리	• 이 과정으로 획득할 수 없음
월드 보스 협공 보상	• 획득 가능
수련장	• 이 과정으로 획득할 수 없음
용병 활동	• 이 과정으로 획득할 수 없음
보유한 재화 판매 (장비, 소모품, 재료 등)	• 이 과정으로 획득할 수 없음
아이템 강화 결과	• 이 과정으로 획득할 수 없음
아이템 제작 결과	• 일반 감정서는 이 과정으로 획득할 수 없다. • 고급 감정서는 일반 감정서와 과금 화폐를 합성해서 획득할 수 있다.
일일 미션 달성 보상	• 일반 감정서는 이 과정으로 획득 가능하다.
업적 달성 보상	• 업적 목표를 달성했을 때, 그 보상으로 주어질 수 있다.
일일 접속 보상	• 게임에 일정 기간 동안 반복적으로 접속했을 때 보상하는 재화의 종류에 포함할 수 있다.
서비스 운영에 의한 비정기적 지급	• 그 외 비정기적인 방식으로 서비스 운영 단계에서 사용자에게 공급하는 경우 • 예를 들면, 장기 미접속자가 오랜만에 다시 접속했을 때, 플레이 유도를 위해 지급하는 경우를 들 수 있다.
VIP 레벨에 의한 지급 방식 변화	• VIP 레벨을 올렸을 때의 혜택으로 제공할 수 있다. • 매일 일정량 지급하는 방식 • VIP 레벨에 따라 지급하는 양이 더 많아질 수 있다.
상설 상점 구매	• 과금 화폐(큐빅, Cubic)를 이용해 구입할 수 있다. • 명시적인 교환권이 따로 있는 건 아니고, 제작 메뉴에 과금 화폐를 이용한 가격이 매겨져 있다. • 실제 시간 간격으로 구입할 수 있는 횟수의 제한이 있을 수 있다.
숨겨진 상점 구매	• 구매 가능한 재화의 종류 중 하나로 등장할 수 있다.
무작위 뽑기	• 어떤 재화라도 나올 수 있기 때문에, 그 중에 한 종류로써 등장할 수 있다.

※ 수정(2015. 03. 16 조수운)

아이템 감정서의 개념을 제거하기로 한다.

이렇게 결정한 이유는, 식별되지 않은 아이템의 개념과, 아이템 제작(등급 상승)의 개념이 충돌하는 면이 있기 때문이다.

기존 설정 상으로는, 희귀 등급부터 아이템 감정서를 사용하도록 되어 있었다. 그런데 만약 마법 등급의 아이템을 희귀 등급으로 등급을 올렸을 경우(즉, 제작했을 경우), 희귀 등급으로 상승한 아이템은 식별되지 않은 상태여야 할까, 아니면 식별된 상태여야 할까?

두 경우에 다 문제가 있다.

전자의 경우, 아이템 제작, 혹은 뽑기를 할 때마다 희귀 등급 이상의 아이템이 최초로 가방에

생성되는 모든 순간마다 식별되지 않은 상태의 희귀 등급 이상의 아이템이 등장하는 점이 문제다. 심지어, 마법 등급 상태일 때는 '식별되어 있었던' 아이템이 희귀 등급이 되자마자 식별되지 않은 상태가 되어야 한다는 점도 문제다.

후자의 경우, 웬만한 상황에서 아이템 식별이 다 되어 있는 상황으로 희귀 등급 이상의 아이템을 제공한다면, 아이템 식별을 할 상황 자체가 별로 없어지게 되고, 아이템 감정서가 딱히 쓸데가 많이 없다. 그냥 없는 거나 그게 그것인 상황이 만들어진다.

아이템 뽑기를 할 때는 어떤가? 아이템 뽑기는 등급이 어떤 게 등장할지 사전에 정해지지 않는다. 만약 희귀 등급 이상의 아이템이 뽑기로 등장했으면, 그 아이템은 식별되지 않은 상태로 등장해야 하나? 하지만, 그보다 하위 등급 아이템은 항상 '식별된' 상태로 등장한다.

이런 애매한 상황들이 많이 생기기 때문에, 아이템 감정서 개념 자체를 삭제하는 게 더 좋겠다고 판단했다.

◆ B-24-17. 초월 모드 코인(이름을 못 정했다...)

- **B-24-17-1.** 초월 모드를 플레이 하면 획득할 수 있는 화폐이다.
  : 초월 모드 완료 점수와는 별도로 주어진다.

- **B-24-17-2.** 이 재화를 획득할 수 있는 컨텐츠들은 다음과 같다.

종류	설명
스토리 모드(일반) 완료	· 이 과정으로 획득할 수 없음
스토리 모드(정예) 완료	· 이 과정으로 획득할 수 없음
~~일일 던전 완료~~	~~· 이 과정으로 획득할 수 없음~~
주간 던전 완료	· 이 과정으로 획득할 수 없음
혼돈 던전 완료	· 이 과정으로 획득할 수 없음
초월 던전 완료	· 이 과정으로 획득할 수 없음
숨겨진 던전 완료	· 이 과정으로 획득할 수 없음
결투장 승리	· 획득 가능
길드 던전 완료	· 이 과정으로 획득할 수 없음
길드 전쟁 승리	· 이 과정으로 획득할 수 없음
월드 보스 협공 보상	· 이 과정으로 획득할 수 없음
수련장	· 이 과정으로 획득할 수 없음
용병 활동	· 이 과정으로 획득할 수 없음
보유한 재화 판매 (장비, 소모품, 재료 등)	· 이 과정으로 획득할 수 없음
아이템 강화 결과	· 이 과정으로 획득할 수 없음
아이템 제작 결과	· 이 과정으로 획득할 수 없음

일일 미션 달성 보상	• 이 과정으로 획득할 수 없음
업적 달성 보상	• 이 과정으로 획득할 수 없음
일일 접속 보상	• 이 과정으로 획득할 수 없음
서비스 운영에 의한 비정기적 지급	• 이 과정으로 획득할 수 없음
VIP 레벨에 의한 지급 방식 변화	• 이 과정으로 획득할 수 없음
상설 상점 구매	• 이 과정으로 획득할 수 없음
숨겨진 상점 구매	• 이 과정으로 획득할 수 없음
무작위 뽑기	• 이 과정으로 획득할 수 없음

◆ B-24-18. 결투장 코인

- **B-24-18-1.** 사용자 대전을 플레이 하면 획득할 수 있는 화폐이다.
  : 사용자 대전 점수와는 별도로 주어진다.

- **B-24-18-2.** 사용자 대전 상점에서 사용할 수 있다.
  : 여기에서 일반적인 방법으로는 구하기 어려운 재화와 아이템들을 구입하는 데 사용한다.

- **B-24-18-3.** 이 재화를 획득할 수 있는 컨텐츠들은 다음과 같다.

종류	설명
스토리 모드(일반) 완료	• 이 과정으로 획득할 수 없음
스토리 모드(정예) 완료	• 이 과정으로 획득할 수 없음
~~일일 던전 완료~~	~~• 이 과정으로 획득할 수 없음~~
주간 던전 완료	• 이 과정으로 획득할 수 없음
혼돈 던전 완료	• 이 과정으로 획득할 수 없음
초월 던전 완료	• 이 과정으로 획득할 수 없음
숨겨진 던전 완료	• 이 과정으로 획득할 수 없음
결투장 승리	• 획득 가능
길드 던전 완료	• 이 과정으로 획득할 수 없음
길드 전쟁 승리	• 이 과정으로 획득할 수 없음
월드 보스 협공 보상	• 이 과정으로 획득할 수 없음
수련장	• 이 과정으로 획득할 수 없음
용병 활동	• 이 과정으로 획득할 수 없음
보유한 재화 판매 (장비, 소모품, 재료 등)	• 이 과정으로 획득할 수 없음

아이템 강화 결과	• 이 과정으로 획득할 수 없음
아이템 제작 결과	• 이 과정으로 획득할 수 없음
일일 미션 달성 보상	• 이 과정으로 획득할 수 없음
업적 달성 보상	• 이 과정으로 획득할 수 없음
일일 접속 보상	• 이 과정으로 획득할 수 없음
서비스 운영에 의한 비정기적 지급	• 이 과정으로 획득할 수 없음
VIP 레벨에 의한 지급 방식 변화	• 이 과정으로 획득할 수 없음
상설 상점 구매	• 이 과정으로 획득할 수 없음
숨겨진 상점 구매	• 이 과정으로 획득할 수 없음
무작위 뽑기	• 이 과정으로 획득할 수 없음

◆ B-24-19. 길드 코인(Guild Coin)

- **B-24-19-1.** 길드 전쟁을 플레이 하면 획득할 수 있는 화폐이다.
  : 길드 명예 점수와는 별도로 주어진다.

- **B-24-19-2.** 길드 상점에서 사용할 수 있다.
  : 여기에서 일반적인 방법으로는 구하기 어려운 재화와 아이템들을 구입하는 데 사용한다.

- **B-24-19-3.** 이 재화를 획득할 수 있는 컨텐츠들은 다음과 같다.

종류	설명
스토리 모드(일반) 완료	• 이 과정으로 획득할 수 없음
스토리 모드(정예) 완료	• 이 과정으로 획득할 수 없음
~~일일 던전 완료~~	~~• 이 과정으로 획득할 수 없음~~
주간 던전 완료	• 이 과정으로 획득할 수 없음
혼돈 던전 완료	• 이 과정으로 획득할 수 없음
초월 던전 완료	• 이 과정으로 획득할 수 없음
숨겨진 던전 완료	• 이 과정으로 획득할 수 없음
결투장 승리	• 획득 가능
길드 던전 완료	• 이 과정으로 획득할 수 없음
길드 전쟁 승리	• 이 과정으로 획득할 수 없음
월드 보스 협공 보상	• 이 과정으로 획득할 수 없음
수련장	• 이 과정으로 획득할 수 없음
용병 활동	• 이 과정으로 획득할 수 없음

보유한 재화 판매 (장비, 소모품, 재료 등)	• 이 과정으로 획득할 수 없음
아이템 강화 결과	• 이 과정으로 획득할 수 없음
아이템 제작 결과	• 이 과정으로 획득할 수 없음
일일 미션 달성 보상	• 이 과정으로 획득할 수 없음
업적 달성 보상	• 이 과정으로 획득할 수 없음
일일 접속 보상	• 이 과정으로 획득할 수 없음
서비스 운영에 의한 비정기적 지급	• 이 과정으로 획득할 수 없음
VIP 레벨에 의한 지급 방식 변화	• 이 과정으로 획득할 수 없음
상설 상점 구매	• 이 과정으로 획득할 수 없음
숨겨진 상점 구매	• 이 과정으로 획득할 수 없음
무작위 뽑기	• 이 과정으로 획득할 수 없음

◆ B-24-20. 무작위 뽑기 횟수

- **B-24-20-1.** 무작위 뽑기 상점에서 무료로 뽑기를 할 수 있는 횟수이다.

- **B-24-20-2.** 무작위 뽑기는 기본적으로, 요금을 지불하고 이용하는 컨텐츠이지만, 경우에 따라 제한적으로 무료로 이용할 수 있는 기회를 제공하기도 한다.

- **B-24-20-3.** 이 재화를 획득할 수 있는 컨텐츠들은 다음과 같다.

종류	설명
스토리 모드(일반) 완료	• 이 과정으로 획득할 수 없음
스토리 모드(정예) 완료	• 이 과정으로 획득할 수 없음
~~일일 던전 완료~~	~~• 이 과정으로 획득할 수 없음~~
주간 던전 완료	• 이 과정으로 획득할 수 없음
혼돈 던전 완료	• 이 과정으로 획득할 수 없음
초월 던전 완료	• 이 과정으로 획득할 수 없음
숨겨진 던전 완료	• 이 과정으로 획득할 수 없음
결투장 승리	• 이 과정으로 획득할 수 없음
길드 던전 완료	• 이 과정으로 획득할 수 없음
길드 전쟁 승리	• 이 과정으로 획득할 수 없음
월드 보스 협공 보상	• 이 과정으로 획득할 수 없음
수련장	• 이 과정으로 획득할 수 없음

용병 활동	• 이 과정으로 획득할 수 없음
보유한 재화 판매 (장비, 소모품, 재료 등)	• 이 과정으로 획득할 수 없음
아이템 강화 결과	• 이 과정으로 획득할 수 없음
아이템 제작 결과	• 이 과정으로 획득할 수 없음
일일 미션 달성 보상	• 이 과정으로 획득할 수 없음
업적 달성 보상	• 이 과정으로 획득할 수 없음
일일 접속 보상	• 이 과정으로 획득할 수 없음
서비스 운영에 의한 비정기적 지급	• 그 외 비정기적인 방식으로 서비스 운영 단계에서 사용자에게 공급하는 경우 • 예를 들면, 장기 미접속자가 오랜만에 다시 접속했을 때, 플레이 유도를 위해 지급하는 경우를 들 수 있다.
VIP 레벨에 의한 지급 방식 변화	• 이 과정으로 획득할 수 없음
상설 상점 구매	• 과금 화폐로 구매할 수 있다.
숨겨진 상점 구매	• 이 과정으로 획득할 수 없음
무작위 뽑기	• 뽑기 과정 중에, 횟수를 일정량(이라고 해봐야 보통 1 회) 늘려주는 아이템을 획득할 수도 있다.

B-25. 플레이 보상

◆ B-25-1. 보상 체계의 원칙

- **B-25-1-1.** 모든 종류의 게임 플레이에 대한 보상은, 그 종류와 수량을 온전히 **서버에서만 결정**한다.
  : 클라이언트는 게임 플레이에서의 보상에 대한 어떠한 판정도 하지 못한다.

- **B-25-1-2.** 게임 플레이에 대한 보상을 획득할 수 있는 조건에 도달했음에도 불구하고, 보상을 획득하기 전에, 해당 플레이어의 기기가 연결이 끊어졌다면(연결 유실 등의 이유로 인해), 서버는 그 플레이어가 아직 받지 못한 보상 품목과 수량을 기억하고 있다가, 다음 접속 시에 지급해야 한다.

※ 구현을 복잡하게 만들지 않으려면, 연결이 끊어진 플레이어가 다시 로그인을 한 직후에, 이전에 끊겼던 플레이에 대한 보상을 곧바로 지급하게 만드는 편이 좋다고 판단한다.

- **B-25-1-3.** 게임 플레이에 대한 보상 정보들을 서버와 클라이언트가 주고 받는 방식은 실제로는 비동기적이다. 하지만, 동기적으로 주고 받는 것처럼 연출할 수 있다.
  : 보상을 획득하는 시점은 게임 플레이 도중에 무작위로 만나는 것처럼 보이지만, 실제 보상은 해당 스테이지가 다 끝나고 몰아서 주는 방식으로 만든다.

※ PC 용 MMORPG 처럼 모든 상호작용들이 즉시 동기화하는 방식에 비추어보면 좀 이상하다고 여길지도 모른다.
그러나 모바일 환경에서 최대한 네트워크 연결을 덜 하면서도, 비동기적으로 동작하는 보상 방식(스테이지 끝나면 몰아주기)이 덜 이상하도록 보이기 위한 방편이다.

- **B-25-1-4.** 아이템이나 스킬 룬에 대한 보상을 할 때는, 종류와 등급까지만 지정할 수 있고, 보유한 세부 능력치의 종류와 값을 구체적으로 지정할 수 없다.
  : 아이템이나 스킬 룬의 능력 속성들의 종류와 값은, 그것을 생성할 때 결정하기 때문에, 미리 결정해서 플레이어에게 보여줄 수 없다.

※ 플레이어에게 보상으로 지급할 아이템을 '희귀 등급 장검'까지는 지정해 줄 수 있어도, '공격력이 156 ~ 175 에 화염 피해 +100, 힘 +25 인 희귀 등급 장검'과 같이 세부 능력치까지 구체적으로 정해줄 수는 없다는 뜻이다.

◆ B-25-2. 스테이지 내에서의 보상

- **B-25-2-1.** 스테이지를 플레이 하는 도중에, 적을 물리치면 무작위로 돈과 아이템을 '땅으로 떨어뜨린다.'

- **B-25-2-2.** 그러나, 이 '무작위로 떨어뜨리는 보상'은 해당 스테이지를 불러올 때, 이미 서버에서 정의해서 내려주는 정보들이다.
: 즉, 예를 들자면, X번째 몬스터를 잡으면 10골드가 떨어지도록 미리 설정이 되어 있다는 의미이다.

- **B-25-2-3.** 스테이지를 플레이 하는 도중에 얻는 보상들은 **그 스테이지의 임무를 성공하거나 실패하는지 여부와 관련 없이 지급**한다.

- **B-25-2-4.** 스테이지를 플레이 하는 도중에 등장하는 보상을 정하고 지급하는 구체적인 방식은 다음과 같다.

1. 플레이어가 자신이 플레이 할 스테이지를 선택한다.

2. 서버는 플레이어의 스테이지 진입 요청을 받고, 해당 스테이지에 대한 보상 정보들을 만들기 시작한다.
: 이 때, 해당 스테이지의 몬스터 배치 정보들을 참조하여, 각 몬스터들이 보상을 줘야 할지, 줘야 한다면 어떤 아이템이나 화폐를 얼만큼 주도록 설정할지를 결정한다.
또한, 이러한 각 보상 목록들은 인덱스를 가지고 있어서, 어떤 보상을 획득했는지, 아닌지를 구분할 수 있다.

3. 만들어 낸 보상 정보들을 클라이언트에게 전달한다. 이 때, 클라이언트는 아직 로딩 장면에 머물러 있다.
이 정보들을 받지 못해도 게임 플레이를 진행할 수는 있으나, 이런 경우에는 **'게임 플레이 중에 어떠한 화폐나 아이템도 떨어지지 않는다.'**

4. 클라이언트는 서버로부터 로딩하는 스테이지에 대해, 서버가 만들어 낸 보상 관련 정보들을 수신 받을 때까지 로딩 화면을 유지한다.
단, 사전에 정의한 시간 이내에 정보를 수신 받지 못했고, 그 상태에서 스테이지를 구성하는 클라이언트 측 리소스 데이터들을 모두 불러왔으면, 그대로 게임 플레이 장면으로 진행한다. (바로 이 때는 게임 플레이 도중에 무작위로 떨어지는 보상이 없다는 뜻이다.)

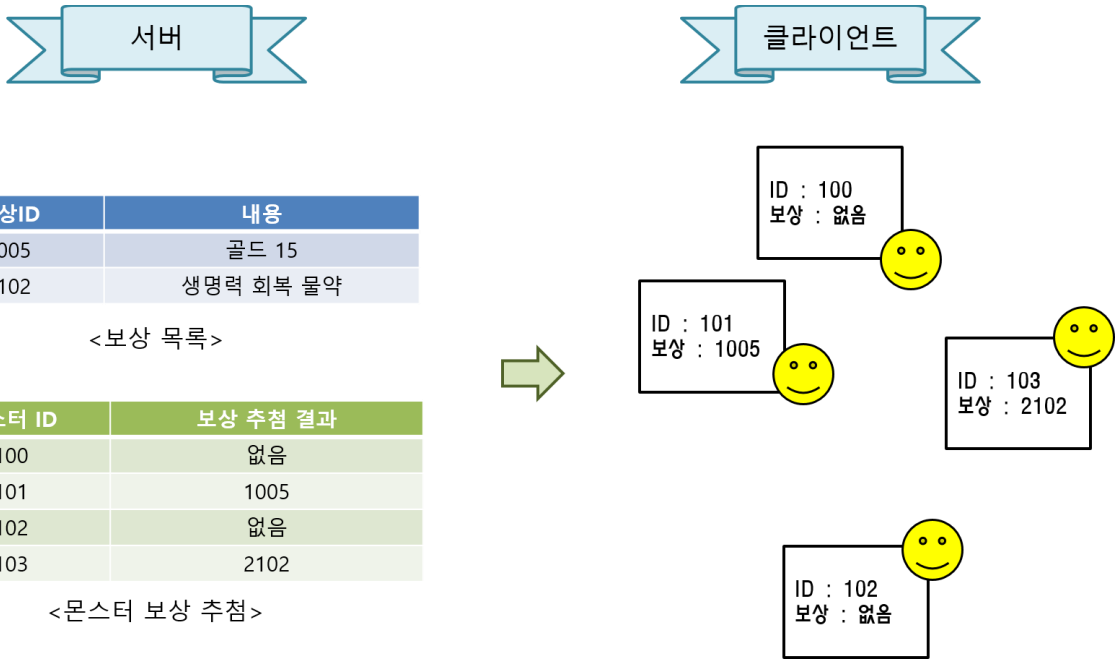
5. 스테이지를 플레이 하는 도중에 보상을 만나면 (예를 들어 10번 몬스터를 죽이면 50골드가 떨어진다든가...), 클라이언트는 이를 기록한다.

단, 기록하는 방식은 서버가 보내 온 보상 목록의 인덱스일 뿐이다.
보상 내용에 대한 판정은 서버가 하기 때문에, 보상의 내용은 기록하지 않는다.

6. 스테이지를 종료할 때(스테이지의 임무를 완료했든 아니든 간에), 보상 단계에서 클라이언트는 현재까지 기록한 플레이어 캐릭터의 보상 획득 정보를 서버에게 전달한다.

7. 서버는 클라이언트가 보내 온 보상 정보의 인덱스들을 토대로, 클라이언트에게 지급해야 할, 스테이지를 진행하면서 얻은 아이템과 화폐의 종류와 액수를 계산한다.
: 클라이언트는 보상 목록의 인덱스만 알 뿐, 구체적인 보상 종류와 개수, 화폐 액수 등은 모른다.

8. 서버가 클라이언트로부터 확인한 정보를 토대로 계산한 보상을 플레이어의 계정 정보에 적용한다.
: 아이템 지급, 화폐 지급을 말한다.



보상ID	내용
1005	골드 15
2102	생명력 회복 물약

<보상 목록>

몬스터 ID	보상 추첨 결과
100	없음
101	1005
102	없음
103	2102

<몬스터 보상 추첨>

<스테이지 플레이 중 보상에 대한 서버와 클라이언트의 처리 개념도>

- **B-25-2-5.** 스테이지를 플레이 하는 도중에 만나는 보상 아이템과 화폐들은 획득하는 과정이 별도로 없고, 발생하자마자 플레이어 캐릭터가 자동으로 획득한 것으로 처리한다.

- **B-25-2-6.** 스테이지를 플레이 하는 도중에 만나는 보상 아이템과 화폐를 획득할 때마다, 무엇을 얼마나 획득했는지에 대해 알려주는 연출을 재생한다.

※ 게임 상 화폐를 획득할 때, 금화 모양의 모델이 땅으로 떨어졌다가 플레이어의 캐릭터를 향해 휙 날아가면서 흡수되는 연출과 함께 '+10'의 숫자가 잠깐 떠오르는 식이다. 이러면 10골드의

금화를 획득했다고 알려준다는 뜻이다.

- **B-25-2-7.** 스테이지를 플레이 하면서 '무작위로' 획득하는 보상은, 스테이지를 플레이 한 뒤, 임무 성공으로 주는 보상보다 **평균적으로 가치가 낮아야 한다.**
: 어디까지나 **주된 보상은 스테이지의 임무 수행을 완료한 뒤의 보상**이고, 스테이지를 플레이 하면서 만나는 보상들은 작은 보너스를 주는 셈이라는 관점으로 설계한다.

※ 기술적인 이유로는, **무작위로 주어진 보상을 획득했는지 여부를 서버가 아니라 클라이언트가 판정하기 때문**이다. 사실, 보상을 획득할 때마다 서버에게 확인을 요청하는 방법도 있긴 한데, 급박하게 돌아가는 게임 플레이 도중에 그런 연결을 수시로 시도한다는 게 그다지 좋은 설계라고 생각하지 않았다.
그런 이유로 인해, '게임 다 끝나고 여유가 있을 때', 서버에서 판단을 한 번에 다 하도록, 클라이언트가 그 동안 기록을 대행하는 셈인데, 이유야 어쨌든, 클라이언트에게 어떤 판정이라도 맡긴다는 건 위험한 일이다. 클라이언트가 실제로 플레이를 어떻게 진행을 했든 간에, 스테이지를 플레이 하면서 얻을 수 있는 보상을 모두 획득했다고 기록을 조작해서 서버에게 제출할 수도 있다.

위의 관점에서 보자면, **스테이지를 플레이 하는 도중에 무작위로 떨어지는 보상의 값어치를 사소하게 만든다면,** 클라이언트에게 뭔가 판정을 맡겼을 때, 그 피해(?)를 최소화할 수 있는 방법이 된다. 클라이언트에서 할 수 있는 해킹이라고 해 봐야, 무작위 보상들을 전부 획득했다고 조작하는 정도인데, 그 '전부 획득한 총합'이 스테이지 임무 수행 결과에 대한 보상에 비해 가치가 미미하다면, 클라이언트의 판정 과정을 해킹하는 수고를 하는 가치도 따라서 같이 미미해지기 때문이다.

※ 또한, 스테이지의 임무를 완수했을 때의 보상이 훨씬 더 커야, 플레이어들이 스테이지 임무를 완수하도록 유도할 수 있다.

- **B-25-2-8.** 스테이지를 플레이 하는 도중에, 어떤 장애로 인해 게임 플레이가 비정상적인 경로를 통해 완전히 중단되고(네트워크 연결 끊김, 전원 차단 등), 아직 그 스테이지 내에서의 플레이 보상을 받지 못한 경우, 클라이언트는 그 보상 정보를 소실한 것으로 간주한다.

※ 그냥 메모리에 들고 있는 정보는 휘발성이므로, 애플리케이션이 종료되는 즉시 사라진다.
그 때문에, 클라이언트가 보상 정보가 발생할 때마다 매번 디스크에 기록을 해야 하는 셈인데, 파일 I/O 동작 자체가 가장 느린 저장 방식인 점이 이러한 결정의 원인 중 하나다.

다른 원인으로는 역시 클라이언트 측에서의 조작 우려 때문이다. 클라이언트가 비정상 종료했을 때, 이후에 서버에게 '재청구' 하는 게 가능하다면, 극단적으로 클라이언트는 수도 없이 '재청구'를 해서 서버로부터 아이템과 화폐를 뜯어내는 것도 가능할 수 있다. 아예 그런 가능성 자체를 차단하기 위해서, 게임 플레이 도중에 얻는 보상은, 게임을 비정상 종료할 때에는 그 회수

를 책임져주지 않게 정책을 결정한다.

◆ B-25-3. 스테이지 임무 결과에 대한 보상

- **B-25-3-1.** 스테이지 임무 결과에 대한 보상은, 해당 스테이지의 임무를 성공했을 경우에만 적용한다.

- **B-25-3-2.** 스테이지 임무 결과에 대한 보상은, 그 스테이지의 게임 플레이를 종료하고 로비로 돌아가기 직전에 별도의 보상 수여 장면을 두고, 그 장면 중에만 지급한다.
: 다 끝나고 일시에 지급하는 방식이다.

※ 서버와의 연결을 계속 유지하는 방식이 아니고, 필요할 때만 제한적으로 연결하기 때문에, 지속적으로 서버와 연결을 유지해야 하는 보상 체계를 구축하기 어렵다.
그러므로, 게임의 결과가 나오는 시점에 한 번에 서버에 요청해서 보상을 주는 방식으로 설계한다.

- **B-25-3-3.** 스테이지 임무 결과에 대한 보상은 스테이지가 시작할 때 미리 결정하지 않는다.
: 임무 결과가 나온 시점에 서버에 요청하면, 그 때 보상 품목과 수량을 결정하기 위한 계산을 시작한다.

※ 스테이지가 막 시작하는 시점은 아직 결과도 나오기 전이므로, 보상 품목과 수량을 결정하는 건 이치에 맞지도 않다.

~~- **B-25-3-4.** 임무 결과에 대한 보상은 고정되어 있지 않으며, **플레이어가 '무작위로 뽑는' 과정을 거쳐야 한다.**~~
~~: 일반적으로 카드나 보물상자를 3개 늘어놓고, 화려한 FX와 함께 고르도록 만드는 연출을 보여준다.~~

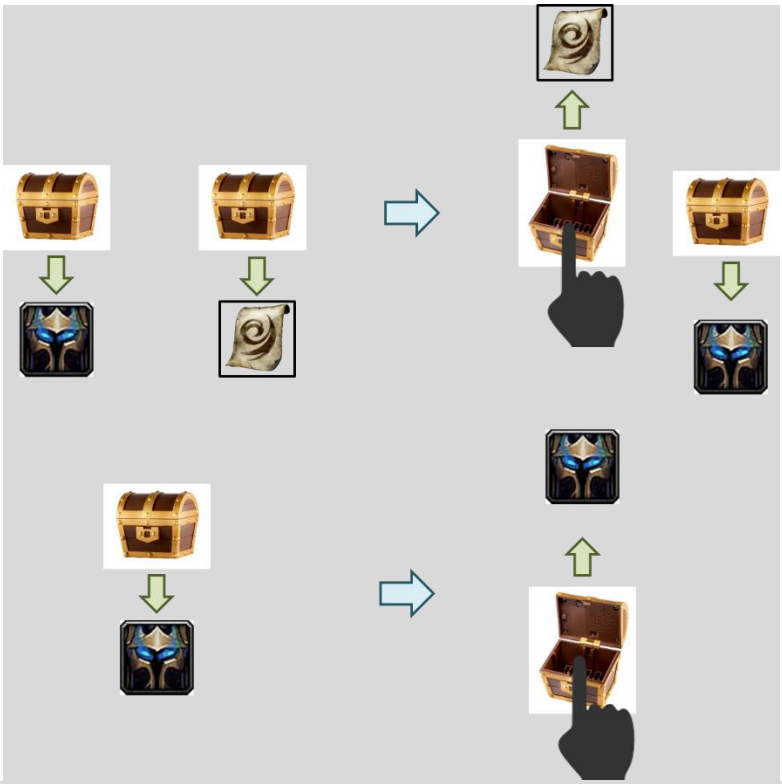
~~- **B-25-3-5.** 플레이어가 뽑기를 수행하기 전에, 어떤 보상을 얻을 수 있는지 확인하는 과정이 있어야 한다.~~
~~: 그래야 좋은 보상을 획득하기 위해 '한 번 더 뽑는' 과정을 거치고자 할 테니까...~~

<보상에 대한 무작위 뽑기 수행>

~~**- B-25-3-6.** 플레이어는 최초에 수행한 뽑기 결과를 수령하지 않고, 비용을 지불해서 해당 스테이지 임무 결과에 대한 보상 뽑기를 다시 수행할 수 있다.~~
~~: 구체적인 비용은 관련 기획서의 요구사항을 따른다.~~

~~**- B-25-3-7.** 하나의 스테이지 결과에서 보상 뽑기를 다시 수행할 수 있는 횟수는 **[뽑기 항목의 전체 개수 - 1] 회**이다.~~
~~: 뽑기 항목이 3 개짜리로 구성하였으면 최초 시도 1 회 + 재시도 2 회이다.~~

~~**- B-25-3-8.** 보상 뽑기를 재시도할 때마다, 이전에 뽑았던 패는 뽑기에서 제외한다.~~
~~: 물론, 제외한 패의 보상은 수령할 수 없으며, 그냥 없어진다.~~



<원하는 걸 얻을 때까지 집념의(?) 뽑기 재시도. 물론 이렇게 하려면 비용이 필요하다.>

- **B-25-3-9.** 뽑기에 대한 보상을 수령하면, 보상 절차는 마무리하며, **보상 수령 이후에는 결과를 번복하거나 추가로 그 스테이지 임무에 대한 보상을 획득할 수 없다.**

- **B-25-3-10.** 임무 결과에 대한 보상은 **상, 중, 하의 3등급**으로 나눈다.
: 이는 플레이어에게 연출할 때, 카드 혹은 보물상자를 3개 중에서 선택하는 경우에 대응한다.

※ 대개는 3 개 중에서 선택하는 방식으로 하지만, 혹여 '난 독특함을 추구하겠어!'(...)라고 해서 4 개나 5 개 중에서 뽑는 방식이라면, 등급도 4 개, 5 개로 나눠야 한다.

- **B-25-3-11.** 보상 등급에 따라 구체적으로 어떤 아이템이 등장하는지는 사전에 데이터 테이블로 정의한다.
: 일정한 규칙으로 공식화해서 적용할 수 있다면, 그렇게 해도 된다.

※ 상급 보상이라면, 등장할 수 있는 아이템들은 높은 등급의 아이템들이다. 하급 보상이라면 게임 화폐 몇 푼 정도 나오거나 낮은 등급의 아이템들 중에서 등장하도록 만든다. 중급 보상은 그 중간 정도로 책정한다.

- **B-25-3-12.** 플레이어가 패를 뽑을 때, 좋은 보상을 가진 패가 나올 확률은 보상을 가진 패의 급이 높을수록 낮아진다.

- **B-25-3-13.** 플레이어에게 지급하는 보상의 등급은, 플레이어가 선택한 패와는 관련이 없다.
: 플레이어가 보상에 대한 패를 선택하는 과정은 일종의 연출일 뿐이며, 실제로는 서버 내부에서 사전에 정의한 확률에 따라, 선택된 등급의 보상을 지급한다.

※ 직설적으로, 이 과정에서 클라이언트가 하는 일은, 그저 서버가 계산한 결과를 보여주는 것뿐이다. 그러므로, 일단 서버가 결정한 이상, 플레이어가 어떤 패를 선택하더라도 서버가 결정한 결과만을 보여주도록 되어 있다.

<보상 뽑기에 대한 암울한 진실(...)>

※ ~~플레이어에게 나열되어 있는 패들 중에서 좋은 아이템이 있는 패가 존재한다는 걸 '보여주기만 하고', 실제로는 뽑기 아주 어렵게 만드는 방식이라고 할 수 있다. (아무튼 33.3333...%는 절대 아니다.)~~

~~하지만, 일단 보상 지급을 선택하는 패들 중의 하나로 나온 이상, 비용만 끝까지 지불한다면 확실하게 지급을 보장(?)해준다.~~

~~이는 플레이어들이 '다시 뽑기'에 대한 비용을 지불하도록 유도하기 위한 장치다.~~

※ 수정(2015. 05. 18 조수운)

스테이지의 보상은 이제 더 이상 '카드 뽑기' 방식으로 이루어지지 않고, 자동 지급하는 방식으로 변경한다.

이 부분의 내용 역시 그에 맞춰서 개정하였다.

◆ B-25-4. 게임 플레이 외 활동에 대한 보상

- **B-25-4-1.** 업적에 대한 보상

: 구체적인 보상 내역은 관련 기획서의 요구사항을 따른다.

- **B-25-4-2.** 업적에 대한 보상은, 그 업적을 완료한 시점에 곧바로 지급하고 플레이어에게 통지한다.

- **B-25-4-3.** 출석에 대한 보상
: 구체적인 보상 내역은 관련 기획서의 요구사항을 따른다.

- **B-25-4-4.** 출석 점검은 현실 시간으로 달이 바뀔 때마다 초기화한다.

- **B-25-4-5.** 출석에 대한 보상 방식은 매일, 혹은 주간, 한달 등의 특정 주기마다 보상을 부여할 수 있고, 일정한 기간마다 다른 보상을 주거나 추가 보상을 줄 수 있어야 한다.

- **B-25-4-6.** 출석 점검과 이에 대한 보상은, 플레이어가 자신의 계정을 로그인해서 게임 로비에 접속한 시점에 확인하고 지급한다.
: 그냥 인트로 화면에만 머물고, 로그인을 하지 않고 종료하면 출석 점검 및 보상을 제공하지 않는다.

※ 로그인을 해서 서버에 접속을 하지 않으면 해당 사용자의 접속 여부를 확인할 길이 없기 때문에, 플레이어의 로그인은 출석 점검에 있어 필수 사항이다.

- **B-25-4-7.** 일일, 주간, 월간 미션에 대한 보상은 누적하여 지급하지 않는다.
: 보상을 수령하지 않은 상태에서, 조건을 여러 번 충족하더라도, 보상은 1회만 받을 수 있다.

- **B-25-4-8.** 일일, 주간, 월간 미션에 대한 보상은, **그 보상을 수령하기 전까지는 다음 번 보상을 얻기 위한 조건 점검을 정지**한다.
: 보상을 수령하지 않은 상태의 일일, 주간, 월간 미션은 보상을 수령할 때까지는 그 다음 회차의 미션을 수행하지 않은 상태로 간주한다.

◆ B-25-5. 맞춤형 보상 처리

- **B-25-5-1.** 플레이어의 직업에 따른 맞춤형 보상
: 보상을 지급할 때는, 그 보상을 수령할 플레이어 캐릭터의 직업이 사용할 수 있는 아이템만을 보상으로 지급해야 한다.

※ 아이템 중에는 직업에 관계없이 공통으로 사용할 수 있는 아이템들도 있지만, 해당 직업만 전용으로 사용해야 하는 아이템들도 있다. 특히, 장착 장비 아이템(Wearable Item)들은 전부 직업 전용 아이템으로 계획하고 있다.

- **B-25-5-2.** 캐릭터가 돌파한 스테이지, 막(Act)의 난이도에 따른 맞춤형 보상
 : 캐릭터가 돌파한 스테이지나 막(Act)의 난이도에 맞춰서 아이템들의 기본 수준을 결정한다.

※ 게임 초반에 등장하는 스테이지의 보상으로 등장하는 검과 갑옷은, 게임의 가장 후반에 등장하는 스테이지의 보상으로 등장하는 검과 갑옷과는 기본 능력치에서부터 현격하게 차이가 난다.

- **B-25-5-3.** 과금을 이용한 무작위 아이템 뽑기를 통해서 아이템을 획득할 때도, 캐릭터가 막과 스테이지를 어느 정도 돌파했느냐에 따라 아이템의 기본 수준이 달라진다.
 : 플레이어가 무작위 아이템 뽑기를 통해 얻은 아이템들이, 플레이어의 현재 스테이지 수준에 어울리지 않게 너무 강력해서 플레이가 재미없어지지 않게 하려는 의도이다.
 이 게임은 기본적으로 아이템에 어떤 레벨 제한이 있는 게 아니기 때문에, 플레이어가 현재 플레이 해야 하는 스테이지보다 지나치게 강력한 아이템이 나올 가능성을 아예 차단하는 방식으로 제한을 둔다.

※ 다만, 특별히 더 많은 요금을 지불한 경우에는 플레이어의 스테이지 진행 상황과 관계없이 아이템의 '전체 영역'에서 무작위 뽑기를 할 수 있는 프리미엄을 주는 방식을 고려할 수도 있다고 본다.

◆ B-25-6. 보상 재화

- **B-25-6-1.** 보상 재화는 다음과 같은 분류 체계를 가진다.
 : 재화를 분류하는 방식 자체와 동일하다.

1. 게임 플레이 화폐
2. 과금 화폐
3. 입장료 단위
4. 전용 화폐
   4-1. 초월 코인
   4-2. 결투장 코인
   4-3. 길드 코인
5. 아이템 제작(승급) 재료
6. 보석
7. 아이템 뽑기 횟수
   7-1. 무료 무작위 아이템 뽑기 횟수(일반)
   7-2. 무료 무작위 고급 아이템 뽑기 횟수(고급)
   7-3. 무료 부위별 아이템 뽑기 횟수(일반)
   7-4. 무료 부위별 아이템 뽑기 횟수(고급)
8. 스킬 룬 뽑기 횟수

8-1. 무료 무작위 스킬 룬 뽑기 횟수
8-2. 무료 특정 타입 스킬 룬 뽑기 횟수
9. 무료 선물 뽑기 횟수
9-1. 무료 일반 선물 뽑기 횟수
9-2. 무료 고급 선물 뽑기 횟수
10. 스테이지 일회성 부여효과 무료 사용 횟수(각 종류 별로)
11. (실제 시간 기반의) 부여 효과

- **B-25-6-2.** 보상이 필요한 시점에서, 무작위로 보상을 해야 하는 경우가 있다면, 위 분류에 속하는 재화들을 보상으로 획득할 수 있다.

- **B-25-6-3.** 보상해야 하는 성격에 따라, 특별한 재화의 종류나, 아예 특정한 고유 코드 번호를 가진 재화 대상을 보상으로 지급할 수 있다.

※ 이를 테면, 오랜만에 접속한 플레이어에게는 반드시 입장 자원과 과금 화폐를 보상(보통 선물로 표현하는)으로 보내줘야 한다면, 보상 타입으로 입장 자원과 과금 화폐를 특별히 지정해서 보상할 수도 있어야 한다는 뜻이다.

B-26. 파츠

◆ B-26-1. 파츠(Parts)의 정의

- **B-26-1-1.** 캐릭터를 표현하는 모델링 데이터가 여러 개의 부위로 나누어져 있고, 그 부위들을 조합해서 하나의 캐릭터를 표현하는 방식일 때, 그 각각의 부위 모델을 파츠라고 한다.

- **B-26-1-2.** 실제로는 캐릭터의 신체 부위 뿐만 아니라, 캐릭터의 신체 일부에 부착하는 모델들도 파츠라고 부른다. (개체의 신체 일부로써, 또는 개체의 신체 중 특정 위치에 달라붙기 때문이다.)

※ 파츠의 분류는 기획 및 성능 정책, 아트 리소스 데이터의 표현 한계 정책 등에 의해 달라질 수 있다.

- **B-26-1-3.** 파츠는 **부착형 파츠(Attach Parts)**와 **신체형 파츠(Skinning Parts)**로 구분한다.

◆ B-26-2. 부착형 파츠(Attach Parts)

- **B-26-2-1.** 캐릭터 개체의 특정한 부위에 붙이는 모델을 가리킨다.

- **B-26-2-2.** 매달린 모델 자체도 별도의 독립적인 개체 오브젝트라고 할 수 있지만, 일반적으로 캐릭터 개체에 달아놓고 쓴다.

- **B-26-2-3.** 특징은 **애니메이션이 없다**는 점이다. 그래서 단순히 캐릭터 개체의 특정 부위에 Transform을 계층적으로 달아놓거나, 아니면 특정 Transform으로부터 위치 정보만 받아서 위치 갱신만 하기도 한다.

- **B-26-2-4.** 애니메이션이 필요 없는 아이템 모델들은 다 여기에 해당한다.
: 주로 무기 종류가 그렇다.

※ 모든 무기나 보조 무기, 방패 종류 등이 부착형 파츠일 것이라고 단정지어서는 안 된다.
그런 식으로 타입을 분류한다면, 나중에 채찍과 같이 캐릭터와 연결된 애니메이션을 필요로 하는 무기 컨텐츠를 추가할 때 문제가 된다.

뭐, 그런 거 구현해달라고 할 때 시스템 상 안 된다고 하면 그만일 수도 있지만, 할 수 있는데 구현해보니 재미 없어서 안 넣은 것과, 넣기 곤란해서 못 넣은 건 차이가 크다.(...)

◆ B-26-3. 신체형 파츠(Skinning Parts)

- **B-26-3-1.** **캐릭터 개체의 특정한 부위를 교체하는 모델**을 가리킨다.

- **B-26-3-2.** 캐릭터 개체의 신체 일부를 교체하는 것이기 때문에 당연히 **파츠 모델마다 애니메이션이 존재**한다.

- **B-26-3-3.** 또한, 캐릭터 개체가 다른 종류일 경우, 애니메이션 키 프레임의 구성 정보 역시 다르기 때문에, **같은 종류의 아이템이라도 캐릭터 개체의 종류 별로 모델이 따로 들어가야** 한다.

- **B-26-3-4.** 신체형 파츠는 파츠를 구성하는 메시(Mesh)들이 각 애니메이션 뼈대 정보를 직접 참조해야 하기 때문에, 스키닝(Skinning) 과정이 반드시 필요하다.
: 또한, 이런 특성 때문에 **신체형 파츠는, 붙일 대상과 완전히 똑같은 애니메이션 정보를 가져야 한다.**

◆ B-26-4. 파츠의 표현

- **B-26-4-1.** 어떤 물체를 파츠 시스템으로 표현하는 경우, 파츠로 나뉜 부위 별로 별도의 객체로 분류할 수 있는 Skinning Mesh가 존재한다.
: 만일 캐릭터의 경우라면, 신체 일부를 표현하는, Skinning Mesh 들을 합치는 식으로 캐릭터의 전체적인 신체 외형을 표현할 수 있는 것이다.

※ 어떤 개체의 모델 리소스를, **특정한 부위 별로 Skinning Mesh 를 나눠서 표현하는 행위가 반드시 파츠 시스템을 사용한다는 사실을 의미하지는 않는다.** 그 이유는 다음과 같다.

이를테면, 마법으로 빛나는 커다란 보석이 박혀 있는 왕관 모델이 있다고 치자. 이 왕관의 보석은 수시로 일정 시간마다 색깔이 바뀌어야 한다. 하지만 왕관에 박힌 보석만 색깔이 바뀌는 것이지, 왕관 자체가 색깔이 바뀌는 것은 아니다.
이럴 경우, 왕관 Mesh 와 보석 Mesh 가 일체화된 상태이면, 객체 단위로 보석 Mesh 부분에만 접근할 수 없으므로 이런 상황을 표현하기 난감해진다. 그러나 보석 부분만 제외한 왕관 나머지 부분과 보석의 Mesh 가 별도의 Mesh 층으로 분리된 상태라면, 정확히 보석 부분의 Mesh 에만 재질 변화를 줘서 보석만 색깔이 변하는 것처럼 보이게 만들 수 있다.

- **B-26-4-2.** 신체형 파츠의 표현

: 교체 대상이 되는 Skinning Mesh 들을 교환하는 방식으로 표현한다.
이러한 표현 방식으로는, 캐릭터의 신체 전체를 신체의 각 부분 별로 나눈 신체형 파츠들의 집합으로 나타낼 수 있다. 이 때, 신체형 파츠들의 구획 방식은, 파츠를 어떻게 나눌지에 대한 분류 기획에 따라 달라진다.

※ 신체형 파츠가 될 각 파츠 Mesh 들은, 리소스 데이터 입장에서 보면, 해당 캐릭터의 애니메이션 뼈대 정보에 붙어 있는 신체 일부분을 보여주는 Skinning Mesh 라고 볼 수 있다.
이러한 Skinning Mesh 들은 별도의 파일로 나뉘어져 있더라도, 애니메이션을 위한 뼈대 정보가 반드시 같이 들어가게 된다. 즉, **파츠 리소스 파일마다 애니메이션 뼈대 정보가 하나씩 붙어 다닌다.**

파츠 리소스 데이터에서 Skinning Mesh 가 붙어 있는 애니메이션 뼈대의 정보는, **파츠를 조합하는 캐릭터의 애니메이션 뼈대 정보와 동일해야** 한다. 그래야 파츠들을 서로 조합해도 모델의 애니메이션이 정상적으로 보일 것이다. 따라서, **애니메이션 뼈대 정보가 서로 다른 신체형 파츠들끼리는 조합할 수 없다.**
(인간의 신체 부위를 뱀이나 4 족 보행 동물에 적용할 수 없는 것과 같은 이치다.)

이런 파츠가 여러 개 합쳐지게 되면, 뼈대 정보가 파츠 개수만큼 붙어버리기 때문에, 성능 부하가 커지게 된다. 파츠가 4 개인 경우, 파츠들을 그냥 불러서 하나의 객체 안에 다 넣고 애니메이션을 돌리게 되면, 눈에 보이지는 않아도 똑같은 애니메이션 뼈대 정보가 4 개가 같이 움직이게 된다. 이런 상황은 대단히 불필요한 낭비일 뿐더러, 프로그래밍 코드에서 객체 관리를 하기에도 어려워진다.

따라서 각 **파츠의 Skinning Mesh 를, 해당 파츠 리소스 파일에서 사용했던 애니메이션 뼈대 정보로부터, 실제 게임에서 사용할 애니메이션 뼈대 정보에 붙이는 과정이 필요**하다. 실제로 사용하는 애니메이션 뼈대 정보 한 개만 사용하기 위해, 파츠 리소스 데이터로부터 Skinning Mesh 만 떼어내서 이식한다는 의미이다.

- **B-26-4-3.** 부착형 파츠의 표현 (모델 붙이기)
: **무기와 일부 방어구들은 캐릭터 모델의 특정한 더미(Dummy) 객체에 붙이는 식으로 장착된 외형을 보여준다.**
: 이런 경우, 무기와 방어구의 모델이 스키닝 정보를 가질 필요는 없다. 보통 이런 경우는 대부분의 무기 모델과 방패 등의 일부 방어구가 대표적인 예가 된다.

- **B-26-4-4.** 재질 / 텍스처 교체
: **3D 모델의 표면 무늬나 그림 자체가 바뀜으로써, 장비 아이템이 교체되었음을 표시하는 경우가 가능해야 한다.**

◆ B-26-5. 파츠 정책

- **B-26-5-1.** 파츠들의 조합을 이용해서 캐릭터 모델을 표현하는 대상은 **오직 플레이어 캐릭터만을 대상**으로 한다.
: 모바일 기기를 대상으로 하는 게임은 성능 부담이 큰 파츠 시스템을 최대한 억제해야 한다.

※ 파츠를 사용하여 캐릭터를 표현할 경우, 성능에 부정적인 영향을 준다.

1) **Skinning Mesh를 여러 개 사용해야 한다.**
: 하나의 모델에 Skinning Mesh가 1개를 초과하면 성능이 급격하게 떨어진다. 적어도, 프로젝트의 기반으로 사용하는 Unity 엔진에서는 그러하다.

2) **애니메이션 모델의 기준이 될 뼈대(Bone)에 스키닝 정보를 합치는 과정이 필요하다.**
: 두 과정 모두, 새로 캐릭터 모델을 불러와야 할 경우, 추가적인 계산을 요구하기 때문에 성능 부하를 발생하게 한다.

- **B-26-5-2.** 플레이어 캐릭터가 아닌 캐릭터 개체들은 하나의 Skinning Mesh로 모든 신체를 모델링한다.

※ 여러 번 이야기한 대로, 성능에 대한 부담이 가장 큰 이유이다.
몬스터 객체들에게 일부 파츠를 교체하는 방식을 둠으로써 다양성을 연출해볼 욕심도 있을지 모르겠지만, 하드웨어 제약이 존재하기 때문에, 언제나 이상적인 방향으로만 작업할 수는 없다.
다만, 신체형 파츠가 아닌, 부착형 파츠를 통해 구분하는 경우는 고려해볼 수도 있다.
몬스터의 무기 부분은 부착형 파츠로 구현할 것이 분명하니 말이다.

- **B-26-5-3.** 플레이어 캐릭터의 구체적인 파츠의 구분은 관련 기획서의 요구사항을 따른다.

- **B-26-5-4.** 파츠 모델이 붙도록 지정되어 있는 부위에는, 원칙적으로 1 부위 당 1개의 파츠만 붙일 수 있다.
: 동일한 부위에 여러 개의 파츠가 붙을 수 없다.

- **B-26-5-5.** 파츠를 교환할 때는 이전 파츠와 1 대 1로만 교환한다.
: 하나의 파츠가 다른 여러 개의 파츠와 교환되거나, 혹은 그 반대로 작용하지 않게 디자인한다.

※ 파츠 교환이 1 대 1 이 아니라면, 파츠 교환 시스템의 작성이 극도로 복잡해진다.
각 파츠마다 교환 가능한 다른 파츠 부위와, 가려질 부분, 아닐 부분 등을 별도로 지정할 수 있어야 하기 때문이다.

하나의 캐릭터 모델이 여러 직업과 복식을 갖춰야 하는 MMORPG 라면 이러한 시스템이 필요하겠으나, 적어도 이 게임은 그렇지 않다. 캐릭터의 직업마다 모델은 정해져 있고, 장착 아이템의 외형 역시 혼용하지 않는다.

◆ B-26-6. 기본 파츠(맨몸 파츠)

- **B-26-6-1.** 캐릭터가 아무런 장착 아이템도 착용하지 않았을 때, 플레이어에게 시각적으로 보이게 해줄 모델을 표현하는 파츠들을 기본 파츠(Basic Parts, Default Parts), 혹은 맨몸 파츠(Naked Parts)라고 한다.

- **B-26-6-2.** 모든 캐릭터 모델들은 기본 파츠에 대한 모델을 소유해야 한다.

※ 기본 파츠가 반드시 벌거숭이 상태일 필요는 없다.
대부분의 몬스터들은 단 한 개의 기본 파츠만 가지고 있을 뿐, 다른 부위나 종류의 파츠들은 가지지 않을 것이다.

B-27. 위치 소켓

◆ B-27-1. 위치 소켓

- **B-27-1-1.** 게임 세계에서, **어떤 모델 객체를 특정 모델 객체의 부위에 따라다니게 만들어야 할 때, 그 상대적인 위치를 지정해주는 역할을 하는, 보이지 않는 객체**들을 위치 소켓 객체(Position Socket Object)라고 한다.

※ 가장 많이 쓰이는 곳은, FX 모델들을 붙이는 장소를 지정할 경우다.
예를 들면, 피격 FX 들은 보통 캐릭터의 몸통 중심부 쯤에 붙는다. 검을 휘두를 때의 섬광을 표현하는 FX 는, 그 캐릭터가 무기를 휘두르는 애니메이션을 수행했을 때, 무기의 끝 부분에 붙는다.

- **B-27-1-2.** 위치 소켓 객체는 그 자체로는 아무런 기능도 하지 않고, 눈에 보이지도 않는다.
: 그저 위치 정보만 담당한다.

- **B-27-1-3.** 위치 소켓은 그 위치 소켓이 어떤 목적으로 쓰이는지에 대한 정보를 가져야 한다.
: 예를 들면, 피격 이펙트가 붙는 자리인지, 이름 GUI가 붙는 자리인지, 그림자 FX가 붙는 자리인지 등에 대한 정보와 연결되어 있어야 한다.

◆ B-27-2. 파츠와 위치 소켓의 공통점과 차이점

- **B-27-2-1.** 파츠는 캐릭터 개체의 신체 일부, 혹은 장착 아이템을 표현하는 모델을 가리킨다.
그러나 위치 소켓 객체는 단지 캐릭터 개체에서 어디 쯤에 해당하는지에 대한 위치만 표현한다. 파츠처럼 겉으로 드러나는 어떤 모델은 없다.

- **B-27-2-2.** 파츠는 캐릭터 개체의 애니메이션 계통과 뼈대 정보의 위치가 애니메이션에 따라 바뀌는 흐름에 의존하는 경우가 대부분이다.
그러나 위치 소켓은 그러한 정보에 의존할 수도 있고, 아닐 수도 있다.

※ 파츠의 경우에도, 부착형으로 동작하는 파츠들은 사실 애니메이션과 관계 없이 상대적인 위치만 점유하고 있거나, 심지어 애니메이션을 따로 가지고 있을 수도 있다.
여기서의 설명은 일반적인 차이점을 말하고 있을 뿐이다.

- **B-27-2-3.** 둘 다 소유자인 캐릭터 개체에게 종속되어 있고, 캐릭터 개체의 위치가 변할 때마다 자신들의 위치와 회전 값들도 따라다니는 점은 같다.
  : 파츠든 위치 소켓이든, 소유자 캐릭터 개체를 벗어나서 존재하지 않는다.

※ '이론적으로는' 3차원 공간 상의 절대적인 위치를 차지하고 있어서, 그래서 소유자 캐릭터 개체와는 '비종속적으로' 동작하는 파츠나 위치 소켓의 개념과 구현이 불가능한 건 아니다.
하지만, 대개의 경우 현실적으로 그런 모델의 구조가 필요할 만한 경우가 없다시피 하다. 따라서, 이론적으로 가능하지만, 실질적으로 쓸 일이 없는 경우는 고려하지 않기로 한다.

◆ B-27-3. 위치 소켓의 관리

- **B-27-3-1.** 각 캐릭터 개체들마다 위치 소켓들의 목록과 역할을 관리하는 임무를 담당하는 컴포넌트를 소유한다.

- **B-27-3-2.** 각 캐릭터 개체들은 생성 및 초기화 단계에서, 자신들이 보유한 위치 소켓 객체들을 모두 찾아서 정리해놓는다.

※ 위치 소켓 객체를 미리 찾아놓는 이유는, 필요할 때마다 캐릭터 개체를 구성하는 객체들의 계통 구조를 헤집으면서 소켓 객체들을 찾아 다니는 작업을 매번 수행하면 무척 느리기 때문이다.
위치 소켓 객체들의 참조자들을 미리 획득해서 정리한 상태로, 나중에 필요할 때마다 모아둔 참조자들을 통해 검색 / 접근하면 훨씬 빠르고 간편한 코드로 위치 소켓이 필요한 작업을 수행할 수 있다.

- **B-27-3-3.** 위치 소켓 객체들은 위치 소켓 관리자 컴포넌트가 자신들을 찾을 수 있도록, 객체 자신의 이름에 특별한 이름 규칙을 지정한다.
  : 이름의 접두어로 'Socket_'을 붙이는 식으로...

※ 위치 소켓 객체들은 모델 데이터의 애니메이션 계통 정보에 매우 많이 의존해야 하기 때문에, 수많은 뼈대 객체 구조의 아주 깊숙한 곳에 처박혀 있어야 할 경우가 많다.
이런 상황에서 뼈대 객체들의 계통 구조를 일일이 헤집으면서 어떤 게 위치 소켓 객체인지 확인하려면, 이를 손쉽게 확인해 줄 어떤 구별법이 존재해야 한다. 가장 손쉬운 방법은, 객체의 이름에 위치 소켓임을 나타내는 특별한 명명법이 존재하는 것이다.
문자열 비교가 빠른 연산이라고 할 수는 없지만, 대부분 캐릭터 생성시 최초 1회만 문자열을 통한 위치 소켓 객체 찾기를 시도할 것이므로, 큰 문제가 안 된다.

- **B-27-3-4.** 프로그램에서는 각 위치 소켓의 역할 구분에 따라 상수 값을 정의해야 한다.
  : 캐릭터에 따라 그 캐릭터만 가지는 고유한 위치 소켓들도 존재할 수 있으므로, 위치 소켓의

타입을 나타내는 상수 값의 종류는 꽤 많을 수 있다.

- **B-27-3-5.** 위치 소켓 관리 컴포넌트에서 위치 소켓을 찾을 때는, 위치 소켓의 역할을 정의한 상수를 키(Key)로 삼아서 찾는다.
  : 위치 소켓은 관리 컴포넌트에서 찾을 때가 빈번하므로, 여기서는 느린 문자열로 검색하지 말고 미리 정의한 열거형 혹은 상수 값으로 검색한다.

B-28. 거래 / 교환 시스템

◆ B-28-1. 제한사항

- **B-28-1-1.** 실제 세계에서의 화폐(프로그램 내부에서는 Cash로 일반화해서 호칭한다.)는 오직 **게임 상의 과금 화폐(Game Cash Money로 일반화해서 호칭한다.)로만 교환**할 수 있어야 한다.

- **B-28-1-2.** **획득한 골드는 어떠한 경우에도 절대로 과금 화폐로 교환하지 못한다.**
  : 이 부분은 재화들의 가격 책정할 때 절대적으로 지켜져야 하는 사항이다.

※ 만약 이게 가능하다면, 게임 플레이에서 계속해서 쏟아져 나오는 게임 플레이 화폐들이 과금 화폐의 기능을 사실상 대체할 수 있고, 현실 세계의 화폐 결제를 통해 과금 화폐를 구입할 이유도 없어진다.
이는 수익 모델에 결정적인 영향을 미칠 수 있으므로 절대로 허용해서는 안 된다.

- **B-28-1-3.** 거래 / 교환은 상점 방문 -> 처리의 과정으로 표현한다.

- **B-28-1-4.** 사용자 대 사용자 간 거래는 지원하지 않는다.
  : 즉, 계정 단위로 재화나 아이템을 옮겨 다닐 수 없다.

- **B-28-1-5.** 같은 사용자의 계정에 속한 캐릭터들끼리도, 과금 화폐와 게임 플레이 화폐를 제외한 모든 재화는 공유하지 않는다.

- **B-28-1-6.** 여기서 서술하는 상점의 분류는 기능상의 분류일 뿐이다. 이를 GUI로 구현할 때, 실제로 어떻게 상점을 구성할지에 대해 분류한 것이 아니다.

※ 이 명세서에서는 분리된 형태의 상점처럼 서술했지만, 실제 UI에 구현할 때는 별도의 상점 UI를 제공하지 않고, 그냥 물품 보관함에서 할 수 있는 메뉴의 한 형태로 제공할 수도 있다.
여기서 결제 상점이나 뽑기 상점, 제작 상점 등을 나눴다고 해서, 게임에 실제로 구현하는 GUI들도 그러한 분류를 꼭 따라야 할 필요가 없다.
즉, 이 항목에서 서술한 상점의 각 기능들이 작동하도록 구현하되, 그러한 상점 기능을 표현하는 형태와 인터페이스는 **플레이어들이 가장 편리하게 사용할 수 있는 형태를 고안해서 구현**하면 된다.

◆ B-28-2. 결제 상점

- **B-28-2-1.** 결제 상점에서는 과금 화폐를 구입 및 과금 화폐를 통해서 직접 구매가 가능한 상품들의 판매를 취급한다.

- **B-28-2-2.** 과금 화폐를 구입할 수 있는 상점은 모든 플레이어가 언제나 접근이 가능하다.
  : 상설 상점이라는 의미이다.

- **B-28-2-3.** 결제 상점에서 취급하는 요소들은 다음과 같다.

종류	구매 화폐 단위	설명
과금 화폐 충전	현실 세계의 화폐	• 구매했을 때, 실제 화폐의 결제액을 토대로 VIP 점수를 올려준다. • 서비스 지역과 마켓마다 구매하는 화폐 단위가 달라질 수 있다.
게임 플레이 화폐 충전	과금 화폐	• 서비스 지역과 마켓에 관계 없이 구매하는 화폐 단위와 가격이 같다.
입장 자원 충전	과금 화폐	• 서비스 지역과 마켓에 관계 없이 구매하는 화폐 단위와 가격이 같다.

- **B-28-2-4.** 과금 화폐를 구입하는 횟수나 양에 대한 제한은, 게임 서비스 내에서는 원칙적으로 없다.

※ 하지만, 실제로는 현실의 화폐를 이용한 결제이기 때문에, 현실의 디지털 상품과 소액 결제와 관련된 제도와 법령의 영향을 받는다.
실정법의 제한 범위를 벗어나지 않는 한도 내에서 특별히 자체적인 제한을 가하지는 않는다는 뜻이다.

- **B-28-2-5.** 게임 플레이 화폐의 충전 횟수는 기본적으로 1일 1회로 제한한다.

- **B-28-2-6.** 게임 플레이 화폐의 충전 횟수는 특정 VIP 레벨을 달성할 때마다 늘어날 수 있다.

- **B-28-2-7.** 입장 자원의 충전 횟수는 기본적으로 1일 1회로 제한한다.

- **B-28-2-8.** 입장 자원의 충전 횟수는 특정 VIP 레벨을 달성할 때마다 늘어날 수 있다.

- **B-28-2-9.** 입장 자원의 충전은 **최대 입장 자원 개수와 상관 없이 계속해서 충전이 가능**하다.
  : 하지만, 최대 입장 자원 개수를 넘어섰다면, 입장 자원을 소모해서 최대 입장 자원 개수보다 낮아질 때까지는, 시간이 지남에 따라 충전되는 기능이 작동하지 않는다.

◆ B-28-3. 뽑기 상점

- **B-28-3-1.** 무작위 / 선택 뽑기에 대한 상품들을 판매하는 상점이다.

- **B-28-3-2.** 뽑기 1회 당 뽑기 권한 점수를 1개 소모한다.
  : 아이템 뽑기 1회에 뽑기권을 여러 개 소모하는 방식은 고려하고 있지 않다.

- **B-28-3-3.** 뽑기 명령의 GUI는 1회 수행과 특정 횟수를 연속해서 수행하는 방식이 공존한다.
  : 단, 연속 수행을 한다고 해서 가격이 더 저렴해지지는 않는다.

※ 한 번 터치해서 뽑기 1회 수행하는 버튼이 있고, 이게 가격이 과금 화폐 10개이면, 같은 동작을 10회 수행하는 버튼을 터치할 때 소모하는 비용은 과금 화폐 100개이다.
10회 수행한다고 해서 조금 더 깎아주거나 하는 법은 없다.
이건 그저 여러 차례 뽑기를 하는 사용자의 편의성을 위한 기능일 뿐이다.

- **B-28-3-4.** 뽑기를 연속 수행하는 메뉴에서, 수행 횟수 자체는 고정되어 있고, 플레이어가 연속 횟수를 조절하지는 못한다.

※ 버튼에 '10 연속 뽑기'라고 되어 있으면, 한 번 터치하면 10차례 연속으로 뽑기를 수행할 뿐, 10회 연속을 20회 연속이나 5회 연속으로 바꿀 수는 없다.

- **B-28-3-5.** 아이템 뽑기나 스킬 룬 뽑기의 연속적인 사용은 VIP 레벨이 일정 이상이어야 가능하도록 지정할 수 있다.

- **B-28-3-6.** 아이템이나 스킬 룬을 무작위로 뽑는 방식에 있어, **어떤 뽑기권을 사용하든 관계없이, 아이템의 등급 선택은 무작위하다.**
: 즉, 뽑기권의 등급에 의해서, 뽑히는 아이템의 등급이 정해지거나, 더 높은 확률로 더 상위 등급의 아이템이 등장하지 않는다.

- **B-28-3-7.** 뽑기를 통해 등장하는 아이템들은 모두 캐릭터 장비 슬롯에 장착할 수 있는 아이템들이다.
  : 뽑기 상점에서 소모품이나 제작 재료, 보석 등은 등장하지 않는다.

※ 정말로 '모든' 종류의 아이템이 등장하는 뽑기를 하는 상점은 무작위 뽑기 상점이다.

- **B-28-3-8.** 아이템 뽑기나 스킬 룬 뽑기를 해서 등장하는 아이템들은, **그 캐릭터의 직업이 쓸 수 있는 아이템만 등장**한다.

: 아이템 뽑기를 하는 캐릭터가 착용할 수 없는 아이템은 뽑을 수 없다.
스킬 룬의 경우에는, 현재는 캐릭터 직업 별 제한 사항은 없게 디자인하므로, 여기에 해당하지는 않는다. 물론, 향후 직업 별 고유 스킬 룬이 생긴다면 역시 동일한 조건을 적용한다.

※ 이러한 제약 사항은, '고급' 아이템을 직업 별로 얻고 싶으면 여러 캐릭터를 다양하게 키워볼 것을 유도하는 측면도 있지만, 그것보다도 사용자 편의의 이유가 더 크다.
아이템 뽑기이든, 스킬 룬 뽑기이든 과금 화폐를 지불해야 하는 컨텐츠인데, 직업 제한 때문에 당장 사용할 수도 없는 아이템이 등장하는 현상을 환영할 플레이어는 없을 것이기 때문이다.

- **B-28-3-9.** 제작하는 아이템에 부여되는 능력 옵션은, 제작을 수행하는 캐릭터의 직업에 생성할 수 있는 한계 범위 이내에서 고르도록 설계한다.

- **B-28-3-10.** 상점에서 취급하는 판매 목록과 가격 단위는 다음과 같다.

종류	구매 화폐 단위	설명
장비 아이템 뽑기 (일반, 장비 부위 전체)	과금 화폐 (비용 예 : 10)	• 무작위로 장착 부위 중 하나를 뽑는다. • 수행하는 캐릭터가 착용할 수 있는 아이템 중에서 등장함. • 등장하는 아이템의 능력 옵션은, 뽑기를 수행하는 캐릭터의 현재 레벨에서 생성 가능한 옵션으로 등장한다. • 이 과정으로 생성한 장비 아이템은 장착 가능한 요구 레벨의 제한이 있다.
장비 아이템 뽑기 (일반, 장비 부위별)	과금 화폐 (비용 예 : 30)	• 특정 장착 부위만 뽑는 티켓이 장착 부위 종류만큼 존재한다. • 수행하는 캐릭터가 착용할 수 있는 아이템 중에서 등장함. • 등장하는 아이템의 능력 옵션은, 뽑기를 수행하는 캐릭터의 현재 레벨에서 생성 가능한 옵션으로 등장한다. • 이 과정으로 생성한 장비 아이템은 장착 가능한 요구 레벨의 제한이 있다.
장비 아이템 뽑기 (고급, 장비 부위 전체)	과금 화폐 (비용 예 : 50)	• 무작위로 장착 부위 중 하나를 뽑는다. • 등장하는 아이템의 능력 옵션은, 뽑기를 수행하는 캐릭터의 현재 레벨과 상관 없이 결정된다. : 즉, 당장 게임 플레이에서 얻을 수 있는 것보다 더 높은 능력 옵션을 가진 아이템도 등장할 수 있다. • 이 과정으로 생성한 장비 아이템은 장착 가능한 요구 레벨의 제한이 없다. : 물론, 직업 제한은 유지한다.
장비 아이템 뽑기	과금 화폐	• 특정 장착 부위만 뽑는 티켓이 장착 부위 종류만큼 존

(고급, 장비 부위별)	(비용 예 : 150)	재한다. • 등장하는 아이템의 능력 옵션은, 뽑기를 수행하는 캐릭터의 현재 레벨과 상관 없이 결정된다. : 즉, 당장 게임 플레이에서 얻을 수 있는 것보다 더 높은 능력 옵션을 가진 아이템도 등장할 수 있다. • 이 과정으로 생성한 장비 아이템은 장착 가능한 요구 레벨의 제한이 없다. : 물론, 직업 제한은 유지한다.

- **B-28-3-11.** 스킬 룬의 뽑기는 아래와 같은 종류가 있다.

종류	구매 화폐 단위	설명
무작위	과금 화폐 (비용 예 : 10)	• 무작위한 스킬 룬의 타입 중에서 하나가 뽑힌다.
타입 선택	과금 화폐 (비용 예 : 30)	• 특정한 스킬 룬 타입의 범위에서만 뽑는 티켓이, 스킬 룬 타입의 종류만큼 존재한다.

※ 스킬 룬은 일반 – 고급 뽑기의 구분이 존재하지 않는다.
스킬 룬이 가지는 능력의 종류로는 상 / 하위 구분이 없고, 같은 능력의 세기에 비례해서 스킬 룬의 등급이 정해지기 때문이다.
그러니까, 스킬 룬에 들어가는 속성이 '대상을 얼리는지'와 '스킬 사용시 이동 속도 증가'인지 여부는 우열을 가리는 관계가 아니다. 하지만 똑 같은 '대상을 얼리는' 능력에서는, 얼리는 시간이 1초인지 3초인지에 따라 그 룬의 등급이 달라진다는 말이다.

◆ B-28-4. 아이템 마법 부여 상점

- **B-28-4-1.** 일정한 비용을 내고 장비 아이템의 능력 속성을 교체할 수 있다.

- **B-28-4-2.** 마법 부여는 장비하는 아이템에만 가능하다.
: 장착할 수 없는, 물품 보관함에만 존재하는 소모품 등의 아이템에는 마법 부여를 할 수 없다.

- **B-28-4-3.** 마법 부여가 가능한 능력 옵션들은 아이템의 **주 능력을 제외한 부가 옵션**들이다.
: 아이템의 주 능력 옵션은 어떤 경우에도 다른 옵션으로 교체할 수 없다.

※ 무기는 공격력이 주 능력이고, 방어구는 방어력이 주 능력이다.
만약 주 능력이 교체가 가능하다면, 공격력이 전혀 존재하지 않는 무기도 가능해야 한다는 의미이기 때문에, (이론상으로는 가능하다 해도) 게임 플레이의 기본 개념상 오류가 된다.
따라서, 주 능력의 종류는 어떤 경우에도 교체하지 않으며, 그 외 부가적으로 붙는 능력 옵션

에 대해서만 교체가 가능하도록 전제하고 있다.

- **B-28-4-4.** 마법 부여는 **마법 등급** 이상의 아이템에만 가능하다.
: 주 능력 외의 부가 능력 옵션이 최소한 1개는 있어야 마법 부여의 대상이 된다.

※ 일반 등급의 아이템은 주 능력 밖에 없으므로, 아예 교체할 수 있는 부가 능력 옵션이 없다.
주 능력의 교체는 이 게임의 플레이 개념 상으로 오류이기 때문에, 당연히 마법 등급 이상의 아이템만 마법 부여가 가능할 수 밖에 없다.

- **B-28-4-5.** 마법 부여의 종류는 다음과 같다.

종류	구매 화폐 단위	설명
무작위	과금 화폐 (비용 예 : 10)	• 아이템의 부가 능력 옵션의 전체를 교체한다.
선택	과금 화폐 (비용 예 : 30)	• 플레이어는 마법 부여를 하는 아이템에 이미 존재하는 능력 속성들 중 1 개를 다른 능력 속성으로 교체하도록 선택할 수 있다. • 단, 새로운 능력 속성은 무작위하게 부여하며, 플레이어가 이를 선택할 수는 없다. • 플레이어가 마법 부여를 하기 위한 능력 속성의 항목을 선택했으면, 반복해서 선택적인 마법 부여를 하더라도, 이미 선택한 능력 속성의 항목만을 반복적으로 교체할 수 있다. • 이미 선택한 능력 속성의 항목 외, **다른 능력 속성의 항목들은 더 이상 건드릴 수 없다.**

- **B-28-4-6.** 마법 부여의 종류와 관계없이, 마법 부여로 교체하는 능력 옵션들은, **대상 아이템이 가질 수 있는 보물 레벨의 능력 옵션들 중에서 무작위하게 고른다**.
: 플레이어에게는 특정한 옵션이 등장하도록 통제할 수 있는 수단이 없다.

- **B-28-4-7.** 마법 부여의 종류와 관계없이, 마법 부여를 통해 새롭게 부여하는 속성들은 **그 아이템이 원래 가지고 있던 속성 부여의 허용 범위 안에서 결정**된다.

※ 위 사항들은 아래와 같은 경우가 발생할 수 있기 때문에 필요하다.
저레벨일 때 희귀 아이템을 하나 획득했는데, 그 아이템을 나중에 캐릭터가 고레벨이 된 이후에 마법 부여를 시도한다고 가정하자.
비록 캐릭터는 고레벨이 되었지만, 획득했던 희귀 아이템은 획득 당시에는 저레벨이 착용할 수 있는 낮은 능력치로 설정되어 나왔다. 즉, 기술적으로 정확하게 표현하자면, 보유할 수 있는 능력 옵션의 보물 레벨(Treasure Level)이 낮았다.
그렇다면 고레벨의 캐릭터가 나중에 마법 부여를 시도하더라도, 등장할 수 있는 능력 옵션의

보물 레벨이 낮은 아이템은 캐릭터의 현재 레벨이나 강력함에 상관 없이, 그 아이템에 명시되어 있는 낮은 보물 레벨의 능력 옵션들 중에서 교체할 옵션을 고른다.
각 (장비)아이템들은 그 아이템이 생겨나는 단계에서, 이미 그 아이템의 근본적인 잠재적 한계를 가지고 태어나는 셈이다. 그러니, 높은 능력치를 얻을 가능성이 있는 마법 부여를 하고자 한다면, 기본적으로 생성 당시에 높은 보물 레벨의 능력 옵션을 가질 수 있는 아이템을 대상으로 해야 한다.

- **B-28-4-8.** 선택 마법 부여는 **희귀 등급** 이상의 아이템에만 가능하다.
: 마법 등급의 아이템은 부가 능력 옵션이 1개만 붙기 때문에, '선택'을 할 수 없다. 따라서 부가 능력 옵션이 2개 이상인 등급의 아이템들만 선택적인 마법 부여를 할 수 있다.

※ 마법 등급의 아이템은 부가 능력 옵션이 1개 뿐이므로, 무작위 마법 부여와 선택 마법 부여가 결과적으로 같은 동작을 한다.
그런데, 선택 마법 부여는 사용자에게 선택권을 주기 때문에 가격이 훨씬 높게 책정되어 있다.
즉, 마법 등급 아이템들은 같은 결과를 내는 두 동작에 대해 서로 다른 값을 지불할 수도 있게 되는 셈이다!
이런 상황의 모순을 막기 위해, 선택적 마법 부여는, 능력 옵션이 2개 이상 붙을 수 있는 등급의 아이템들만 대상으로 하도록 한다.

- **B-28-4-9.** 마법 부여로 **셋 아이템의 아이템 셋 부가 옵션 효과들을 다른 옵션 효과로 교체할 수 없다.**

※ 셋 아이템의 아이템 셋 옵션 효과는 아이템의 장비 상황에 따라 가변적이기 때문에, 일반적인 아이템의 능력 옵션처럼 아이템마다 고정적으로 붙지 않고, 사전에 옵션 목록이 정해져 있다.
이런 옵션 목록은 무작위로 다른 옵션으로 교체하는 방식을 쓸 수 없고, 그렇게 한다 해도 전혀 의미가 없다. '좀 더 좋은 옵션'이라는 게 없기 때문이다.
아이템 셋의 옵션을 변경할 수 있다면, 이를 통해 '최종적으로 가장 효율적인' 셋 아이템 시리즈만 살아남게 되므로, 게임 밸런스와 플레이의 다양성에도 영향을 많이 주게 된다.

- **B-28-4-10.** 마법 부여는 한 번에 1회만 가능하다.
: 한 번의 UI 이벤트로 연속적으로 마법 부여를 할 수는 없다.

※ 마법 부여라는 게, 본래 사용자가 한 번 시도한 뒤, 결과를 보고 판단해야 하는 과정이므로, 연속적인 사용을 가능하게 할 필요가 없다. 그러므로, 연속 사용을 가능하게 하는 UI도 제공할 필요가 없다.

- **B-28-4-11.** 같은 장비 아이템에 마법 부여를 할 때마다, 다음 번 마법 부여에 들어가는 비용이 증가한다.
: 마법 부여의 종류(무작위, 선택)와 상관 없이 이 원칙이 적용된다.

※ 이 제약사항이 반드시 필요한 이유는, 아이템 뽑기의 존재 때문이다.
마법 부여는 이미 사용자가 보유한 장비 아이템 내부의 옵션을 바꾸기 때문에, 이미 장비의 등급과 부위 타입을 고정한 조건에서의 교체라고 할 수 있다. 이 상태에서 같은 비용으로 연속해서 능력 옵션을 교체할 수 있다면, 장기적으로 마법 부여는 아이템 뽑기보다 항상 유리한 조건이 되어 버린다.
극단적으로, 최고 등급인 전설 등급의 아이템을 하나라도 획득한 사용자는 그 부위의 장비 아이템에 대해서는 더 이상 아이템 뽑기를 해야 할 이유가 없어져 버린다! 아이템 뽑기는 아이템 등급이 전설 등급이 나올지 마법 등급이 나올지 알 수 없기 때문이다. 나머지 부가 능력 옵션 전체를 교체하는 건 아이템 뽑기나 무작위 마법 부여나 같으니, 결국 무작위 마법 부여가 무작위 아이템 뽑기보다 항상 유리한 조건이다.
위와 같은 결과를 방지하기 위해서는, 같은 아이템에 마법 부여를 하는 비용을 계속해서 늘려서, 일정한 횟수 이상으로 마법 부여를 하지 못하도록 제약을 둬야 한다. 그래야 그 비용이 부담스러운 사용자들은 아이템 뽑기로도 선택지를 돌릴 것이기 때문이다.

※ 아니면, 아이템 마법 부여를 처음부터 제한된 횟수만 가능하게 하는 방식도 고려할 수 있다.
일정 횟수 이상 마법 부여를 한 아이템은 더 이상 마법 부여를 할 수 없게 만드는 방식이다.

- **B-28-4-12.** **같은 아이템에 무작위 마법 부여 방식과 선택적인 마법 부여 방식을 섞어 쓸 수 없다.**
: 최초로 마법 부여를 할 때, 둘 중 한 가지 방식을 사용했다면, 그 다음 마법 부여는 최초에 선택했던 방식만을 계속 사용해야만 한다.

※ 이는 선택적 마법 부여를 했던 아이템이 나중에 무작위 마법 부여를 하는 경우보다, 반대의 경우(무작위 마법 부여 -> 선택적 마법 부여)가 문제가 되기 때문이다.
위 제약 사항이 지켜지지 않는다면 다음과 같은 상황이 가능하다.
무작위 마법 부여를 수행했는데, 능력 옵션 중 딱 한 가지가 마음에 들지 않는다면, 선택적 마법 부여를 통해 그 옵션만 계속해서 마음에 드는 능력 옵션이 등장할 때까지 마법 부여를 한다.

사실, 원래 의도는 이러한 '완벽한 능력 옵션으로 조합된' 아이템이 등장할 확률이 극히 드물어야 한다는 점이다. 완벽한 능력을 가진 장비 아이템이 등장한다면, 플레이어는 다시 그 종류의 장비 아이템을 획득하려고 노력을 할 필요가 없어지기 때문이다.
아이템의 마법 부여 방식을 섞어서 쓸 수 있다면, 위와 같이 아이템의 능력 옵션들의 조합을 플레이어가 의도한대로 조정할 수 있는 여지(꼼수)를 주는 셈이다.
이는 마법 부여 시스템의 목적 의도에 벗어나는 방식이기 때문에, 허용하지 말아야 한다.

- **B-28-4-13.** 한 번 수행한 마법 부여는 취소할 수 없다.
: 한 번 마법 부여를 수행하면, 아이템의 능력 옵션이 교체되며, 다시 이전 상태로 되돌아가지 못한다.

※ 만약 수행한 마법 부여를 되돌이기는 게 가능하다면, 다음과 같은 현상이 발생할 수 있다.
어떤 장비 아이템에 마법 부여를 했는데, 마음에 들지 않는 능력 옵션들의 조합이 나왔다.
그러면 플레이어는 그 능력 옵션들의 조합 결과를 '미리 보고', 취소한 뒤, 제작에 들어간 금액과 재료들을 돌려받는다. 그리고 다시 아이템 마법 부여를 시도한다. 그러면 상위 등급의 다른 능력 옵션들의 조합이 등장한다. 이러길 여러 번 반복해서 가장 최적의 능력 옵션을 가진 상위 등급 아이템으로 제작한다.
이런 방식이면 **아이템 뽑기 기능은 전혀 쓸모가 없어지는 셈**이다.
왜냐하면 본래 아이템 마법 부여 시스템에서는 무작위로 능력 옵션들을 조합해야 하는데, 이 과정이 중간에 취소 가능하다면 아무런 실제적 비용도 들이지 않고 무한정 옵션 조합들을 대입할 수 있는 꼼수가 되기 때문이다.

◆ B-28-5. 장비 아이템 제작 상점

- **B-28-5-1.** 하위 등급의 장비 아이템을 상위 등급의 아이템으로 합성할 수 있다.
: 이 과정을 아이템 제작으로 일컫는다.

- **B-28-5-2.** 제작을 수행하는 캐릭터의 직업이 장착할 수 있는 장비 아이템만을 제작할 수 있다.
: 다른 캐릭터 직업이 사용하는 장비 아이템은, 그 직업을 가진 캐릭터로 제작을 수행해야 만들 수 있다.

- **B-28-5-3.** 제작하는 장비 아이템은 등급을 한 번에 2단계 이상 뛰어넘을 수 없다.
: 한 번의 제작은 1단계의 등급 상승만 가능하다.

- **B-28-5-4.** **아이템 제작은 한 번 수행하면 소모한 비용과, 제작된 결과를 취소할 수 없다.**
: 한 번 제작 명령을 내리면, 그대로 제작이 완료되며, 다시 이전 상태로 되돌아가지 못한다.

※ 만약 수행한 아이템 제작을 되돌이기는 게 가능하다면, 다음과 같은 현상이 발생할 수 있다.
어떤 하위 등급 장비 아이템을 원판 재료로 해서 제작을 했는데, 마음에 들지 않는 능력 옵션들의 조합이 나왔다.
그러면 플레이어는 그 능력 옵션들의 조합 결과를 '미리 보고', 취소한 뒤, 제작에 들어간 금액과 재료들을 돌려받는다. 그리고 다시 아이템 제작을 시도한다. 그러면 상위 등급의 다른 능력 옵션들의 조합이 등장한다. 이러길 여러 번 반복해서 가장 최적의 능력 옵션을 가진 상위 등급 아이템으로 제작한다.
이런 방식이면 재료와 게임 플레이 화폐만 충분하다면 **아이템 뽑기 기능은 전혀 쓸모가 없어지는 셈**이다.
왜냐하면 아이템 제작 시스템에서는 다음 등급의 아이템을 만들어주기 위해서 무작위로 능력 옵션들을 조합해야 하는데, 이 과정이 중간에 취소 가능하다면 아무런 실제적 비용도 들이지

않고 무한정 옵션 조합들을 대입할 수 있는 꼼수가 되기 때문이다.

◆ B-28-6. 보석 세공 상점

- **B-28-6-1.** 보석과 관련된 상점이지만, 여기서는 보석을 판매하지 않는다.
  : 보석 자체는 게임 스테이지를 플레이하면서 획득해야 한다.
  (하급 보석에 한해서 판매를 허용하는 건 어떨지 모르겠다...)

※ 현재는, 보석 세공 상점에 대한 기능을 별도의 상점 GUI를 두는 방식이 아니라, 물품 보관함에서 각 장비아이템의 세부 항목 보기 GUI의 한 메뉴로써 동작하게 디자인하고 있다.

- **B-28-6-2.** 낮은 단계의 보석들을 합쳐서, 상위 단계의 보석으로 만들 수 있다.

- **B-28-6-3.** 이 과정에서 하위 등급 보석 3개와 게임 플레이 화폐를 소모해야 한다.

등급	제작 비용(예시)
1 단계	없음
2 단계	같은 종류의 1단계 보석 × 3, 게임 플레이 화폐 1,000
3 단계	같은 종류의 2단계 보석 × 3, 게임 플레이 화폐 3,000
4 단계	같은 종류의 3단계 보석 × 3, 게임 플레이 화폐 5,000
5 단계	같은 종류의 4단계 보석 × 3, 게임 플레이 화폐 10,000
6 단계	같은 종류의 5단계 보석 × 3, 게임 플레이 화폐 20,000
7 단계	같은 종류의 6단계 보석 × 3, 게임 플레이 화폐 40,000
8 단계	같은 종류의 7단계 보석 × 3, 게임 플레이 화폐 80,000
9 단계	같은 종류의 8단계 보석 × 3, 게임 플레이 화폐 100,000
10 단계	같은 종류의 9단계 보석 × 3, 게임 플레이 화폐 250,000

- **B-28-6-4.** 보석 장착 홈이 있는 장비 아이템에는 보석을 장착할 수 있다.
  : 보석 장착 홈이 없으면, 보석을 장착할 수 없는 아이템이다.

- **B-28-6-5.** 장비 아이템에 보석을 장착할 때는 별도의 비용이 들지 않는다.

- **B-28-6-6.** 장비 아이템에 한 번 장착한 보석은 임의로 제거할 수 없고, 별도로 비용을 내야 한다.

- **B-28-6-7.** 장착한 보석을 제거하는 비용은 게임 플레이 화폐만 사용한다.
  : 게임 플레이 화폐를 소모하는 대표적인 컨텐츠 중 하나다.

- **B-28-6-8.** 장착한 보석을 해제하는 비용은, 보석을 장착한 아이템의 등급과, 장착한 보석의 등급에 비례하여 점점 더 비싸진다.

보석 등급	일반 아이템	마법 아이템	희귀 아이템	전설 아이템
1 단계	500	600	750	1,000
2 단계	700	900	1,100	1,500
3 단계	1,000	1,200	1,500	2,500
4 단계	1,500	1,800	2,500	4,000
5 단계	2,200	2,750	4,000	7,000
6 단계	3,000	4,000	7,000	12,000
7 단계	5,000	7,000	12,000	25,000
8 단계	10,000	12,000	25,000	45,000
9 단계	20,000	25,000	45,000	75,000
10 단계	40,000	45,000	75,000	120,000

※ 간단한 수식 개념으로 정의하자면, 대략 다음과 같은 관계가 된다.
(표에 있는 가격 수치는 단지 예시일 뿐이다.)

보석의 장착 해제 비용 = 장착한 아이템의 등급 별 가격 × 장착한 보석의 등급 별 가격

◆ B-28-7. 장비 아이템 / 스킬 형상 판매 상점

- **B-28-7-1.** 비용을 지불하면, 장비 아이템이나 스킬의 형상을 바꿀 수 있는 기회를 준다.

- **B-28-7-2.** 형상 변환을 하는 요금은 과금 화폐로 매긴다.

- **B-28-7-3.** 형상 변환을 1회 시도할 때마다 요금을 지불해야 한다.
: 동일한 아이템이 여러 번 형상 변환을 시도한다면, 시도한 횟수만큼 비용을 내야 한다.

- **B-28-7-4.** 아이템 형상 변환의 종류는 다음과 같다.

종류	화폐 단위	설명
일반	과금 화폐 (비용 예 : 10)	• 아이템의 형상을, 해당 아이템에 맞는 형상들 중에서 무작위 선택해서 바꾼다. • 단, 일반적인 게임 플레이를 통해 획득할 수 있는 형상들 중에서만 무작위로 뽑는다.

고급	과금 화폐 (비용 예 : 50)	• 아이템의 형상을, 해당 아이템에 맞는 형상들 중에서 무작위 선택해서 바꾼다. • 일반적인 게임 플레이를 통해 획득할 수 있는 형상들 외에도, 형상 변환을 통해서만 획득할 수 있는 아이템 형상들도 무작위 뽑기 범위 안에 포함한다.
지정	과금 화폐 (비용 예 : 100)	• 변환할 장비 아이템의 형상을 플레이어가 직접 선택할 수 있다. • 선택할 수 있는 형상의 범위는 고급 형상 변환의 범위와 같다. • 사용자의 VIP 레벨이 특정 단계에 도달한 이후에 이용이 가능하다.

※ 형상 변환할 때, 특정한 형상으로 변환하는 과정에서, 형상의 종류에 따라 별도의 요금을 더 지불하지는 않는 방식으로 설계하기로 하였다.
하지 않기로 결정은 했지만, 만약 위 내용을 적용한다면, **형상의 종류에 따라서 차등적인 가격을 매길 수 있다.** (보통 스테이지에서 등장하는 형상들은 낮은 가격으로 매긴다거나, 여름 한정으로 제작한 비키니 수영복 스타일은 높은 가격으로 매긴다거나...;)
오직 형상 변환권 그 자체만을 판매한다면, 이렇게 형상마다 가격을 차별화할 수는 없을 것이다.

- **B-28-7-5.** 한 번에 하나의 아이템, 하나의 스킬에만 형상 변환을 시도할 수 있다.

- **B-28-7-6.** 아이템의 형상 변환은 **장착 가능한 아이템(Wearable Item)**만 가능하다.
: 파츠 모델을 가지고 있는 아이템만 형상 변환이 가능하다. 장신구(Accessory)에 속하는 장비 아이템들은 물품 보관함 내에서, 아이콘 형태로만 존재하기 때문에, 형상 변환의 대상이 아니다.
(장신구와 같은 경우, 아이콘만 바꾸는 형상 변환도 허용할지 생각을 해봐야겠다.)

- **B-28-7-7.** 플레이어가 형상 변환을 실제로 수행하기 전에, 형상 변환으로 외형이 바뀐 결과를 먼저 확인할 수 있어야 한다.
: 플레이어가 형상 변환으로 바뀐 외형을 확인하고, 형상 변환을 실제로 적용하기로 결정하면 그 때 비용이 지불된다.
만약, 미리 보기 기능으로 확인한 뒤, 형상 변환을 적용하지 않고 취소한다면, 대상 아이템의 형상 변환은 취소되고, 원래의 외형으로 되돌아간다.

※ 형상 변환은 외형과 관련된 기능이기 때문에, 미리 결과를 볼 수 있는 기능을 제공해줘야 합리적이다.

- **B-28-7-8.** 같은 아이템, 혹은 스킬에 대한 형상 변환이 여러 번 이루어졌다면, 가장 마지막에 수행한 형상만 남는다.

- **B-28-7-9.** 이전에 변환했던 형상이라고 할지라도, 다음에 다른 형상으로 바꾼 뒤, 다시 원래의 형상으로 변경하려고 한다면 다시 비용을 지불해야 한다.

※ 즉. 플레이어가 형상 변환에 대해 지불한 금액은, **선택한 아이템이나 스킬이 현재 사용할 형상을 쓸 수 있는 권한을 주는 것이지, 그 형상 자체를 언제든 바꿔서 사용할 수 있도록 소유권을 주는 개념이 아니다.**
이전의 구매 여부와 관계없이, **어떤 형상이든 현재의 형상과 다른 형상으로 바꾸기 위해서는 비용을 지불**해야 한다.

- **B-28-7-10.** 스킬 이펙트의 형상을 바꿀 수 있는 기능은 향후 업그레이드 컨텐츠이다.

◆ B-28-8. 숨겨진 상점(암시장?)

- **B-28-8-1.** 이 상점은 상설 상태가 아니며, 무작위한 날과 시간에 나타난다.

- **B-28-8-2.** 사용자 계정의 VIP 레벨이 일정 이상 높으면, 이 상점은 항상 등장하는 상태로 바뀔 수 있다.

- **B-28-8-3.** 숨겨진 상점에 등장할 수 있는 재화들의 조건은 다음과 같다.

• 보관함에 보관할 수 있는 종류의 아이템 재화만 등장한다.
: 과금 화폐, 게임 플레이 화폐, 입장 자원, 뽑기 횟수 등 수치를 축적하는 방식의 재화들은 등장할 수 없다.

• 등장하는 장비 아이템의 등급은 마법 등급 이상이다.
: 일반 등급의 장비 아이템은 등장하지 않는다. (일반 아이템은 별로 의미가 없기 때문이다.)

• 등장하는 아이템들은, 상점에 입장한 캐릭터가 사용할 수 있는 아이템들만 등장한다.

• 등장하는 아이템과, 그 아이템의 능력 속성들은 상점에 입장한 캐릭터의 현재 레벨에 맞춰서 등장한다.

- **B-28-8-4.** 숨겨진 상점에서 등장하는 아이템들의 가격은 과금 화폐와 게임 플레이 화폐 중 하나의 단위를 이용해서 매긴다.
: 둘 중 한 가지 화폐 단위로만 사용해야 한다.

※ 등장한 아이템의 가격이 큐빅 10 + 골드 1,000 과 같은 방식일 수 없다.
어떤 아이템은 큐빅 50, 다른 아이템은 골드 5,500 처럼, 가격을 매기는 화폐 단위는 둘 중 하

나로만 표기해야 한다.

- **B-28-8-5.** 숨겨진 상점이 상설적으로 등장하는 상태가 아닌 경우, 한 번 등장한 상점은 실제 시간으로 1 시간(60분)동안 유지한다.
: 남은 시간을 카운트다운 알림으로 알려준다.

- **B-28-8-6.** 숨겨진 상점이 상설적으로 등장하는 상태가 아닌 경우, 한 번 숨겨진 상점이 등장했다면, 그 다음 날 초기화 시점이 되기 전까지는 더 이상 숨겨진 상점이 등장하지 않는다.
: 서비스 지역의 표준시로 매일 오전 6:00에 이 조건을 초기화한다.

◆ B-28-9. 초월 던전 상점

- **B-28-9-1.** 초월 던전을 완료하면 초월 모드 코인을 획득할 수 있는데, 이 재화로 구매를 할 수 있는 상점이다.

- **B-28-9-2.** 초월 던전에 진입할 수 있는 레벨에 도달하지 못한 캐릭터는 이 상점에 접근할 수 없다.

- **B-28-9-3.** 초월 던전 상점에 등장할 수 있는 재화들의 조건은 다음과 같다.

• 보관함에 보관할 수 있는 종류의 아이템 재화만 등장한다.
: 과금 화폐, 게임 플레이 화폐, 입장 자원, 뽑기 횟수 등 수치를 축적하는 방식의 재화들은 등장할 수 없다.

• 등장하는 장비 아이템의 등급은 마법 등급 이상이다.
: 일반 등급의 장비 아이템은 등장하지 않는다. (일반 아이템은 별로 의미가 없기 때문이다.)

• 등장하는 아이템들은, 상점에 입장한 캐릭터가 사용할 수 있는 아이템들만 등장한다.

• 등장하는 아이템과, 그 아이템의 능력 속성들은 상점에 입장한 캐릭터의 현재 레벨에 맞춰서 등장한다.

- **B-28-9-4.** 초월 던전 상점에 등장하는 재화들의 가격은 초월 던전 코인 단위로만 매긴다.
: 다른 화폐 단위로는 구매할 수 없다.

◆ B-28-10. 결투장 상점

- **B-28-10-1.** 결투장에서 획득한 결투장 코인으로 구매를 할 수 있는 상점이다.

- **B-28-10-2.** 결투장에 진입할 수 있는 레벨에 도달하지 못한 캐릭터는 이 상점에 접근할 수 없다.

- **B-28-10-3.** 결투장 상점에 등장할 수 있는 재화들의 조건은 다음과 같다.

• 보관함에 보관할 수 있는 종류의 아이템 재화만 등장한다.
: 과금 화폐, 게임 플레이 화폐, 입장 자원, 뽑기 횟수 등 수치를 축적하는 방식의 재화들은 등장할 수 없다.

• 등장하는 장비 아이템의 등급은 마법 등급 이상이다.
: 일반 등급의 장비 아이템은 등장하지 않는다. (일반 아이템은 별로 의미가 없기 때문이다.)

• 등장하는 아이템들은, 상점에 입장한 캐릭터가 사용할 수 있는 아이템들만 등장한다.

• 등장하는 아이템과, 그 아이템의 능력 속성들은 상점에 입장한 캐릭터의 현재 레벨에 맞춰서 등장한다.

- **B-28-10-4.** 결투장 상점에 등장하는 재화들의 가격은 결투장 코인 단위로만 매긴다.
: 다른 화폐 단위로는 구매할 수 없다.

◆ B-28-11. 길드 상점

- **B-28-11-1.** 길드 활동을 통해 얻을 수 있는 **길드 코인**으로 구매할 수 있는 희귀한 아이템들을 판매한다.

- **B-28-11-2.** 길드에 가입하지 않은 캐릭터는 이 상점에 접근할 수 없다.

- **B-28-11-3.** 길드 상점에 등장할 수 있는 재화들의 조건은 다음과 같다.

• 보관함에 보관할 수 있는 종류의 아이템 재화만 등장한다.
: 과금 화폐, 게임 플레이 화폐, 입장 자원, 뽑기 횟수 등 수치를 축적하는 방식의 재화들은 등장할 수 없다.

• 등장하는 장비 아이템의 등급은 마법 등급 이상이다.
: 일반 등급의 장비 아이템은 등장하지 않는다. (일반 아이템은 별로 의미가 없기 때문이다.)

• 등장하는 아이템들은, 상점에 입장한 캐릭터가 사용할 수 있는 아이템들만 등장한다.

• 등장하는 아이템과, 그 아이템의 능력 속성들은 상점에 입장한 캐릭터의 현재 레벨에 맞춰서 등장한다.

- **B-28-11-4.** 길드 상점에 등장하는 재화들의 가격은 길드 코인 단위로만 매긴다.
: 다른 화폐 단위로는 구매할 수 없다.

◆ B-28-12. 무작위 뽑기 상점

- **B-28-12-1.** 접속한 플레이어가 비용을 내면, 무작위로 아무 재화나 뽑아서 주는 방식의 상점이다.
: 일종의 **복권**과 같은 역할을 하는 상점이다.

- **B-28-12-2.** 무작위 뽑기 상점에 등장하는 재화들은 어떤 재화든 조건 없이 등장할 수 있다.

※ 보관함에 보관할 수 없는 재화들, 그러니까 과금 화폐나 게임 플레이 화폐, 입장 자원 등도 여기에서는 등장할 수가 있다.
물론, 그런 아이템들은 획득하는 즉시 수치가 축적될 뿐, 물품 보관함에 별도로 보관하지 않는다는 사실은 변하지 않는다.

- **B-28-12-3.** 무작위 뽑기 상점에서 비용으로 사용하는 화폐는 과금 화폐이다.
: 다른 단위의 화폐는 사용하지 않는다.

- **B-28-12-4.** 뽑기 1회 수행하는 UI와, 10회 연속 수행하는 UI를 둔다.

- **B-28-12-5.** 무료로 뽑는 기회를 하루에 5회 제공한다.

- **B-28-12-6.** 하루에 제공하는 무료 뽑기 기회는 축적되지 않으며, 24시간이 지나면 다시 초기화한다.

- **B-28-12-7.** 무료 뽑기를 초기화하는 시간은, 서비스하는 지역의 표준시로 오전 6:00에 초기화한다.

- **B-28-12-8.** 무작위 상점의 뽑기를 무료로 수행하는 경우에는, 1회 수행할 때마다 실제 시간으로 재사용 대기 시간이 존재한다.

- **B-28-12-9.** 과금 화폐로 결제해서 뽑기를 진행하는 경우에는 뽑기를 수행하는 데 재사용 대기

시간을 두지 않는다.

- **B-28-12-10.** 각 뽑기를 수행할 때, 등장 가능한 재화와 아이템들의 목록은 뽑기가 1회 시행될 때마다 초기화해야 한다.
: 같은 등장 후보군 중에서 여러 번 고를 수 없다. 이는 10 연속 시행하는 명령에서도 마찬가지로 적용해야 한다.
모든 무작위 뽑기는 각 뽑기가 독립적인 시행이어야 한다.

※ 뽑기를 수행할 때, 등장 가능한 재화 및 아이템의 후보군을 보여주는 방식으로 무작위 뽑기를 수행할 수가 있다. 이에 대한 대표적인 형태가 룰렛을 돌리는 방식으로 무작위 뽑기를 수행하는 경우다.
이와 같은 방식으로 무작위 뽑기를 구현하는 경우, 무작위 뽑기가 1회 수행될 때마다 **룰렛에 등장하는 재화 및 아이템의 후보군 자체가 초기화**되어야 한다. 룰렛의 등장 후보군을 그대로 둔 채로, 여러 차례 뽑기를 수행할 수 없다.

B-29. 커뮤니티

◆ B-29-1. 채팅

- **B-29-1-1.** 플레이어가 자신의 캐릭터를 보기 위해 접속하는 로비가 있는 서버와 채팅 서버는 물리적으로, 혹은 별도의 프로세스로 구분되어 있어야 한다.
  : 이는 채팅 서버의 네트워크 교통량 때문에 캐릭터 로비에 대한 접속까지 장애가 생기지 않도록 하기 위함이다.

- **B-29-1-2.** 채팅 메시지의 폰트는 **동적 폰트(Dynamic Font)를 사용**한다.
  : Bitmap 기반의 정적 폰트는 속도가 더 빠르지만, 폰트를 위한 텍스처들을 미리 준비해야 하기 때문에, 메모리 소모가 막대하다. 또한, 이 때문에 메모리를 가급적 덜 사용하기 위해서는, 사용할 수 없는 문자들을 신중히 선택해야 한다.

※ 현재 프로젝트에 사용하는 GUI 시스템은 NGUI 플러그인 기반인데, NGUI의 정적 폰트에는 텍스처를 한 장만 사용할 수 있다.
이는 한, 중, 일 문자열에는 매우 제한이 심한 방식이며, 국가 별로 빌드를 할 때, 리소스 패키징을 별도로 관리해야 하는 등 유지 관리에도 어려움을 줄 수 있다. (텍스처 이름을 똑같이 가져가야 NGUI에서 사용한 문자열들의 폰트 링크가 깨지지 않기 때문이다.)

- **B-29-1-3.** 채팅 메시지에 색상을 입히는 기능을 허용할 것인지...
  : 허용해도 별로 상관없을 것 같긴 한데...

- **B-29-1-4.** 채팅을 보내고 싶은 대상에 따라, 다음과 같은 구분을 할 수 있어야 한다.
  : 사용자의 선택에 따라 채팅 창을 전환해야 한다.

1. 전체
: 말 그대로 전체 사용자들 대상은 아니고, 접속한 지역의 채널 전체에 대해서 채팅 메시지를 주고 받는다.

2. 길드
: 길드에 소속되어 있으면, 소속 길드원들끼리 대화할 수 있는 별도의 채팅 창으로 전환할 수 있다.

- **B-29-1-5.** 채팅 메시지의 입력기는 사용하는 기기에서 지원하는 입력기의 기능을 그대로 사용

한다.

- **B-29-1-6.** 게임 내 채팅 메시지는 별도의 전용 이모티콘을 지원하지 않는다.
: 사용하는 기기 플랫폼에서 자체적으로 지원하지 않는 한, 별도로 제작하거나 이모티콘 사용 기능을 지원하지 않는다.

◆ B-29-2. 친구 관리

- **B-29-2-1.** 정책상 모바일 메신저 플랫폼 등과 연계하더라도, 게임 내 친구 관리는 게임 서버에서 별도로 해야 한다.
: 게임 친구가 현실 친구는 아니기 때문이다.

※ 이러한 구조에서는, 모바일 메신저, 모바일 기기의 전화번호부, facebook 등의 친구 등은 그냥 친구에게 초대를 보내거나, 등록을 도와주는 기능을 뿐, 게임의 친구 관리 자체에는 관여하지 못한다.

- **B-29-2-2.** 친구로 등록된 사용자들의 캐릭터를 열람할 수 있어야 한다.
: 단, 친구 계정의 캐릭터들은 착용한 장비 아이템을 보는 것만 가능할 뿐, 어떠한 조작(아이템 교체 등)도 불가능하다.

- **B-29-2-3.** 친구로 등록된 사용자들의 프로필이나 캐릭터의 현재 아이템 보유 상황 등은 수동으로 갱신한다.

※ 웹 서버이기 때문에, 요청할 때만 접속하고 끊기를 반복하므로, 실시간 갱신을 하기는 어렵다.
더구나, 친구가 한두 명도 아닌 경우에는 (수십 개의 계정 × 계정 당 캐릭터 수)의 캐릭터 정보들을 갱신해야 하는데, 이 데이터의 트래픽도 (적어도 모바일 기기인 클라이언트의 입장에서는) 결코 무시할 수 없는 수준이다.
상대방 캐릭터를 그냥 구경하는 정도의 기능이기 때문에, 플레이어가 원하는 시점에 갱신을 수동으로 하도록 만드는 게 더 낫다. 물론, 아주 짧은 시간에 빈번하게 갱신하는 걸 막기 위해서 최소한의 대기 시간을 강제적으로 부여하는 게 좋다.

- **B-29-2-4.** 친구 등록 및 삭제는 하루에 할 수 있는 횟수를 제한한다.
: 무제한으로 친구 등록과 삭제를 반복할 수 있다면, 이를 이용한 네트워크 트래픽 공격도 가능할 수 있는 셈이다.
구체적인 제한 횟수는 관련 기획서의 요구사항을 따른다.

- **B-29-2-5.** 친구에 대한 플레이 초대 요청 횟수는 하루에 할 수 있는 횟수를 제한한다.
: 게임에 친구를 초대하면 경품을 주는 방식으로 마케팅을 하는 경우가 많은데, 이것도 제한이

아예 없으면 네트워크 트래픽 공격(+스팸 메시지 공격) 수단이 될 수 있다고 봐야 한다.
구체적인 제한 횟수는 관련 기획서의 요구사항을 따른다.

※ 친구 등록 / 삭제와 초대 요청에 대한 횟수 제한은 자체적인 결정 외에도, 프로젝트에 연계하고 있는 SNS 플랫폼의 서비스 규약에서 명시하고 있는 기준을 준수해야 할 수 있다.

◆ B-29-3. 우편 관리

- **B-29-3-1.** 수령 대상이 되는 플레이어, 혹은 캐릭터가 접속 여부와 관계 없이 비동기적으로 수신하는 알림 사항은, 그 알림 내용의 단위마다 한 개의 우편으로 취급하여 관리한다.

- **B-29-3-2.** 우편은 **[게임 관리자] -> [플레이어(혹은 플레이어의 캐릭터)]**, [플레이어(혹은 플레이어의 캐릭터)] -> [플레이어(혹은 플레이어의 캐릭터)]의 관계로 송 / 수신할 수 있다.
: 플레이어 송신하고 게임 관리자가 수신하도록 우편을 보낼 수는 없다.

- **B-29-3-3.** 자기 자신의 플레이어 계정, 혹은 그 계정 소유의 캐릭터에게 우편을 보낼 수 없다.
: 같은 계정의 A 캐릭터에서 B 캐릭터로 우편을 보낼 수 없다는 뜻이다.

- **B-29-3-4.** 우편함에 쌓아놓을 수 있는 우편의 개수에는 명시적인 제한이 없다.

- **B-29-3-5.** 플레이어(혹은 플레이어 캐릭터) 간 우편 전송은 제한 사항이 있어야 한다.
: 같은 플레이어에게는 하루 1회만 가능하다든지 이런 제한 사항이 있어야 한다.

※ 일반적으로 플레이어가 다른 플레이어들에게 보내는 우편은, 직접 메시지를 적는 방식이라기보다는, 사전에 기획적으로 설정되어 있는 어떤 점수나 자원 요소를 보내는 경우일 것이다.
대표적으로, '우정 점수'를 들 수 있다. 이런 경우에는 같은 플레이어에게 여러 번 보내는 것을 방지하기 위해, 같은 플레이어에게는 하루 1회 같은 제한을 두는 경우가 많다.

- **B-29-3-6.** 각 우편은 우편 메시지와 첨부물로 구성한다.

- **B-29-3-7.** 우편 메시지는 제목과 본문을 구분할지 여부는 기획 정책에 따른다.

※ 이는 사용자 편의성에 영향을 줄 수 있는 요소이기도 하다.
제목과 본문을 구분하는 방식이라면, 사용자가 거쳐야 하는 GUI 단계가 더 늘어나겠지만, 본문 내용의 길이에 대한 제한은 덜할 것이다.
반면, 제목이 곧 본문인 방식이라면, 가급적 많은 우편 목록을 보여줘야 하는 우편함 GUI의 특성상, 본문의 내용에 대해 길이 제한이 강할 것이다.

- **B-29-3-8.** 우편의 본문 메시지에는 일반 텍스트만 사용할 수 있다.

※ 그림을 첨부하거나, 웹 페이지 뷰를 이용해 본문을 구성하는 기능은 지원하지 않는다.
우편함 용도에 비해 개발 비용이 높을 것이라 판단한다.

- **B-29-3-9.** 우편의 **첨부물은 아이템으로 취급한다.**
: 아이템은 화폐나 티켓 등의 재화일 수 있고, 또는 생성한 장비 아이템일 수 있다.

※ 모든 아이템들은 고유한 아이템의 ID 와 그 수량을 가지고 있다.
첨부물에는 그러한 아이템의 ID 를 링크하는 역할을 한다. (실제로는 어떤 타입의 아이템인지 여부 등 클라이언트가 사용자에게 알려주기 위한 추가 정보가 필요하다.)

- **B-29-3-10.** 우편에 들어가는 첨부물은 **아이템 슬롯 1단위**로만 가능하다.
: 즉, 아이템 1종류를, 그 아이템이 아이템 슬롯에서 적재 가능한 단위의 수량으로 전달할 수 있다.

※ 장비 아이템은 슬롯 당 1 개씩 적재할 수 있으므로, 우편함에서도 1 개 단위로 전달해야 한다.
게임 플레이 화폐(골드)라든가 재료 아이템 등은 재화 슬롯에 지속적으로 축적하므로, 더 많은 수의 단위로 전달할 수 있다.

예를 들어, 10,000 골드는 우편 하나로 전달할 수 있다. 하지만, [무기 뽑기권] + [1,000] 골드는 2 개의 우편으로 전달해야 한다.

- **B-29-3-11.** 업적 보상, 미 수령 보상 아이템, 인벤토리 공간 부족으로 전달하지 못하는 아이템은 우편으로 수령할 수 있다.

- **B-29-3-12.** 우편에는 보관 유효기간이 존재할 수 있다.
: 유효기간 내에 수령하지 않은 우편은 자동으로 파기하여 삭제한다.

- **B-29-3-13.** 플레이어가 명시적으로 삭제하거나, 혹은 유효기간 초과 등의 이유로 삭제된 우편의 내용 및 첨부물은 복구할 수 없다.

- **B-29-3-14.** 모든 일반 우편들은 3일의 보관 기간을 가진다.

- **B-29-3-15.** 일반 우편들은 아이템의 등급에 따라 보관 기간이 달라지지 않는다.
: 대상 아이템이 아무리 고위 등급의 아이템이라도, 보관 기간은 하위 등급 아이템과 같다.

- **B-29-3-16.** 서버는 클라이언트의 우편을 전송할 때, **최근에 등록된 순서대로 100개 단위**로 우편 정보를 보낸다.

: 즉, 100개가 넘는 우편을 모두 확인하기 위해서는, 처음에 수신 받은 우편들을 확인하고 삭제해야 한다.

- **B-29-3-17.** 우편의 삭제는 첨부물을 수신하면 자동으로 이루어진다.

- **B-29-3-18.** 우편에 첨부된 아이템 중에서, 재화 종류는 한꺼번에 모든 우편으로부터 받기가 가능하다.
  그러나, 뽑기 티켓처럼 결과가 확정적이지 않은 아이템들은 모두 받기가 불가능하다.

- **B-29-3-19.** 모두 받기로 수령한 재화들은, 그 최종 합계만 사용자에게 알린다.
: 이 때는 각 우편 단위마다 알림 창을 띄우지 말고, 합계만 내서 한 번만 알려주면 된다.

- **B-29-3-20.** 클라이언트에서는 우편함에 있는 우편 내의 첨부물의 종류에 따라, GUI에서 그 목록을 분류할 수 있다.
: 이는 클라이언트의 GUI 단계에서 구분하는 것으로, 서버에서 전달하는 과정에서는 분류해서 전달해주지 않는다.

- **B-29-3-21.** 게임 서비스 운영진에서 플레이어에게 보내는 우편은 항상 우선적으로 전송하고, 표시할 때도 가장 상위에 오도록 표시한다.
: 공지사항이나, 점검에 의한 보상 등 운영진 메시지는 가장 우선적으로 알리기 위해서다.

◆ B-29-4. 길드 시스템(Guild System)

- **B-29-4-1.** 길드는 길드장(Master)과 길드원(Member)으로 구성한다.
: 구체적인 멤버십(Membership) 권한은 관련 기획서의 요구사항을 따른다.

- **B-29-4-2.** 하나의 길드가 가질 수 있는 총 회원 수는 제한되어 있다.
: 상세한 회원 수 제한은 관련 기획서의 요구사항을 따른다.

- **B-29-4-3.** **길드 가입과 탈퇴는 계정 단위로 이루어진다.**
: 플레이어는 하나의 계정 당 하나의 길드만 가입할 수 있다.

※ 캐릭터 별로 길드 가입을 허용하면, 플레이어의 의도에 따라 캐릭터들을 서로 다른 길드에 가입하게 하고 이를 악용하여 보상을 노리는 플레이를 하는 부작용이 생길 가능성이 있다고 판단했다.

- **B-29-4-4.** 길드에 가입한 플레이어는 **길드에서 사용할 대표 캐릭터를 지정해야 한다.**
: 정확하게는, 길드 전쟁에 사용한다. 플레이어가 지정한 캐릭터가 길드 전쟁에 나간다.

※ 처음으로 길드 활동에 참여할 캐릭터를 지정하게 하는 방법은, 자동으로 가장 높은 레벨의 캐릭터를 등록한다든가, 길드 가입할 때 플레이하고 있는 캐릭터를 자동으로 등록한다든가, 아니면 사용자에게 선택을 해 달라고 별도의 강제적인 과정을 거치게 하는 방법들이 있다.
어느 쪽이든, 사용자들에게 부담감을 줄이고 자연스럽게 받아들일만한 방식을 선택하면 된다.

- **B-29-4-5.** 각 플레이어들은 일정한 권한을 취득하면 새로운 길드의 창설이 가능하다.
: 구체적인 제한은 관련 기획서의 요구사항을 따른다. (대표적으로 일정한 레벨 달성이라든가, 스테이지 돌파 여부 등을 요구할 수 있다.)

- **B-29-4-6.** 길드에 가입하는 절차는 다음과 같다.

1. 길드장이 다른 플레이어에게 길드 가입 초청을 한다.
-> 초청을 받은 플레이어가 수락을 한다.
2. 플레이어가 대상 길드의 길드장에게 길드 가입 신청을 한다.
-> 신청을 받은 길드의 길드장이 수락을 한다.

- **B-29-4-7.** 길드 가입 절차에서 어느 한 쪽이라도 수락하지 않으면 길드 가입을 취소한다.

- **B-29-4-8.** 길드에 소속한 캐릭터들은 소속된 길드를 탈퇴할 수 있다.
: 탈퇴에는 별도의 허가 절차가 없다.
남은 길드장 및 길드원들에게 대상 플레이어의 탈퇴 사실만 통보한다.

- **B-29-4-9.** 길드장도 길드 탈퇴에 대해서는 길드원과 차이점이 없다.
: 길드장이 탈퇴하는 경우, 길드장을 자동으로 선출해야 한다.
그리고 길드장을 승계한 캐릭터에게는 길드장 승계 사실이 통보된다.

- **B-29-4-10.** 길드장은 자신의 길드장 권한을 다른 길드원에게 넘길 수 있다.
: 길드장 권한이 넘어간 전 길드장의 길드 내 권한은 길드원이 된다.

- **B-29-4-11.** 길드장은 자신의 길드에 소속되어 있는 길드원을 강제로 길드에서 추방할 수 있다.
: 길드원은 다른 길드원 혹은 길드장을 길드에서 추방하는 권한이 없다.

- **B-29-4-12.** 창설한 길드를 해제하면, 길드장을 포함한 모든 길드 구성원들은 길드 소속에서 벗어난다.

- **B-29-4-13.** 각 길드는 고유한 이름을 부여할 수 있다.
: 구체적인 길드 이름에 대한 제한은 관련 기획서의 요구사항을 따른다.

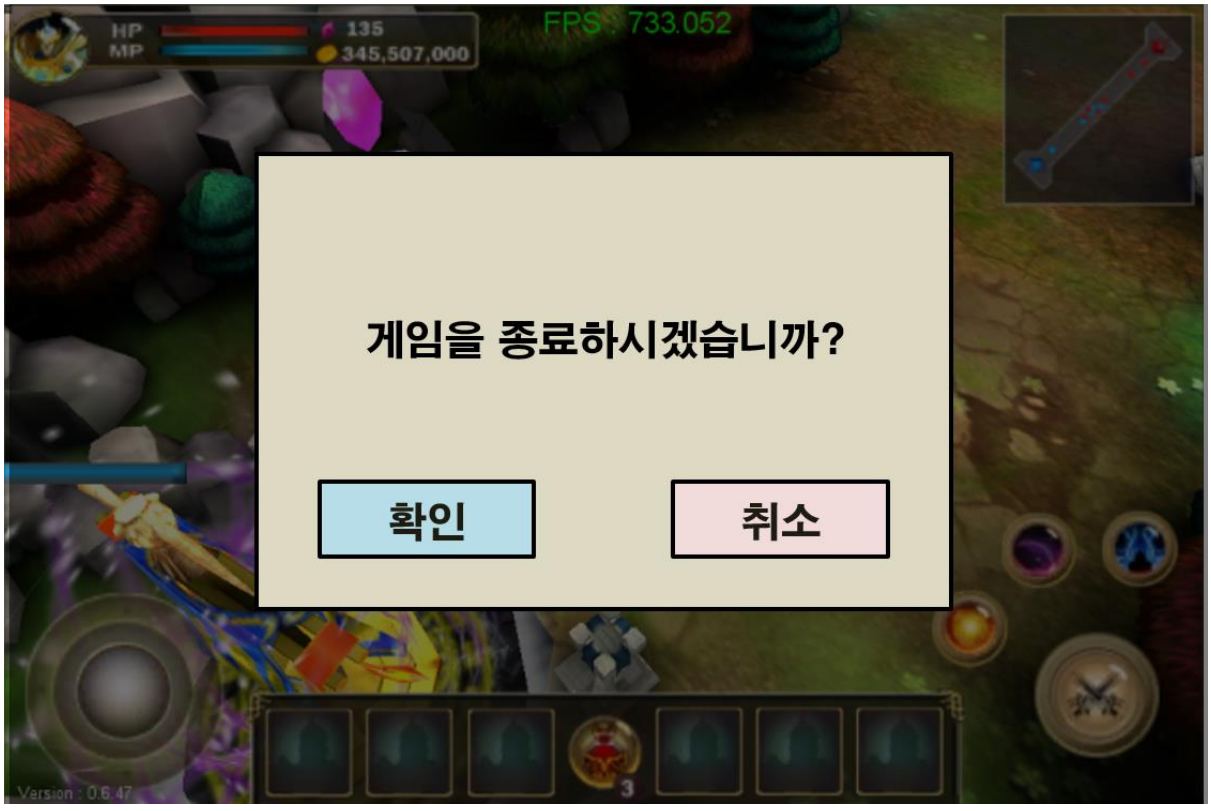
◆ B-29-5. 알림 메시지(Notice Message)

- **B-29-5-1.** 알림 메시지는 우편함 알림, 이벤트 보상 알림과 같은 고정된 게임 플레이 시스템 내 알림 말고, 그 외 상황에 대해 문자열 방식으로 알려주는 내용들을 말한다.

- **B-29-5-2.** 알림 메시지의 외적인 유형은 확인 방식과 통보 방식으로 나눈다.

- **B-29-5-3.** **확인 방식 알림 메시지(Comfirm Type Notice Message)**
: 알림 메시지를 Modal 팝업 윈도우에 출력해서 플레이어에게 보여준다. 플레이어는 그 메시지를 보고 확인 버튼을 터치해야 Modal 팝업 윈도우가 사라지고 이전 화면을 볼 수 있다.

※ 사용자가 꼭 확인을 해야만 하는 메시지들은 이 방식으로 알리는 게 적합하다.
예를 들면, 게임을 종료하는 확인 팝업 윈도우 같은 것들이 그렇다.



<확인 방식 알림 메시지>

- **B-29-5-4.** 확인 방식의 알림 메시지는 사용자가 수동으로 확인하여 Modal 팝업 윈도우를 확인하거나 끄지 않으면, 다음 메시지를 보여주지 않는다.

※ 여러 페이지로 구성되어 있는 (Web View 페이지로 구성되어 있는) 공지사항 팝업 윈도우들은

특히 이러한 방식의 좋은 예시이다.

- **B-29-5-5.** **통보 방식 알림 메시지(Inform Type Notice Message)**
: 사용자가 인식할 수 있도록, 기기 화면의 정해진 부분에 문자열 텍스트를 이용해서 알림 메시지를 일정 시간 동안 표시한다.
그렇지만, 별도로 사용자의 수동 확인 과정을 거치지 않는다.

- **B-29-5-6.** 통보 방식의 알림 메시지의 속성은 다음과 같이 구성한다.

1. **긴급 여부**
: 긴급 메시지는 중요하기 때문에, 일단 수신이 되면 긴급으로 설정하지 않은 메시지를 모두 가리고 표시한다.
또한, 긴급 메시지는 긴급하지 않은 메시지와 다르게, 다른 긴급 메시지도 가린다.

2. **표시 시간**
: 알림 메시지들은 동시에 여러 개가 적재되어 있는 상태에서 순차적으로 표시를 해야 하기 때문에, 표시하는 시간에 제한을 둬야 한다.

- **B-29-5-7.** 통보 방식 알림 메시지는 한 번에 2개 이상의 메시지를 화면에 표시할 수 있다.
: 구체적인 제한사항은 기기의 화면 넓이에 의한 UX를 고려하여 기획 측 요구사항을 따른다.

<통보 방식 알림 메시지>

B-30. 미션 / 퀘스트

◆ B-30-1. 정의 및 분류

- **B-30-1-1.** **미션(Mission)**
  : 하나의 스테이지를 플레이 할 때, 달성할 수 있는 여러 목표들의 집합을 미션으로 정의한다.

- **B-30-1-2.** **퀘스트(Quest)**
  : 미션을 구성하고 있는 각각의 개별 임무들을 퀘스트로 정의한다.

- **B-30-1-3.** 즉, 미션은 1개 이상의 퀘스트들의 목록으로 이루어진 집합이다.
  : 미션은 퀘스트의 상위 집합이다.

- **B-30-1-4.** 미션 당 최대 퀘스트 개수를 고정할 수 있다.
  : 이는 관련 기획의 요구사항을 따른다.

- **B-30-1-5.** 퀘스트는 미션을 완수하는데 필수적인 퀘스트와, 완수하지 않아도 미션을 완료하는 데는 지장이 없는 선택적인 퀘스트로 나눌 수 있어야 한다.

- **B-30-1-6.** **필수 퀘스트들을 모두 완료해야 그 미션을 완료한 것으로 판정**해야 한다.
  : 필수 퀘스트가 여러 개인 경우, 하나라도 완료하지 않으면 그 미션은 완료하지 못한 것으로 간주한다.

- **B-30-1-7.** 미션 완수에 대한 등급을 둘 것인지?

◆ B-30-2. 제한사항

- **B-30-2-1.** 하나의 스테이지는 하나의 미션을 가진다.
  : 미션 하나가 여러 개의 퀘스트를 가질 수 있으므로, 스테이지의 퀘스트는 여러 개일 수 있다.

- **B-30-2-2.** 모든 스테이지들은 최소한 한 개의 미션과 한 개의 퀘스트를 가져야 한다.
  : 즉, 어떤 스테이지에 들어가든지 간에 '적을 전멸하면 스테이지 승리' 같은 미션(과 퀘스트)이 존재한다는 뜻이다.

- **B-30-2-3.** 퀘스트는 목표를 한 종류만 지정할 수 있고, 분기 방식으로 완료하게 설정하지 못한다.
: A나 B 중 하나를 달성하면 완료라거나, A를 달성하면 B를 달성하지 못하는 방식의 디자인은 할 수 없다.

- **B-30-2-4.** **각 미션과 미션, 퀘스트와 퀘스트끼리는 의존성을 가질 수 없다.**
: 퀘스트의 완료 조건은 퀘스트마다 독립적이다. 퀘스트의 시작 조건이나 완료 조건이, 다른 퀘스트의 시작 또는 완료 여부에 의존하는 방식으로 디자인할 수 없다.

※ 다만, 미션과 퀘스트의 관계에서는 공식적인 의존성이 존재한다.

- 필수 퀘스트를 모두 완료해야 미션을 완료할 수 있는 점
- 선택적인 퀘스트의 달성 여부에 따라 미션 완료 보상이 달라지는 점

또한, 미션과 퀘스트의 의존성 관계 역시 위 두 가지를 제외하면 다른 방식으로 의존성을 디자인할 수 없다.
이렇게 의존성 제한을 심하게 두는 이유는, **게임의 시스템 구성 상 서버 측에서 클라이언트의 미션 / 퀘스트의 달성 정도를 실시간으로 추적하기가 어렵기 때문**이다. 이런 경우에는 미션 / 퀘스트의 달성 조건이 복잡할수록 이를 검증하기도 훨씬 더 까다로워진다.

- **B-30-2-5.** 하나의 스테이지가 여러 개의 구역으로 되어 있는 경우, 퀘스트 완료 조건이 있는 구역을 지나쳐버린 경우, 그 퀘스트를 완료하기 위해서 지나쳐 버린 스테이지 구역으로 다시 되돌아갈 수 없다.

※ 게임 클라이언트는 동시에 한 개의 스테이지 구역에 대한 정보만 유지하고 있다. 그 때문에 한 번 지나쳐 버린 스테이지 구역은 다시 되돌아갈 수 없게 되어 있다.
이에 대한 상세한 내용은 스테이지 관련 요구사항들을 참고할 것.

만약 지나쳐 버린 스테이지 구역을 다시 갈 수 있게 해야 한다면, 이전 스테이지 구역에서도 퀘스트 조건을 일부만 완료하고 지나쳤을 경우도 생길 수 있다. 이런 상황도 고려하면서 올바르게 처리를 하기 위해서는, 지나쳐 온 맵에 대해서도 퀘스트 진행 관련 객체들의 현재 상태까지 전부 유지를 해줘야 할 것이다.
위와 같은 방식으로 구현하자고 하면 아키텍처도 복잡해질 뿐 아니라, 사실상 하나의 스테이지를 여러 개의 조각으로 나눌 이유도 없어지는 셈이다.

◆ B-30-3. 미션 / 퀘스트 달성 검증

- **B-30-3-1.** 클라이언트가 소유하는 게임 데이터들을 관리하는 게임 서버에서는 제한적으로라도 클라이언트의 미션 / 퀘스트 달성 여부에 대해 검증을 수행해야 한다.

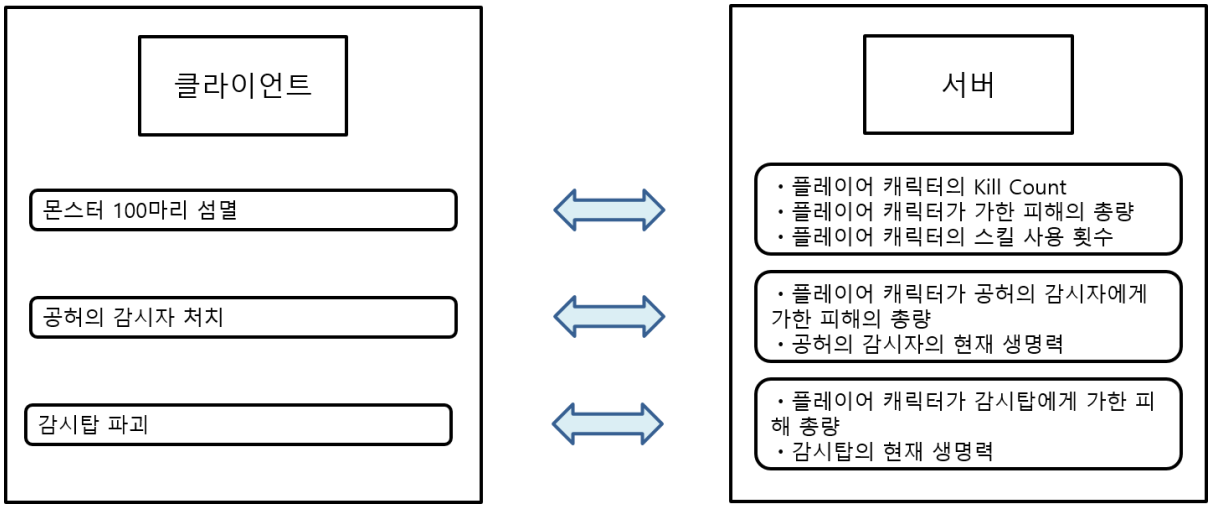
※ 게임 서버라고는 하지만, 이것도 연결을 지속적으로 유지하는 방식이 아니라, 필요할 때만 접속해서 연결을 주고 받는 웹 서버 방식이기 때문에, 미션 / 퀘스트에 대한 조건 달성에 필요한 정보들은 어쩔 수 없이 클라이언트가 쥐고 있다.
그렇지만 클라이언트가 미션 / 퀘스트 달성을 위해 점검하는 정보와, 서버에게 제출하는 결과를 100% 신뢰할 수는 없다. 클라이언트에서 판정을 하는 이상 100% 데이터 조작에 노출되어 있다고 가정해야 한다.

- **B-30-3-2.** 가장 좋은 방법은, **복잡한 검증이 필요 없을 정도로 미션과 퀘스트를 단순하게 만드는 것**이다.

※ 이건 농담이 아니다. 검증이 불필요할 정도로 미션과 퀘스트의 목표 달성이 단순하다면 가장 적은 비용으로 개발할 수 있다.
특정 종료 지점에 도달한다든가, 맵 위의 모든 적을 섬멸하는 미션은 조건이 아주 단순하다.
이런 미션만 존재한다면 재미가 없다고 느낄 수는 있을 것 같다. (하지만 게임 자체가 다 때려잡기 방식인데 뭐...)

- **B-30-3-3.** 클라이언트에서는 현재 플레이 하는 스테이지의 미션과 퀘스트의 달성 조건들을 기록하는 변수들 전체에 대해 Checksum 검증이 가능하게 해야 한다.
: 일단 이게 기본이다.

- **B-30-3-4.** 서버는 클라이언트의 각 미션과 퀘스트에 대해, 미션과 퀘스트의 목표를 적법한 방식으로 달성했는지 확인할 간접 조건들을 클라이언트로부터 받아오거나 서버 자체의 데이터에서 가져와야 한다.
: 클라이언트 뿐 아니라, 서버에서도 간접적인 데이터를 통해서 이중으로 확인하는 방식이다.



<서버는 클라이언트로부터 추가 정보를 받아서 검증한다.>

※ 클라이언트에서 무작정 서버에게 미션을 완료했다고 우기는 현상을 방지하기 위한 방책 중 하나다.
비록 클라이언트에서 각 조건 변수들에 대해 Checksum을 만들기는 하지만, 이런 직접 조건들 외에 간접 조건들도 같이 전송하게 해서 서버에서 같이 판단을 하게 만드는 편이 안전하다.

- **B-30-3-5.** 검증을 통과하지 못한 클라이언트는 미션 보상을 수령하지 못한다.

- **B-30-3-6.** 서버는 검증 과정에 통과하지 못하는 일이 발생한 클라이언트에 대한 정보를 기록한다.

※ 상습적으로 검증 과정 실패가 나는 계정은 게임 프로그램을 악용하는 사용자로 판정하게 할 수 있다.
몇 회 차일 때, 얼마나 자주 벌어지는지 여부를 '상습적'으로 판단할지는 차후 결정한다.

◆ B-30-4. 보상 처리

- **B-30-4-1.** 선택적인 퀘스트를 완료해서 얻는 (추가적인) 보상은, 게임 일반 화폐(골드) 및 이를 이용해서 구입할 수 있는 재화(일반 아이템 뽑기 티켓과 같이 상대적으로 흔한 보상)에 한한다.

※ 사용자의 게임 내 플레이를 실시간으로 감시할 수 없는 서버 구조이기 때문에 이렇게 방향을 잡는다.
미션과 퀘스트의 달성 여부 및 적법한 플레이로 달성했는지 여부를 서버 측에서 확인하는 데 한계가 있기 때문에, 애초에 기획할 때부터 게임 재화 밸런스나 과금 밸런스에 타격을 주지 않는 방식으로 기획을 해야 한다.

- **B-30-4-2.** 스테이지를 돌파한 시간에 따라 스테이지 돌파 등급을 매기고, 이에 대해 추가적인 보상을 수여한다.
  : 구체적인 스테이지 돌파 등급과 추가 보상 내역은 관련 기획서의 요구사항을 따른다.

- **B-30-4-3.** 스테이지 돌파 등급에 따른 추가 보상은, 게임 일반 화폐(골드) 및 이를 이용해서 구입할 수 있는 재화(일반 아이템 뽑기 티켓같이 상대적으로 흔한 보상)에 한한다.

※ 스테이지 돌파 등급 역시 서버에서 확고한 검증을 할 수 없으므로, 플레이어에게 가치가 대단한 보상을 주기는 어렵다.

B-31. 업적

◆ B-31-1. 업적 구성

- **B-31-1-1.** 업적은 계정 단위로 관리한다.
  : 즉, 하나의 캐릭터로 달성한 업적은 다른 캐릭터로 다시 달성할 수 없다.

- **B-31-1-2.** 업적은 게임의 플레이 모드에 관계없이 한 가지 종류만 작동한다.

- **B-31-1-3.** 업적은 해당 계정을 삭제하기 전까지 언제나 유지한다.

- **B-31-1-4.** 업적은 게임 플레이 모드와 관계없는 조건으로도 달성할 수 있도록 설계해야 한다.
  : 첫 과금 업적, 첫 구매 업적, 친구 초대 업적 같은 조건들도 받아들일 수 있어야 한다.

- **B-31-1-5.** 업적 조건은 **서버에서 열람이 가능한 요소들을 이용해서 구성**해야 한다.
  : 데이터베이스에 저장이 되는 항목들을 사용하는 업적 조건을 만드는 게 좋다.

※ 현재까지 사냥한 몬스터의 수, 획득한 골드의 양 등은 서버가 저장해두지도 않기 때문에, 열람하거나 계산해 낼 방법이 없고, 클라이언트가 보내준 정보에 지나치게 의존해야 하기 때문에 이런 정보들은 업적 대상에서 제외한다.

※ 업적에 대한 보상에는 과금 화폐, 아이템, 티켓과 같이, 민감한 현금 결제와 직접 관련이 있는 아이템들이 포함될 가능성이 높기 때문에, 클라이언트만의 정보에 의존해야 하는 조건을 더더욱 피하는 게 맞다.

◆ B-31-2. 업적 탐지의 구현 방식

- **B-31-2-1.** 업적 객체는 **(업적 종류) + ((업적 조건 + 업적 매개 변수-쌍Pair) × ○개)** 의 형태로 이루어진다.
  : 이러한 틀은 1개의 업적을 관리하는 객체를 표현한다.

- **B-31-2-2.** 업적 매개 변수-쌍은 **(현재 업적 매개 변수) + (목표로 하는 업적 매개 변수)**로 이루어져 있다.

- **B-31-2-3.** 업적 매개 변수는 반드시 저장할 수 있어야 하며, 중간 값이 없이 셈할 수 있는 자료형이어야 한다.
  : 즉, 문자열이거나 실수 자료형은 안 된다. 정수형 자료형만 가능하다.

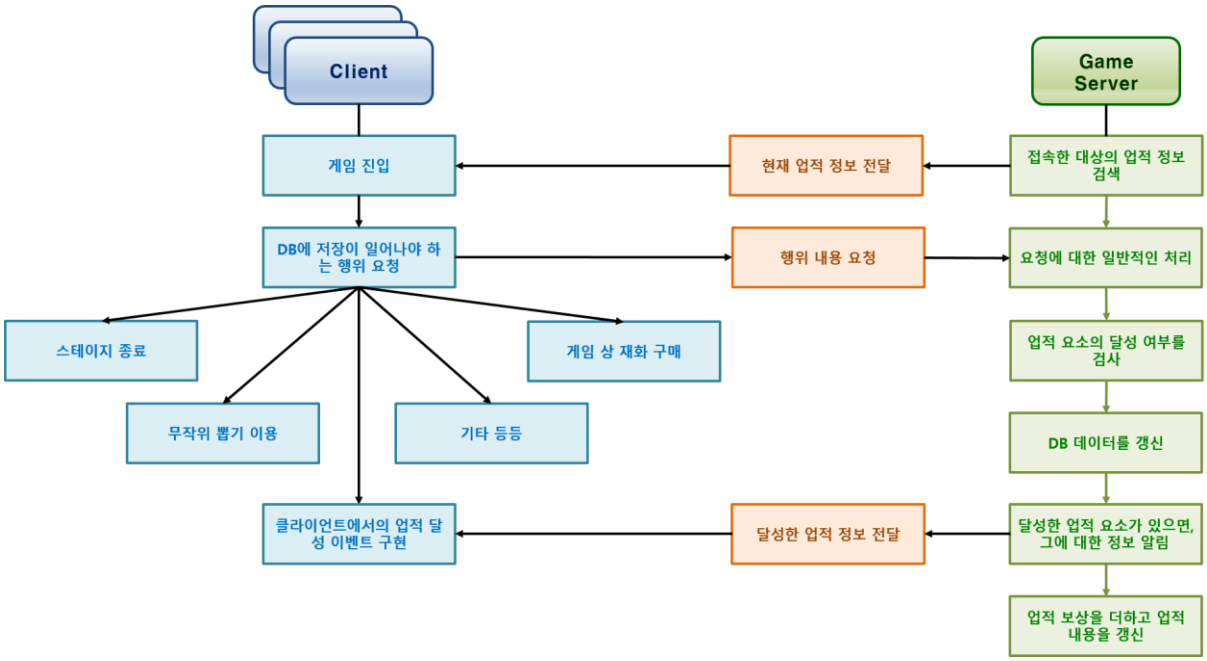
- **B-31-2-4.** 하나의 업적에 대한 달성률은 **% 단위**로 나타낸다.
  : 이 달성률은 **0 ~ 100%** 사이의 값을 가져야 하며, **소수점 두 자리**까지 허용한다.

- **B-31-2-5.** 하나의 업적에 대해 달성률이 0%이면 그 업적에 대해 하나의 조건도 달성하지 못한 상태이다.
  반면, 100%이면 해당 업적을 달성한 상태가 된다.

- **B-31-2-6.** **(현재 업적 달성률) = (현재 업적의 매개변수들의 총합) / (목표로 하는 업적 매개변수들의 총합) × 100**

: 이는 개별적인 업적 조건들의 상대적인 난이도를 고려하지는 못하지만, 공식과 계산이 쉽고, 어느 업적에나 적용 가능한 방식이다.

- **B-31-2-7.** 업적 이벤트의 대략적인 프로세스는 아래와 같다.



<업적 이벤트의 대략적인 프로세스>

- **B-31-2-8.** 업적 이벤트에 대한 검사를 하는 시점은, **대상 계정의 정보에 대해 데이터베이스에 대한 쓰기(혹은 갱신) 요청이 들어가야 하는 경우**이다.
  : 즉, 바꿔 말하자면, 해당 계정의 데이터베이스 정보를 갱신하지 않는 요청을 한 경우에는 업적 달성 여부를 검사하지 않아도 된다.

※ 사실 구현할 때 나올 수 있는 대부분의 서버에 대한 요청 상황들은 계정의 특정 정보를 업데이트하기 위한 것이다.
아이템의 사용 여부, 구매 여부, 재화를 얼만큼 구매하려고 했다든가, 재화를 소모하는 행위를 요청했다든가... 스테이지 완료를 알려왔다든가... 이들은 전부 해당 계정 정보의 DB 내용을 갱신해야 하는 경우이다.

이 게임은 웹 서버 기반으로 운영되기 때문에 업적이든 무엇이든 상시적이고 실시간으로 돌아가는 감시는 가능하지 않다고 본다. 또한, 업적 조건들은 전에 명세했듯, DB에서 추출할 수 있는 정보들만을 기반으로 한다.
따라서, DB의 내용이 변경되어야 하는 시점에만 업적 이벤트의 달성 여부를 검사하는 게 가장 효과적이라고 할 수 있다. 그리고 그 시점은 실질적으로는, 클라이언트에서 DB에 뭔가 변경이 일어날만한 사항을 요청하는 시점이다.

P.S : 구현 여부에 따라서는 DB에 정보를 갱신하는지 아닌지에 대한 요청을 구분하기가 애매하거나 어렵거나, 실수하기 쉬울 수도 있다. 아니면 업적 조건이 몹시 독특해서 DB 갱신 여부와 상관이 없을 수도 있다.
그럴 때는 차라리 고민하지 말고 웹 서버에 요청이 일어날 때마다 업적 조건들을 검사하게 만드는 것도 한 방법이다.

◆ B-31-3. 보상 처리

- **B-31-3-1.** 업적에 대한 보상은 해당 업적을 소개하는 GUI를 통해 직접 수령한다.
: 우편 수령도 생각을 해봤는데, 그냥 직접 그 자리에서 수령하는 게 사용자들에게 덜 귀찮을 것으로 판단했다. (2단계가 1단계로 줄잖아.)

- **B-31-3-2.** 업적 보상에 대한 특별한 제약은 없다.
: 업적 보상의 내용은 정책적인 부분에 달려 있다.

※ 업적 달성에 대한 사항들은 데이터베이스에 저장되며, 달성 여부를 서버가 판정하기 때문에, 상대적으로 비정상적인 플레이에 의한 보상을 하게 될 위험이 적다.
또한, 업적 자체가 1회성이기 때문에 보상에 지나치게 제한을 둘 수 없는 이유 또한 존재한다.

B-32. 랭킹 / 리그

◆ B-32-1. 랭킹 / 리그 시스템 구성

- **B-32-1-1.** **최초 출시에는 반영하지 않으며, 이후 업데이트에 반영할 컨텐츠이다.**

- **B-32-1-2.** 결투장과 길드 전쟁에 대해 별도의 랭킹을 구성한다.

- **B-32-1-3.** 사용자의 랭킹은 순위 전체, 혹은 특정한 순위 구간 별로 리그를 구성할 수 있다.

- **B-32-1-4.** 사용자의 랭킹은 그 랭킹이 소속되어 있는 리그 내에서 상대적인 순위로 표시할 수 있다.
  : 즉, 리그를 고려하지 않은 (절대 기준의)랭킹이 있지만, 플레이어에게 표시할 때는 소속되어 있는 리그 내에서의 상대적인 순위로 표시할 수도 있다

※ 어떤 플레이어의 랭킹 순위가 160위라고 가정하고, 0 ~ 50위는 플래티넘, 51 ~ 150위는 골드, 151 ~ 300위는 실버 등급으로 리그가 나뉜다고 치자.
그러면 160위인 플레이어의 순위는 실버 등급 리그 내에서 표시할 때, 실제 순위인 160위 그대로 표시할 수도 있겠지만, 리그 내의 상대적인 순위인 10위로 표시할 수도 있다는 뜻이다.

- **B-32-1-5.** 같은 게임 플레이 모드에서의 리그는 여러 개가 존재할 수 있다.

- **B-32-1-6.** 단, 그러한 리그의 등급은 수직적으로 구성해야 한다.
  : 동일한 게임 플레이 모드에서, 같은 등급의 리그가 2개 이상 존재할 수 없다.

- **B-32-1-7.** 플레이어의 게임 플레이 모드 별 리그는 하위 등급에서 상위 등급으로 승급하거나, 상위 등급에서 하위 등급으로 하강할 수 있다.

- **B-32-1-8.** 게임 플레이 모드에 따른 리그의 등급제는 동일하지 않을 수 있다.
  : 결투장 모드는 여러 개의 리그 등급으로 나눠도, 길드 전쟁은 단일 리그로 진행하게 할 수도 있는 법이다.

※ 만약 이용자 층이 넓지 않다면, 극도로 세분화 되어 있는 리그 구성은 오히려 부자연스럽고 개발력만 과도하게 소모하는 셈이 될 수 있다.

◆ B-32-2. 랭킹 관련 정책

- **B-32-2-1.** ELO Rating을 기반으로 구성한다.

- **B-32-2-2.** 오랫동안 랭킹에 참여하면, 서서히 점수가 오르는 방식으로 구성한다.
  : 이 '서서히'의 정도는 관련 기획서의 요구사항을 따른다.

※ 즉, 이는 같은 Rank 등급일 경우, 승리했을 때의 점수가 패배했을 때의 점수보다 높아야 가능하다. 그렇지만, 순위는 변동이 없이 점수만 계속해서 높아지는 사항이 적절한지는 논의를 해야 할 것이다.
분명 달성하는 성취가 있다는 점은 사실이지만, 이 성취가 실질적으로는 순위를 향상하기 위한 수단에 불과한 점을 생각한다면, 과연 랭킹 점수만으로 성취감을 느낄 수 있을지는 확신할 수 없기 때문이다.

※ GUI 입장에서는, 이 숫자의 누적이 지나치게 커지지 않도록 주의해야 한다. 랭킹 정보는 대개 하나의 열(Row)에 대해 많은 정보를 표시해줘야 하는데, 숫자의 길이가 지속적으로 길어질 수 있는 점은, 모바일 기기가 주된 출시 대상이라는 점을 고려하면 우려해야 할 사항일 수 있다.

- **B-32-2-3.** 승급 및 강등에 대한 구체적인 내용은 관련 기획서의 요구사항을 따른다.

- **B-32-2-4.** 클라이언트가 보는 랭킹을 최신으로 갱신하는 시점은 다음과 같다.
  : 제시하는 시점에서만 서버에게 최신 랭킹 정보를 요청해야 한다.

1. 랭킹 페이지에 처음 진입했을 때
2. 랭킹 페이지를 사용자가 새로고침(Reset) 버튼을 터치해서 갱신하도록 명령했을 때

- **B-32-2-5.** 랭킹 정보는 클라이언트가 요청한 시점에서 최신 랭킹일 뿐, 항상 최신 랭킹 정보임을 보장하지는 않는다.

※ 랭킹 정보는 한 순간도 다르지 않고 최신으로 동기화해야 할 만큼 필수적인 정보는 아니다. 그저 참고사항에 불과한 정보를 매 순간 갱신하는 건 낭비에 불과하다.

B-33. 튜토리얼

◆ B-33-1. 제한사항

- **B-33-1-1.** 튜토리얼은 사용자의 의사에 의해 건너뛸 수 있어야 한다.
  : 만약 튜토리얼에 의해 보상을 지급한다면, 튜토리얼을 건너뛰면 그 보상을 획득할 기회는 사라지는 셈이다.

- **B-33-1-2.** 동영상을 재생하는 GUI를 사용할 수 없다.
  : 만들 수 있을지 없을지도 모르겠고, 복잡하기도 하고, 화면 일부의 GUI 패널에 아주 작게 재생하고 있는 동영상을 누가 좋아할까?

※ (특히 PC나 콘솔 용의)일부 게임들은 게임 내 GUI 일부에 동영상을 재생할 수 있는 기능이 있어서, 이를 통해 처음 사용자들에게 게임의 각 기능을 구체적으로 어떻게 사용하는지, 사용한 결과가 어떻게 보이는지를 미리 만든 동영상을 통해 보여주곤 한다. (대표적으로 ©NCSoft의 'AION'의 튜토리얼도 그러하다.)
이런 튜토리얼들의 GUI처럼, In Game GUI에 동영상을 재생할 수 있는 기능을 구현하거나 지원하지 않는다는 뜻이다.

◆ B-33-2. 튜토리얼 구성 도구

- **B-33-2-1.** 이런 거 필요 없다는 의견도 있다. 뭐가 효율적인지는 생각을 해봐야겠다.

- **B-33-2-2.** 유니티 프리팹 단위에서 조정할 수 있는 인터페이스를 제공하는 정도라면 충분하지 않을까 싶다. 굳이 텍스트 기반 스크립트 테이블을 써야 할만한 이유가 없다면...

B-34. 싱글 플레이 컨텐츠

◆ B-34-1. 스테이지 돌파

- **B-34-1-1.** 여러 개의 스테이지를 순차적으로 통과하는 방식으로 구성한다.

- **B-34-1-2.** 각 스테이지는 순차적으로 진입할 수 있으며, 이전 스테이지를 완료하지 않으면, 다음 스테이지로 갈 수 없다.

- **B-34-1-3.** 이미 완료한 스테이지는, 입장 화폐만 있다면 아무런 제한 없이 입장할 수 있다.

- **B-34-1-4.** 스테이지의 진행 방향은 갈라지지 않는다.
  : 즉, 30-1, 30-2 같은 스테이지를 두지 않는다. 모든 스테이지 진행은 일방 진행이다.

- **B-34-1-5.** 같은 스테이지에 대해 난이도를 여러 개 두지 않는다.
  : 각 스테이지는 한 종류의 난이도만 존재한다.

※ 스테이지에 난이도를 주게 되면, 향후 스테이지 컨텐츠를 확장해야 하는 경우가 있을 때 문제가 될 수 있다.
스테이지를 처음에 제작한 그대로 고정한다면, 난이도를 여러 개 배치하는 방향으로 컨텐츠를 확장할 수 있지만, 별개의 스테이지를 제작하는 경우에는 난이도의 동선이 꼬여버릴 수 있다.

예를 들어, 50개의 스테이지가 존재했는데, 향후 업데이트로 인해 20개의 스테이지를 더 추가한다고 가정하자. 그런데 기존 50개 스테이지가 3단계 난이도로 구성되어 있었고, 난이도가 서서히 증가했다고 치자. 그렇다면 51 ~ 70 스테이지의 1번째 난이도가 1 ~ 10 스테이지의 2번째 난이도와 별 차이가 없거나 심지어 역전하는 사태까지도 나올 수가 있다. 이를 자연스럽게 하기 위해서는 스테이지를 새로 추가할 때마다 기존 스테이지의 난이도를 재조정해야 하는데, 이는 스테이지를 추가하면 할수록 막대한 작업량이 될 수 밖에 없다. 심지어, 컨텐츠의 특성과 복잡도를 고려하면 불가능할 수도 있다.

그러니, 향후 컨텐츠 업데이트를 통해 새로운 스테이지를 확장할 계획이라면, 기존 스테이지 난이도를 건드리지 않아도 되도록, 스테이지 난이도를 한 개만 두는 게 서비스에 더 효과적이라고 본다.

- **B-34-1-6.** 스테이지의 난이도는, 스테이지에 입장한 캐릭터의 레벨과 강력함에 비례하지 않고, 사전에 고정된 난이도로만 제공한다.

: 구현상의 복잡도를 줄이고, 플레이어에게 다음 스테이지로 진행할 동기를 부여하기 위해 이렇게 결정한다.

- **B-34-1-7.** 스테이지의 입장 화폐는 공통적으로 사용한다. 단, 입장에 사용하는 화폐의 개수는 스테이지 성격에 따라 다를 수 있다.
: 일반적인 스테이지는 1개의 입장 화폐를 요구하지만, 보스 스테이지는 특별히 2장 이상의 화폐를 요구하는 경우가 있을 수 있다.

- **B-34-1-8.** 스테이지를 완료하면, 플레이어는 해당 스테이지에 설정되어 있는 경험치와 보상을 획득하는 과정을 거친다.

◆ B-34-2. 혼돈 던전

- **B-34-2-1.** 무작위로 등장하는 스테이지에서, 무작위로 등장하는 몬스터들을 물리치고, 스테이지 보상을 획득하는 게임 모드이다.

※ ©Blizzard의 Diablo Ⅲ의 게임 모드 중 하나인 균열 / 대균열 모드와 비슷하다. 실제로 모티프도 여기에서 따 왔다.
최근 모바일 RPG에서 보통 등장하는 '무한의 탑' 모드의 전형적인 방식과는 다른 느낌을 주는 반복적인 성격의 컨텐츠를 만들고자 했다.

- **B-34-2-2.** 혼돈 던전에 등장하는 지형 맵들은 모두 스토리 모드에서 사용했던 스테이지의 지형 맵들이다.
: 그 중에서 무작위로 골라서 등장한다.

- **B-34-2-3.** 하지만 스테이지 내부의 몬스터 구성은 혼돈 던전에 들어갈 때마다 달라진다.
: 몬스터의 종류와 배치 숫자, 배치 위치, 등장할 최종 보스 몬스터의 종류는 혼돈 던전에 들어갈 때마다 바뀐다.

※ 여기에는 숨겨져 있는 기술적인 이슈가 있다.
임의의 지형 맵 정보를 가지고서, 무작위로 몬스터 그룹들을 생성해서 배치하는 기능이다.

- **B-34-2-4.** 무작위로 구성할 지형 맵과 몬스터들은 본래 스토리 모드에서 쓰이던 리소스들을 그대로 활용한다. 또한, 구태여 지형 맵과 몬스터들의 컨셉을 맞추지도 않는다.
: 빙하 맵에 등장하는 화염 몬스터들이나, 사막에 등장하는 늪지 몬스터들과 같은 구성도 얼마든지 허용한다.

- **B-34-2-5.** 혼돈 던전의 구성이 반드시 **'모든 내용에 대해 기술적으로' 무작위함을 의미하지는**

않는다.

: 미리 정의한 요소들이 존재하더라도, 플레이어가 무작위적으로 '느낄 수 있는' 요소들이 있으면 이를 통해서 구현해도 된다.

※ 예를 들면, 무작위 지형을 생성하는 알고리즘 및 코드를 구현하는 데 비용이 크기 때문에, 기존 스테이지의 지형 맵들을 그대로 재활용하는 부분을 들 수 있다.
또한, 몬스터의 배치 위치 역시 임의 생성이 곤란하다면, 각 지형 맵마다 사전에 정의한 혼돈 던전 용 몬스터 배치 위치를 그대로 쓰거나, 혹은 그 일부만 골라서 쓰는 방식으로 구현할 수도 있다. (생성 위치 50개 중 20개만 무작위로 골라서 쓴다거나...)
몬스터의 그룹이나 스테이지 보상 역시 무작위하게 보이지만, 사실 사전에 그룹 지어놓은 목록 중에서 고르는 방식이어도 된다.
결국, 이 모든 내용들은 **사용자의 눈에 '무작위로 보이게끔' 하면 될 뿐**이지, 내부 구현이 기술적으로도 완전히 무작위적이어야 할 이유는 없다.

- **B-34-2-6.** 혼돈 던전은 최고 레벨을 달성한 캐릭터만 이용할 수 있는 컨텐츠이다.
: 이는 혼돈 던전의 몬스터의 능력과 배치 방식을 특정한 캐릭터 레벨 대역에 맞춰야 개발 난이도를 낮추는 데 유리하기 때문이다.

※ 혼돈 던전 컨텐츠를 이용하는 플레이어 캐릭터의 레벨이 고정 되어 있지 않다면, 던전의 전보를 생성할 때 플레이어의 현재 성장 단계까지 고려해주는 방식이어야 한다.
그것보다는 아무래도 특정한 성장 단계만 고려하는 편이 기능을 구현하는 데 더 명확하다. 그리고 최종 컨텐츠 성격으로써 고정할 수 있는 성장 단계는 결국 '최고 레벨'이기 때문에 이렇게 결정하였다.

※ 사실, 여기에는 좀 더 고민해봐야 할 마케팅 이슈가 있다.
혼돈 던전이 최상위 컨텐츠 성격으로 구현하는 모드라고는 하지만, 무작정 최고 레벨까지 도달해야만 사용할 수 있다면, 사용자들의 이용 빈도가 더 낮아서 매출에도 악영향이 있지 않을까 하는 우려가 있다.
아무래도, 혼돈 던전을 이용하지 못하는 사용자 층들은 죽어라 싱글 플레이로 스토리 모드나 파야 하는데, 이러면 플레이 할 컨텐츠 종류가 제한적이어서 재미가 덜할 수도 있지 않겠느냐는 예상도 있는 편이다.

- **B-34-2-7.** 몬스터들의 외형과 액션 데이터들은 스토리 모드의 난이도에 관계없이 가져 오되, 능력치는 캐릭터 최상위 레벨에 맞춰서 어렵게 설정한다.

※ 애초에 능력치를 설정하는 부분에서 기준 값의 XX%를 적용하는 식으로 설계하면 큰 문제 없이 구현이 가능하다.

- **B-34-2-8.** 모든 혼돈 던전 스테이지의 최종 지점에는 그 스테이지의 최종 보스가 되는 몬스터

가 등장해야 한다.
: 이 몬스터를 무찔러야 혼돈 던전 스테이지를 완료할 수 있다.

◆ B-34-3. 친구 소환

- **B-34-3-1.** 친구 캐릭터의 정보를 가지고 있는 NPC를 아군으로 소환해서 같이 스테이지를 진행하는 개념이다.

- **B-34-3-2.** 플레이어들이 실제 시간으로 같은 시간에 같은 방에 접속해서 파티를 맺고 플레이하는 방식은 구현하지 않는다.

※ 복수의 플레이어가 연결을 유지하면서 같은 게임을 플레이 하는 방식과 관련된 기능들은 설계 단계에서부터 배제하고 있다. (상세한 내용은 동기화 관련 요구사항을 참고할 것.)

- **B-34-3-3.** 당연히, 친구 계정 플레이어는 접속을 하지 않더라도 얼마든지 내 쪽에서 (비용만 치르고 기획상 허용 조건만 맞으면) 이용하는 데 문제가 없다.

- **B-34-3-4.** 파티 플레이는 스토리 모드에서만 사용할 수 있다.
: 다른 게임 모드에서는 사용할 수 없다.

- **B-34-3-5.** 소환한 친구 캐릭터의 동작
: 소환한 친구 캐릭터는 내 캐릭터를 따라다녀야 하고, 내 캐릭터의 친구 / 적대 상황을 똑같이 공유한다.
상세한 동작 내역은 관련 기획의 요구사항을 따른다.

- **B-34-3-6.** 친구 캐릭터를 소환하는데 필요한 구체적인 제약 사항들은 관련 기획서의 요구사항을 따른다.

B-35. 멀티 플레이 컨텐츠

◆ B-35-1. 결투장(Duel, PvP)

- **B-35-1-1.** **최초 출시에는 반영하지 않으며, 이후 업데이트에 반영할 컨텐츠이다.**

- **B-35-1-2.** **비동기 방식으로 진행**한다.
: 풀어서 설명하자면, 싸울 상대방 플레이어 캐릭터의 데이터를 가져와서, 그 데이터를 적용한 NPC와 전투하는 셈이다.

- **B-35-1-3.** 결투 대상인 상대 플레이어의 캐릭터가 그 시점에 플레이 중이라고 하더라도, 결투 진행에 있어 아무런 제한 사항이 없으며, 상대방 캐릭터의 소유자 플레이어의 접속 여부와 상관없이 비동기 방식으로 전투를 진행한다.
: 즉, 어떤 경우라도 상대의 플레이어는 자신의 캐릭터가 직접 조종하는 방식은 아니다.

※ 즉, 이 결투라는 시스템은, 상대방 플레이어가 현재 무엇을 하고 있는지, 접속은 한 상태인지와 전혀 상관없이 진행한다.
내 캐릭터가 한창 싱글 플레이로 몬스터들을 때려잡고 있는 도중에, 다른 누군가는 내 캐릭터와 (나 자신은 직접 조종하며 싸우지는 못하지만) 결투를 벌이고 있을 수 있다는 말이다.

- **B-35-1-4.** 결투장에 들어가서 결투를 진행하는 캐릭터들은 플레이어의 조종이 가능하다.
: 실질적으로는 결투장을 만든 플레이어만 결투장 스테이지에 들어가는 셈이므로, 결투장을 신청한 플레이어만 자신의 캐릭터를 조종하는 셈이다. 상대방 플레이어 캐릭터는 능력치와 장비 아이템만 같은 NPC가 인공지능으로 싸운다.

- **B-35-1-5.** 결투장의 매치 단위는 1 대 1 방식만 지원한다.

※ 나중에 2 대 2, 3 대 3 등의 모드를 원할 수도 있겠지만, 일단 기능 출시 시점에서는 1 대 1을 먼저 지원하는 것으로 결정한다.

※ 사용자 경험(UX)에 관련된 문제점도 있다.
만약 2 대 2 이상의 기능이 필요하다면, 게임 친구와 같이 협동해서 결투장에서 팀 결투를 진행하는 기능이 자연스럽게 필요할 것으로 예상한다.
그런데, 친구와 같이 협동하는 방식이라면, 실제로 오프라인에서 만나서 하는 경우도 종종 있을 수 있다. 이런 경우에는 실시간으로 동기화되지 않는 결투 방식이 매우 어색하게 느껴진다.

왜냐하면 개념적으로나 팀을 이루지, 실제로 팀을 이뤄서 '동일한 상황의 결투'를 하는 느낌은 전혀 줄 수 없기 때문이다. 그냥 **말이 팀 대전이지 사실상 개별적으로 따로 진행하는 것과 다름이 없다.**
그래서 만약 2 대 2 이상의 팀 대전 기능이 필요하다면, 결투 시스템 자체를 실시간 동기화 방식으로 만드는 게 좋다고 판단했다. 만일 실시간 동기화를 하지 않는다면, 1 대 1 결투 기능만 지원하는 게 가장 덜 어색한 방법이라고 본다.

- **B-35-1-6.** 상대 팀의 플레이어들을 모두 사망하게 만들고 자신의 팀에 살아 있는 플레이어 캐릭터가 있으면, 그 결투장은 승리 처리한다.

- **B-35-1-7.** 자신의 팀의 팀원들이 모두 사망하고, 상대 팀의 플레이어들 중에서 살아남은 캐릭터들이 있으면, 그 결투장은 패배 처리한다.

- **B-35-1-8.** 두 팀의 팀원이 전부 사망(아마도 동시에?)하는 경우가 있을지도 모르겠다.
: 이 경우에는 먼저 사망 판정이 일어난 쪽이 패배이며, 그 반대편 팀이 승리가 된다.

- **B-35-1-9.** 몇 가지 결정하지 못한 쟁점들

1. 가짜 캐릭터를 데이터만 가져와서 PvP라고 하는 셈인데, 이게 재미있는지에 대한 문제
: 승률이 대체적으로 높게 나올 것이므로(유사한 게임의 경우 거의 승패 비율이 10 : 1 이상 나오는 경우가 많았다.), 승률 스트레스는 상대적으로 덜하겠지만, PvP 그 자체가 지겨워질 가능성이 있다 (보통 비동기 방식의 대전은 대전 횟수가 많아야 점수를 올리기 유리한 구조로 갈 수밖에 없을 경우가 많다.).

2. 전투가 매우 빠르게 끝나는 경우, 극단적으로는 10초 싸우려고 10초 동안 로딩 화면을 봐야 하는 경우도 생길 것이다.
: 당연히 좋아할 수가 없을 것이다.

3. PvP 컨텐츠 자체는 없앨 수가 없는 점이 문제다.

◆ B-35-2. 월드 보스 몬스터 집단 공략(World Boss Monster RAID, PvE)

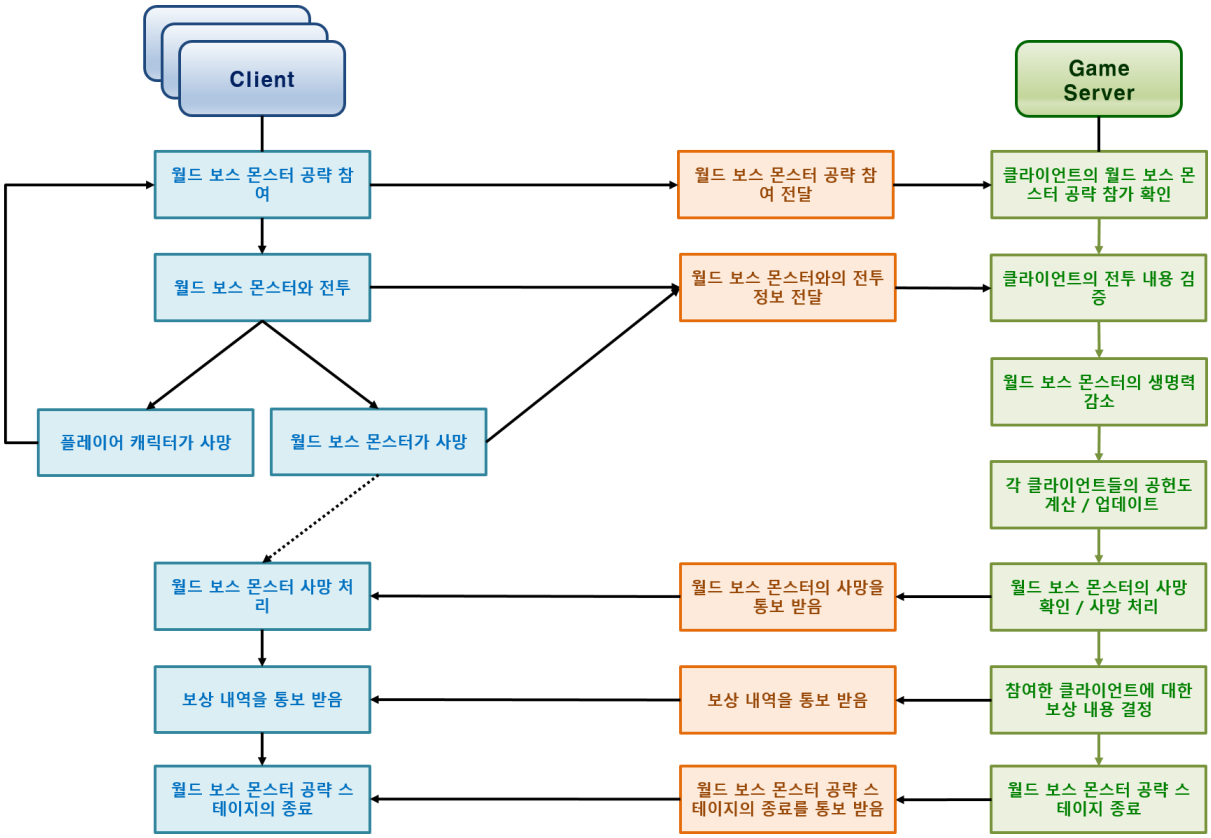
- **B-35-2-1.** **최초 출시에는 반영하지 않으며, 이후 업데이트에 반영할 컨텐츠이다.**

- **B-35-2-2.** 전체 사용자들이 하나의 보스 몬스터를 사냥하는 개념이다.

- **B-35-2-3.** 하지만, 기술적으로는 각 클라이언트들이 개별적으로 보스 몬스터를 공격하는 방식이다.

: 다중 사용자들이 같은 방에 접속해서 동기화를 유지하면서 사냥하는 개념이 아니다.

※ 그렇기 때문에, 월드 보스 몬스터에 대한 전투 역시 클라이언트가 판정하고, 그 결과를 서버에 통보하는 방식이다.
서버에서는 클라이언트의 전투 정보를 받아들이지만, 유효성 여부를 점검하고 처리한다.



<월드 보스 몬스터 공략의 대략적인 프로세스 모델>

- **B-35-2-4.** 플레이어는 월드 보스에 대한 공격을 하기 위해서는, 이를 위한 스테이지에 참가해야 한다.
또한, 이에 대해 요금이 필요할 수 있다.
월드 보스 대전에 참가하기 위한 요금에 대한 세부 내용은 관련 기획서의 요구사항을 따른다.

- **B-35-2-5.** 플레이어는 같은 월드 보스를 공격하는 스테이지에 여러 차례 참가할 수 있다.
: 요금만 지불한다면, 몇 번을 참가하든 관계 없다.

- **B-35-2-6.** 월드 보스 몬스터는 공략에 참여한 전체 플레이어로부터 공격을 받는다.
월드 보스 몬스터의 체력이 0 이하가 되면 사망으로 처리하고, 그 시점에서 월드 보스 몬스터의 공략은 종료한다. ~~(그리고 신나는 보상 타임~)~~

- **B-35-2-7.** 월드 보스 몬스터를 죽였을 때, 그 공헌도가 큰 순서대로 보상을 받는다.

: 세부적인 공헌도 기준 및 보상 정책에 대해서는 관련 기획서의 요구사항을 따른다..

- **B-35-2-8.** 월드 보스 몬스터 공략전에서는 요금을 지불하고 부활하는 기능을 사용할 수 없다.
: 공략하다가 사망했으면 무조건 해당 스테이지를 빠져 나갔다가 다시 참여해야 한다.

- **B-35-2-9.** 각 클라이언트는 월드 보스 몬스터에게 가한 피해량을 계산하여 다음 조건에 도달하면 그 정보를 서버에게 전달한다.

1. 캐릭터가 스스로 방을 퇴장해서 보스 대전 스테이지를 종료하였다.
: 캐릭터가 사망해서일 수도 있고, 스스로 포기하는 경우도 마찬가지다.

2. 월드 보스 몬스터가 내 공격에 사망하였다.
: 클라이언트에서 전투를 진행했는데, 월드 보스 몬스터가 사망했다면, 진짜인지 확인을 하기 위해 현재까지의 전투 진행 정보(내 캐릭터가 가한 피해량 + 검증에 필요한 재료가 될 정보들)를 보낸다.

- **B-35-2-10.** 서버에서는 클라이언트가 전달한 피해량을 받아서 그 클라이언트가 플레이 중인 캐릭터의 착용 장비와 능력치를 토대로 하여 가능한 수치인지 검증한다.
: 검증에 통과했으면, 서버는 그 클라이언트의 플레이어가 월드 보스 몬스터 공략에 기여한 점수를 산정해서 업데이트한다. 이 정보는 서버만 알고 있고 클라이언트에게는 알리지 않는다.
월드 보스 몬스터의 공략에 대한 기여도를 측정하는 방식은 관련 기획서의 요구사항을 따른다.

※ 접속 요청, 피해 판정과 피해량 수치를 클라이언트에게 의존하기 때문에, 클라이언트 측이 말도 안 되는 피해량을 보내오거나, 1초에 수십 번씩 전투를 시도하는 경우를 차단해야 한다.

- **B-35-2-11.** 검증에 통과하지 못한 클라이언트의 공헌도와 피해 수치는 반영하지 않는다.
또한, 검증을 통과하지 못한 클라이언트는 해당 월드 보스를 더 이상 공략할 수 없다.
: 검증에 통과하지 못한 플레이어에 대한 정보는 로그를 남겨서 향후에 징계 근거로 삼는다.

- **B-35-2-12.** 한 번 월드 보스 몬스터의 공략에 참가한 플레이어는 그 스테이지를 퇴장하였으면, 같은 월드 보스 몬스터 공략에 다시 참여하기 위해서는 최소한 3초간 기다려야 한다.
: 과다한 네트워크 트래픽 유발을 방지하고, 불법적인 치트 도구를 배제하기 위함이다.

- **B-35-2-13.** 클라이언트에서 전투를 진행하다가 월드 보스 몬스터의 체력이 0 이하로 떨어졌더라도, 클라이언트가 사망 판정을 내려서 월드 보스 몬스터의 사망 액션을 재생하게 만들면 안 된다. **체력만 0 아래로 내려갈 뿐, 전투는 그대로 계속 지속해야 한다.**
: 사망 판정 여부는 무조건 서버로부터 전송 받아야 하며, 서버가 월드 보스 몬스터가 사망했다고 판정한 사실을 클라이언트가 수신한 뒤에야, 클라이언트는 월드 보스 몬스터 사망 이벤트로 넘어가야 한다.

※ 캐릭터 관련 요구사항 항목에서도 자세히 설명하지만, **캐릭터의 사망 여부를 생명력 수치가 0인지 여부로 자동으로 설정하면 안 되는 게** 이런 이유 때문이다.
게임에서는 다양한 이벤트 상의 이유로 인해 사망 판정을 즉시 내리지 않는 경우가 발생할 수 있다. 사망 판정 Flag가 생명력 수치 값과 별도로 동작해야 하는 이유가 이런 것 때문이다.

- **B-35-2-14.** 월드 보스 몬스터 공략이 끝나면, 각 플레이어들은 월드 보스 몬스터를 공략한 순위와 공헌도를 확인할 수 있는 GUI를 제공해야 한다.
: 내 순위도 물론 확인할 수 있어야 한다. 이에 대한 구체적인 인터페이스는 기획 측의 요구사항에 맞게 구현한다.

- **B-35-2-15.** 월드 보스 몬스터 공략에 대한 보상은, 월드 보스 몬스터를 공략한 결과를 확인하는 GUI를 통해 수령할 수 있다.

◆ B-35-3. 길드 전쟁(Guild War, RvR)

- **B-35-3-1.** **최초 출시에는 반영하지 않으며, 이후 업데이트에 반영할 컨텐츠이다.**

- **B-35-3-2.** 아군과 적 길드의 소속 플레이어들이 한꺼번에 같은 전장에서 대결하여, 전멸한 길드 측이 패배하고, 살아남은 인원이 존재하는 길드 측이 승리한다.

- **B-35-3-3.** 길드 전쟁에 참여하는 플레이어들은 사전에 길드 전쟁에 등록해야 한다.

- **B-35-3-4.** 길드 전쟁은 연속적으로 수행할 수 없으며, 한 번 수행한 뒤에는, 다시 길드 전쟁을 수행하기 위해서는 실제 시간으로 일정한 시간이 지나야 한다.
: 상세한 길드 전쟁의 조건은 관련 기획서를 참고한다.

- **B-35-3-5.** 길드에 소속되어 있는 플레이어들만 길드 전쟁에 참여할 수 있다.

- **B-35-3-6.** 길드 전쟁의 전투는 실시간 네트워크 동기화로 구현하지 않는다.

※ PvP 컨텐츠와 유사한 방식이다.
구현 / 서비스 비용에 비추어 보면, 실시간 네트워크 동기화가 그다지 효과적이지 않다.
경험칙에 근거한 이용 비율로 미루어 봐도, 그다지 사용자들이 이용하지 않는 기능이기도 하다.

- **B-35-3-7.** 길드 전쟁은 **길드에 소속되어 있는 모든 플레이어들이 개별적으로 진행**한다.
: 즉, 길드 전쟁의 내용은, 길드 전쟁을 수행한 플레이어들마다 다르며, 승패 역시 나뉠 수 있다.

※ 만약 길드 전쟁이 모든 플레이어가 동시에 참여하거나, 혹은 완전히 같은 내용으로 진행되어야 한다고 설정하면, 어떤 방식이든 추가적인 구현 이슈가 존재한다.
모든 길드의 플레이어가 같은 내용으로 길드 전쟁을 경험하기 위해서는 어느 하나의 기기를 기준으로 잡고(클라이언트 중 하나이든, 혹은 서버이든), 그 기기에서 수행한 전투 결과를 전투에 참여하고 있는 나머지 클라이언트에게 전달해줄 수 있어야 한다.
단, 특정 클라이언트 기기(예를 들면 길드장)에게 기준을 맡긴다고 치면, 그 기기가 24시간 내내 접속한 상태가 아닐 수 있기 때문에, 제 시점에 길드 전쟁을 수행한다는 보장도 없다. (길드장이 2명이기에 어떤 길드장에게 권한을 부여해야 할지 정해야 한다는 점은 생각하지 않더라도 말이다.)

그렇기에 결국 서버가 길드 전쟁의 진행과 결과를 맡아주는 방식이 최선이라고 할 수 있다. (플레이어가 직접게임 방에 참여하는 방식의 실시간 동기화를 하지 않으므로)
그러나, 이 경우, 서버가 길드 전쟁을 진행하기 위해서는 클라이언트에서 진행하는 실시간 전투를 그대로 재현하거나, 혹은 모사할 수 있는 기능이 갖춰져 있어야 한다. 서버에서 클라이언트가 사용하는 Unity3D 엔진의 기능을 직접 이용할 수 없기 때문에, 별도의 자체적인 라이브러리를 이용해야 할 가능성이 높다.
이는 마치, '서버에서는 보이지 않게 돌아가는 전투 이벤트이지만, 클라이언트에서 같은 전투 이벤트를 재생하면 자연스러운 실시간 전투로 보이는' 기능이 필요하다는 논리가 된다. 이와 같은 기능을 구현하는 방법이 없지는 않다. 스타크래프트나 리그오브레전드 등의 게임에서 쓰니는 것처럼, 실시간 턴 기반의 결정론적 알고리즘을 이용해서 전투 플레이가 동작하게 하고, 이에 대한 이벤트를 일정한 규칙에 맞춰 기록하게 하면 가능하다고 본다.
문제는 이에 대한 개발 비용이 기능의 중요도에 비해 과도하게 많다는 점이다. 실시간 턴 기반의 결정론적인 게임 프레임워크나, 이에 따른 물리 계산 코드, 자체 길찾기 기능(이런 방식에서는 Unity에 내장된 길 찾기 메쉬는 못 쓴다.) 등을 구현해야 할 것으로 예상하는데, 이는 게임의 근본적인 시스템을 뒤엎어야 할 정도로 막대한 변경사항들이다. (덤으로 이 모든 건 클라이언트 뿐 아니라 서버에서도 동작해야 하므로, DLL을 이용해 링크할 수 있어야 한다.)

그래서 여러 검토를 거친 끝에, 길드 전쟁이 비록 단체전이기는 하지만, 대상 길드에 속한 각 길드원들이 개별적으로 플레이 할 수 있는 방식으로 제작하기로 결론을 내렸다.

- **B-35-3-8.** 길드 전쟁에 참여했을 때, 플레이어는 자신의 캐릭터를 수동 / 혹은 자동으로 선택해서 조종할 수 있다.
: 직접 조종하고 싶은 사용자들은 수동으로 조종해가면서 싸울 것이고, 만사가 귀찮은(...) 사용자들은 자동 플레이로 결과가 나올 때까지 기다리면 된다.

- **B-35-3-9.** 단, **내 캐릭터를 제외한 아군 길드의 캐릭터들과, 적 길드의 캐릭터들은 그 캐릭터들의 능력 데이터를 갖추고 있는 NPC로써 동작**한다. 전투 역시 NPC의 인공지능으로 이루어지며, 플레이어의 사용자 조종으로 동작하지 않는다.
: 실시간 네트워크 동기화를 하지 않기 때문에 이런 방식으로 진행해야만 한다.

- **B-35-3-10.** 길드 전쟁에는 한 번에 2개의 길드가 참여할 수 있다.
  : 즉, 길드 단위로 보아서는 1 vs 1 길드 전쟁만 지원한다.

- **B-35-3-11.** 길드 전쟁에 참여한 길드는 소속되어 있는 리그 혹은 순위를 기반으로 하여 보상을 얻는다.

- **B-35-3-12.** 길드 전쟁의 보상은, 길드 전쟁에 참여한 길드의 모든 구성원들에게 특수한 부여 효과(Enchanted Effect)를 주는 것이다.
  : 길드 전쟁에 의한 부여효과의 세부적인 능력과, 얼마나 유지될지 여부, 적용 가능한 게임 플레이 모드에 대한 제한 조건은 관련 기획서의 요구사항을 따른다.

B-36. 게임 설정 데이터

◆ B-36-1. 환경 설정 데이터

- **B-36-1-1.** 배경음(Backgound Music, BGM)
  : 슬라이더로 0 ~ 1 사이 값으로 적용한다. (0 = 0%, 1 = 100%)

- **B-36-1-2.** 환경음(FX Music, FXM)
  : 슬라이더로 0 ~ 1 사이 값으로 적용한다. (0 = 0%, 1 = 100%)

- **B-36-1-3.** 최대 초당 프레임 갱신률(Frame rate Per Second, FPS) 조절
  : 30, 45, 60 FPS 사이에서 선택할 수 있게 한다.
  슬라이더 방식은 아니고, 고정되어 있는 상수 값 몇 개 중에서 선택하는 방식이다.

※ 최대 FPS를 조절함으로써, 화면 갱신을 빠르게 하기 위해 필요한 성능을 제한한다. 이렇게 하면 성능에 여유가 있는 기기들은 발열을 줄이고 배터리를 더 오래 가게 할 수 있다.
물론, 최대 FPS의 화면 갱신을 유지하기조차 버겁거나 어려운 성능의 기기들이라면 하드웨어 단계에서 이미 최대 성능을 가동하는 중이기 때문에 그런 거 없다.(...)
(그러니까 FPS 60으로 맞춰놔도 하드웨어 성능이 따라가지 못해서 FPS가 60 밑으로 나온다면, 그건 버그가 아니라는 말이다.)

- **B-36-1-4.** 한 화면에 표시하는 FX 개수의 제한
  : 슬라이더 방식으로 1 ~ 20 사이 값을 선택할 수 있게 하고, 하나도 나오지 않게 하는 경우와, 제한 없이 나오게 하는 경우를 선택할 수 있는 버튼을 둔다.

※ FX들은 거의 다 투명도가 들어간 텍스처 및 파티클들을 사용하기 때문에, 모바일 기기들에게 성능 부담을 심하게 준다. 특히 일부 기기들, 그 중에서도 구형 기기들의 AP(Advanced Processor)들 중에서는 투명도 계산(Alpha Testing)이 아주 느린 경우가 있어서 성능에 결정적인 영향을 주는 경우가 종종 있다.
이들이 한 화면에 표시되는 개수 및 영역, 시간에 제한을 둘수록 게임은 더 원활하게 돌아가고, 성능도 덜 먹으며, 발열도 줄어든다. (물론, 덜 화려해 보일 것이다.)

- **B-36-1-5.** 한 화면에 들리는 사운드 개수 제한
  : 슬라이더 방식으로 1 ~ 20 사이 값을 선택할 수 있게 하고, 하나도 나오지 않게 하는 경우와, 제한 없이 나오게 하는 경우를 선택할 수 있는 버튼을 둔다.

※ 사운드 객체의 등장 개수를 제한하면 성능에 있어 더 유리하다.
특히, 3D 방식의 사운드 객체는 사운드 수신자(Sound Listener)까지의 거리와 방향 등을 계산하고 사운드가 잦아드는 도플러 효과까지 먹이기 때문에 생각보다 연산이 꽤 세다.

- **B-36-1-6.** 사운드 품질 설정(고려만 하고 있는 사항)
: 사운드 객체는 메모리 성능에도 영향을 주긴 하지만, 메모리를 많이 먹는 주체는 사운드 객체 그 자체보다는, 오디오 클립 데이터의 품질에 더 영향을 많이 받는다. 따라서, 사용자는 조악하지만 더 빠르고 메모리를 덜 차지하는 오디오 데이터와, 고품질이지만 더 느리고 메모리를 더 차지하는 오디오 데이터 중에서 선택을 하고 싶어할 수도 있다.

※ 이런 기능을 지원하려면, 사용자가 선택한 옵션에 따라 데이터 다운로드를 다르게 설정할 수 있는 기반 기능이 있어야 한다.

◆ B-36-2. 사용자 정보 저장

- **B-36-2-1.** 클라이언트에서 가지고 있는 정보들이다.
: 클라이언트가 삭제되거나, 기기가 바뀌면 초기화되는 정보들이다.

- **B-36-2-2.** 마지막으로 접속한 서버의 주소

- **B-36-2-3.** 마지막으로 접속한 계정 ID

- **B-36-2-4.** 마지막으로 접속한 계정의 패스워드
: 단, 패스워드는 반드시 평문이 아닌, 암호화한 형태로 저장해야 한다.

※ 로그인할 때, 패스워드를 입력하는 기능은 평문으로 직접 입력하는 경우를 처리할 때와, 이미 암호화된 패스워드를 전달해서 처리하는 경우 두 가지를 지원해야 한다.

- **B-36-2-5.** 마지막으로 설정한 게임 플레이 환경 정보들
: 상세한 종류는 게임 플레이 환경 정보에 대한 명세를 참조한다.

◆ B-36-3. 게임 플레이 정보 저장

- **B-36-3-1.** 클라이언트에서 가지고 있는 정보들이다.
: 클라이언트가 삭제되거나, 기기가 바뀌면 초기화되는 정보들이다.

- **B-36-3-2.** 마지막으로 플레이 했던 스테이지의 코드 번호

- **B-36-3-3.** 마지막으로 플레이 했던 캐릭터 선택 슬롯

◆ B-36-4. 저장 데이터의 버전 관리

- **B-36-4-1.** 저장 데이터의 형식은 향후 패치에 따라 바뀔 수 있다.
  : 없었던 새로운 능력치가 생길 수도 있고, 저장 구조나 순서가 바뀔 수도 있다.

- **B-36-4-2.** 저장 데이터의 구조는 변경될 때마다 버전을 부여해서 관리해야 한다.

- **B-36-4-3.** 애플리케이션은 모든 버전의 저장 데이터에 대해 불러들일 수 있어야 한다.

※ 그러므로, 저장 기능을 구현할 때, 저장 데이터 구조의 버전 별로 불러들일 수 있도록 코드 조를 설계해야 한다. 또한, 새로운 저장 구조로 개편되더라도, 구 버전의 저장 데이터를 불러들이는 코드를 삭제하거나, 영향이 가지 않게 코드 구조를 짜야 한다.

- **B-36-4-4.** 구 버전의 저장 데이터를 불러들이면, 현재의 최신 버전 저장 데이터 구조로 변환한다.

- **B-36-4-5.** 최신 버전의 저장 데이터 구조로 한 번이라도 변환된 저장 데이터는, 다시는 구 버전의 저장 구조로 저장할 수 없다.

※ 단, 가능하다면 구 버전의 저장구조로 변환하는 기능 자체는 만드는 것이 좋다.
개발 및 테스트를 하다 보면, 구 버전의 실행 파일에서 테스트 / 디버깅해야 할 때가 많은데 그 때 유용하게 써먹을 수도 있기 때문이다. 물론, 최종 사용자(플레이어)에게 이 기능을 직접 제공하지는 않을 것이며, 그저 개발자용 기능이다.

C. 기반 시스템

C-1. 시스템 사양

◆ C-1-1. 지원 기종 및 하드웨어 조건

- **C-1-1-1.** 지원할 하드웨어는 다음과 같다.
  : **아래에 명시한 하드웨어 외에는 공식적으로 지원하지 않는다.**

플랫폼	지원여부	최소 기준 사양	최대 기준 사양
Android(Phone)	○	Galaxy S2 동급	제한 없음
Android(Tablet)	○	나중에 추가함	제한 없음
iOS(Phone)	○	iPhone 5	제한 없음
iOS(Tablet)	○	iPad2	제한 없음

※ 지원 대상 플랫폼은 모바일 플랫폼만을 기준으로 한다.

- **C-1-1-2.** 일반 사용자를 대상으로 하는 클라이언트는 모바일 기기만을 지원한다.

- **C-1-1-3.** 가상 하드웨어에 대해 공식적으로 지원하지 않는다.

◆ C-1-2. 운영체제

- **C-1-2-1.** 지원할 운영체제의 범위는 다음과 같다.

운영체제	최소 지원
Android	3.0 이상
iOS	7.0 이상

◆ C-1-3. 그래픽스 API

- **C-1-3-1.** 공식적으로 지원할 그래픽 API는 OpenGL ES를 기준으로 한다.

- **C-1-3-2.** OpenGL ES의 최소 지원 버전은 **2.0** 이상이다.

※ 목표 장치(Target Device)에 대한 지원 버전 범위는, 출시 당시의 **Unity3D 엔진이 지원하는 한계 범위 이내**다.
참고로, 현재 Unity 엔진의 버전(4.6) 기준으로 OpenGL ES 2.0 이하 버전은 지원 목록에서 사라진 상태다.

- **C-1-3-3.** 최소 지원 API 버전보다 높은 경우에는, 선택적으로 적용할 수 있게 한다.
: 즉, 사용이 가능하다면 그것을 지원하고, 아니라면 최소 지원 API로 작동하게 한다.

- **C-1-3-4.** 단, 최소 지원 API 버전에서 지원하지는 기능을 이용해서 핵심 컨텐츠를 구현하지 않는다.
: 그렇지 않다면, 하드웨어가 지원 가능한지 여부에 따라 리소스가 나뉘어야 할 수도 있고, 여러 가지 복잡한 빌드 분리 공정이 추가되어야 한다. 당연히, 개발 비용도 상승한다.

◆ C-1-4. 네트워크

- **C-1-4-1.** 게임을 실행하는 기기들은 반드시 무선 또는 유선 네트워크를 통해 웹에 접속할 수 있어야 한다.
: 반드시 사용자 계정에 대한 인증을 받아야 게임에 접속할 수 있기 때문이다.

- **C-1-4-2.** 모바일 기기에서 게임을 실행하는 경우, 그 모바일 게임은 반드시 3세대 이상의 모바일 네트워크가 지원되는 기기여야 한다.
: 소위 '피처폰'으로 불리는 2세대 이하 모바일 네트워크를 지원하는 기기에서는 이 게임을 이용할 수 없다.

※ 뭐, 이런 경우는 Unity 엔진에서 해당 운영체제나 API를 지원하지도 않기 때문에, 앱을 만들어 줄 방법도 없긴 하다.
하지만, 혹시 Android 기반의 2세대 이동통신 지원 기기가 해외에 있을지도 모르니, 확인이 필요하다. (그런 건 없을 거야, 아마...;)

◆ C-1-5. 특수 장치

- **C-1-5-1.** 진동 장치가 존재하는 기기들은, 진동 장치를 사용할 수 있다.

- **C-1-5-2.** 카메라 기능을 사용하거나 지원하지 않는다.
: 이 기능이 필요한 기획 요소가 존재하지 않으며, 앞으로도 추가할 생각이 없다.

- **C-1-5-3.** 중력 센서, 자이로스코프 기능을 사용하거나 지원하지 않는다.
  : 이 기능이 필요한 기획 요소가 존재하지 않으며, 앞으로도 추가할 생각이 없다.

- **C-1-5-4.** 음성 인식 기능을 사용하거나 지원하지 않는다.
  : 이 기능이 필요한 기획 요소가 존재하지 않으며, 앞으로도 추가할 생각이 없다.

- **C-1-5-5.** 저장 장치를 제외하고는, 게임을 실행하는 데 있어, 실행 기기 외의 다른 어떠한 물리적인 장치에 대한 연결을 필요로 하지 않는다.
  : 저장 장치는 외장 메모리를 사용하는 경우 때문에, 선택적으로 적용할 수 있어야 한다.

C-2. 성능 요소

◆ C-2-1. 시스템 제한사항

- **C-2-1-1.** 응용 프로그램의 구동에 필요한 최소 사양은 아래와 같다.

1. 가로 4 : 세로 3 이상의 화면 비율.
: 앱을 구동할 때, Landscape 방향(장치의 화면 길이 중에 더 긴 쪽을 가로로 삼는 방식)을 기준으로 구동한다.

2. 768MB 이상의 시스템 메모리
: 운영체제가 점유하고 있는 메모리를 제외한, 응용 프로그램을 구동할 수 있는 전체 메모리 공간을 말한다.

3. 125MB 이상의 가용 메모리

4. ~~50MB~~100MB 이상의 응용 프로그램 저장을 위한 기기 본체의 저장 공간

4. 500MB 이상의 데이터 저장을 위한, 기기 본체 또는 외장 저장 장치의 저장 공간

- **C-2-1-2.** 이 게임 애플리케이션은 1개의 기기에서 동시에 1개만 구동할 수 있다.
: 운영체제가 지원하는지 여부와 관계없이, 응용 프로그램은 반드시 동시에 한 번에 1개만 실행시킬 수 있다.
그 외의 상황에 대해서는 실행 결과에 대해 아무 것도 보장하지 않는다.

※ 애플리케이션의 다중 실행을 지원할 계획은 지금은 물론이거니와, 앞으로도 없다. (이런 게 왜 필요해?)
사실, 대부분의 모바일 기기들에서는 운영체제 상 이유로 인해 그렇게 할 수 없기도 하다.

◆ C-2-2. 속도 관련

- **C-2-2-1.** 원활한 플레이를 할 수 있다고 판단하는 기준이 되는 초당 프레임 수(Frame Per Second)는 **30FPS**이다.

: 즉, 게임이 구동되는 도중에 화면에 등장하는 캐릭터 애니메이션, 이펙트 애니메이션 등은 최소한 초당 30 회의 재생률을 유지할 수 있어야 한다.

- **C-2-2-2.** 게임 판정의 기준이 되는 최소 판정 주기(Cycle)는 **초당 15회**이다.
: 응용 프로그램을 구동하는 하드웨어의 성능이 아무리 좋더라도, 초당 15 회 이상 판정을 하지 않는다.

※ 모바일 게임이기 때문에, 기기를 빡세게 굴리기보다는, 적당히 무리가 없는 최소한의 기준치에 맞춰서 게임 플레이 및 판정 프로세스를 돌린다.
이렇게 하는 편이 배터리 소모도 더 적을 것이므로, 사용자들이 좀 더 게임에 접근하기 편안하게 만들어 줄 것이다.

- **C-2-2-3.** FX나 급격한 객체 생성으로 인해, 일시적으로 FPS가 30 미만으로의 하락할 수도 있다. 다만, 이러한 하락은 **2초 이내**에 본래의 재생률로 복구해야 한다.
: 지킬 수 있을까...(ㅇㅅㅇ;)

※ 시각 FX들은 투명도를 가지는 텍스처들을 그려야 할 경우가 많은데, 이는 Draw Call을 급격히 상승시킨다. 또한 투명도 테스트(Alpha Testing) 그 자체가 그래픽 프로세서의 부담이 큰 작업이기도 하다.
다만, FX 들은 상대적으로 매우 짧은 시간 동안 유지되는 경우가 일반적이기 때문에, 짧은 시간에 한한다면 FPS 가 잠시 기준점 아래로 하락하는 경우도 허용할 수 있다. 단, 이런 경우가 너무 자주 발생해서는 안될 것이다. 이를 위해 한 번에 화면에 표시하는 FX 의 개수를 제한하거나, 혹은 객체 생성 과정을 미리 해놓고 재활용하는 방안 등을 동원해야 할 수 있다.

◆ C-2-3. 용량 관련

- **C-2-3-1.** 실시간으로 메모리를 점유하는 크기는 최대 **350MB** 이내가 될 것을 권장한다.

※ 사실은 이조차도 매우 낙관적으로 잡은 것이다.
실제 iOS, Android 모바일 기기들 중 구형인 경우, 시스템 메모리 전체 용량이 512MB 이하인 경우도 많으므로, 100MB 이상의 실시간 메모리 점유율을 보일 경우, 화면 전환 시 운영체제가 곧장 자동으로 앱을 꺼버리는 등의 문제점이 발생할 수 있다.

※ 유념할 점이 하나 더 있는데, **같은 앱에 대해서도 구형 모바일 기기와 최신 모바일 기기에서의 실행 메모리를 점유하는 양이 다를 수 있다**는 점이다.
최신 기기에서는 300 ~ 500MB 가까이 실행 메모리를 점유하는 응용 프로그램일지라도, 구형 모바일 기기에서는 고작 70 ~ 120MB 정도의 실행 메모리를 점유하는 경우가 있다. 상세한

조사가 필요하겠지만, 이는 아마도 실행 메모리 공간에 여유가 있는 경우, 빠른 데이터 접근을 위해서 운영체제가 더 적극적으로 메모리 캐싱을 하기 때문일 수 있다.
따라서, 이 조항의 내용을 문자 그대로 모든 기기에 적용해야 한다고 가정하면 곤란하다고 본다. 현재 대중적인 모바일 기기에서 대략적으로 필요한 한계치 정도를 제시한 것으로 판단하면 된다.

- **C-2-3-2.** 스토어에 등록하는 앱의 최종 용량이 ~~50MB~~100MB를 넘어서는 안 된다.

※ Google Play 에서 Wi-fi 를 이용하지 않는 다운로드(Download On The Air) 용량은 ~~50MB~~100MB 까지가 한계치이다.
따라서, 게임은 최소 실행 부분까지만 마켓에 등록하고, 그 외 추가적인 리소스들은 모두 자체적인 리소스 패치 / 다운로드 기능을 구축해서 이루어져야 한다.

참고로, Google Play 에 앱을 등록할 경우, 파일 분할을 이용해서도 개당 2GB 이상, 앱 당 최대 4GB 이상의 데이터를 탑재할 수 없다는 제한 사항이 있다.
(Android 5.0 롤리팝 이후에는 64Bit 가 지원이 되기 때문에, 이와 같은 정책도 달라질 가능성이 있다.)

◆ C-2-4. 디스플레이 관련

- **C-2-4-1.** 화면 비율이 **4:3, 16:9, 5 : 3** 화면에서 잘리는 영역이 없이 모든 게임 화면이 보여야 한다.

※ 모바일 기기에서 가장 표준적으로 사용하는 해상도에서 문제 없이 게임 화면의 내용들을 표시할 수 있어야 한다.

- **C-2-4-2.** 스마트 기기에서 **길이가 더 긴 쪽(Landscape)이 게임 화면의 가로**가 되고, **길이가 짧은 쪽(Portrait)은 게임 화면의 세로**가 된다.

※ 명세서 전체에서 게임 화면의 가로 / 세로에 대해 언급할 때는 이 내용이 적용된다고 봐도 좋다.

- **C-2-4-3.** 장치를 뒤집었을 때, 화면이 반전되는 기능
: Landscape 방향(=장치의 긴 쪽이 가로가 되는 방향)으로만 적용한다.
Landscape -> Portrait 혹은 그 반대의 화면 전환 기능은 지원하지 않는다.

※ 이 게임은 디스플레이가 Portrait 방향일 때의 게임 환경을 지원하지 않는다.
Landscape 왼쪽이든 오른쪽이든 간에, 반드시 Landscape 방향으로 플레이 해야 한다.

◆ C-2-5. 프로그램 내부 값 한계 관련

- **C-2-5-1.** 특별히 사용하는 값의 자료형과 한계값을 명시하지 않는 한, 모든 수치형(Numerical) 값들의 한계는 **암묵적으로 정수형과 실수형을 통틀어 4바이트의 수치 한계가 기준**이다.

- **C-2-5-2.** 사용자가 입력할 수 있는 문자열은 **최대 255글자**의 한계를 가진다.
  : 글자 수로 255자라는 점에 유의할 것.
  이는 사용자가 문자열 입력 등으로 과다한 네트워크 부하를 발생시키거나, 혹은 이를 이용한 공격에 악용되는 것을 방지하기 위함이다.

※ 텍스트 인코딩에 따라, 유니코드 1 글자는 1 바이트부터 시작해서 최대 6 바이트까지도 필요할 수 있다. (대부분의 일상적인 경우에는 3 바이트 이내로 끝난다.)
즉, 위 경우는 바이트 수로 따지면 보수적으로 글자당 4 바이트로 잡는다고 해도 1024 바이트(1KB)로 입력 문자열 한계를 지정한다.

※ 또한, 이 사항은 게임 애플리케이션이 제공하는 문자열에 대해서는 해당하지 않는다. 오직 사용자가 키보드 / 키패드 등으로 입력하는 문자열들에 대해서만 적용하는 사항이다.

- **C-2-5-3.** **32비트** 실수 값은 **소수점 셋째 자리**까지의 값만 인정한다.
  : 소수점 셋째 자리까지 같은 두 32비트 실수는 같다고 판정해야 한다.

- **C-2-5-4.** **64비트** 실수 값은 **소수점 여섯째 자리**까지의 값만 인정한다.
  : 소수점 여섯째 자리까지 같은 두 64비트 실수는 같다고 판정해야 한다.

- **C-2-5-5.** 실수 값을 사용하는 경우, 유의미하다고 인정하는 소수점 자리 수 이하의 값들은 버린다.
  : 이 조건을 모든 실수 값들에 대해 모든 계산에 앞서 적용할 것인지, 아니면, 어딘가에 저장할 시점의 값에만 적용할 것인지 여부는 결정을 해봐야 한다. 약간, 값이 달라질 가능성이 있기 때문이다.
  현재로써는 후자의 방식에 무게를 두고 있다. 어쨌든 부동소수 구현에 관한 사항들은 IEEE 표준으로 정의하고 있고, 모든 구현(하드웨어든, 소프트웨어든)이 이를 따르도록 되어 있는데다, 매번 실수 값들의 Epsilon 이하 값을 잘라내는 작업을 하는 연산량이 과하게 누적될 때의 성능 요소도 마냥 무시할 수는 없는 문제라고 판단한다.
  (다만, Epsilon 이하 숫자를 잘라내는 작업이 많을 때, 정말로 성능 문제가 유의미하게 나타나는지는 검증되지 않았다.)

- **C-2-5-6.** 클라이언트에서는 32비트 미만의 자료형들을 수치 계산에 사용하지 않는다.

: 개발 언어인 C#에서, 32비트 미만의 자료형들은 기본적으로 int 타입으로 캐스팅하지 않으면, 수치 계산하기가 번거롭다.

- **C-2-5-7.** 수치를 32비트 미만 자료형으로 사용하는 경우는 **네트워크 통신과 데이터베이스 저장 단계**로 제한한다.

※ 일반적인 경우에서, CPU는 int 형을 가장 빠르게 처리할 수 있다.
네트워크 패킷으로 데이터를 주고받는 경우와, DB 저장의 경우에는 1바이트 단위의 최적화가 민감한 주제이기 때문에, 최적화된 자료형을 사용하는 게 의미가 크다.
그렇지만, 클라이언트는 상대적으로 코드 논리의 수행 속도가 좀 더 중요하며, 이를 위해서 약간 자료형이 공간을 좀 낭비하더라도 충분히 감수할 수 있는 경우가 많다. 이는 현재의 모바일 기기에서도 마찬가지다. (이미 2015년 이후로, 프리미엄 급 모바일 기기들은 2, 3GB의 실행 메모리를 기본으로 장착하고 출시된다.)

C-3. 장치 입력 / 출력 인터페이스

◆ C-3-1. 제한 사항

- **C-3-1-1.** 모든 사용자 입력은 모바일 기기 기준으로 설명한다.

※ PC 버전을 전혀 지원하지 않을 것은 아니지만, 이는 개발 단계에서의 테스트 용, 혹은 관리자 클라이언트를 위한 특수한 모드로 간주하기 때문에, 별도의 사용 장치에 대한 조건을 명시하지 않는다. (그때그때 꼭 필요한 입력 방식만 지원한다는 뜻임.)

- **C-3-1-2.** 모바일 하드웨어를 다른 장치에 연결해서 사용하는 경우의 입 / 출력 인터페이스에 대해서는 정의하지 않는다.

※ 모바일 하드웨어를 다른 장치에 연결해서 플레이하는 경우는 '비공식적인 방식의 사용'으로 간주한다.
따라서, 이와 같은 경우에 대해서, 입 / 출력 작동이 정확함을 보장해 줄 의무는 없다.

- **C-3-1-3.** 가상 운영체제를 이용해 애플리케이션을 작동시키는 경우, 이에 대한 장치 입력 / 출력에 대한 작동을 보장하지 않는다.

◆ C-3-2. 작동 조건

- **C-3-2-1.** 인터럽트가 발생하여, 응용 프로그램의 동작이 잠시 중단되는 기능은 목표 장치의 OS와 게임 엔진이 이미 구현한 사항을 준수한다.

※ 대상 장치의 운영체제 및 Unity 엔진 단계에서 지원한다면, 특별히 안 건드리겠다는 말이다.

- **C-3-2-2.** 터치 방식으로 동작하는 장치(Smart Phone, Tablet 등)에서는 다중 지점 터치 기능을 지원해야 한다.
: 인식할 터치 지점 개수는 최소 2개 이상이어야 한다.

※ 2개 이상의 지점을 터치할 경우에, 장치 인터페이스 상에서 별도의 입력으로 간주하고 별개의 이벤트로 인식하고 처리할 수 있는 기능이 갖추어져야 한다.

그러나, 모든 사용자 입력 및 UI 이벤트가 멀티 터치 기능을 지원해야 한다는 의미는 아니다. 필요에 따라, 그렇게 구분해서 사용할 수 있어야 한다는 의미다.

◆ C-3-3. 조작성 조건

- **C-3-3-1.** GUI에 대한 조작 결과를 기다릴 때, 이 기다림이 **2초 이상** 걸릴 경우, 기다림을 알리는 커서를 표시한다.

- **C-3-3-2.** 네트워크 연결 시간이 **2초** 이상 걸릴 경우, 기다림을 알리는 커서를 표시한다.

- **C-3-3-3.** 네트워크 접속이 불가능할 경우, 다시 시도할지, 아니면 게임을 종료할지 선택할 수 있게 하는 **옵션 메뉴를 제공**한다.

◆ C-3-4. 입력 조건

- **C-3-4-1.** 터치 상태를 유지하는 경우, 연속적인 입력이 들어가는 기능
  : 즉, 누름 상태를 계속해서 유지하는 경우인데, 이 때는 연속적으로 터치 입력이 전송된 것처럼 처리할 수 있어야 한다.
  이 때, 터치 입력의 전송 주기는 필요에 따라 최소 주기를 제한할 수 있어야 한다.

※ 주로 일반 공격 명령을 처리할 때, 사용자 편의를 위해 이런 기능을 사용하곤 한다.
여러 번 터치를 하지 않고, 터치를 유지하기만 해도 그 시간 동안 여러 번 일반 공격 명령을 내린 것처럼 인식해서, 사용자가 계속 인식 패널을 두들겨야 하는 피로감을 덜 느끼면서 적을 공격할 수 있게 한다.

- **C-3-4-2.** 한정된 시간 내에 입력된 명령을 기억하는 기능, 또는 사용자가 내린 마지막 명령을 예약하고 있다가 수행하는 기능
  : 모든 사용자 입력이 즉각적으로 결과를 내지는 않는다. 게임 플레이의 상황에 따라, 사용자의 입력 명령을 무시해야 할 경우도 있다.
  그렇지만, 사용자의 입력 명령이 '어느 정도' 합당한 시점에 가까운 시점에 내려졌다면, 완전히 정확한 시점에 내린 입력 명령이 아닐지라도 이를 '기억'해두었다가, 적절한 시점이 오면 수행하게 해주는 기능이 필요하다.

※ 사용자가 '대략적으로' 적절한 시점에 넣은 명령을 칼같이 무시하게 처리하면, 사용자는 스스로 정확한 입력 가능 시점을 판단해가면서 계속해서 같은 입력 명령을 넣어야 하기 때문에 피로감이 심해진다.
또는, 마지막 입력 명령에 대한 결정을 번복하고 싶어서 다른 입력으로 '교체했으면' 하고 바

라는 경우 또한 있을 수 있다.
그렇기 때문에, 플레이어가 마지막으로 입력한 명령에 대해 기억하고 있다가, 수행 가능한 시점이 되면 기억해 둔 마지막 입력 명령을 수행하는 기능이 있는 게 좋다. 이런 기능은 플레이어로 하여금 게임이 지능적으로 자신의 입력에 반응한다고 느끼게 할 것이며, 플레이어 자신이 게임 플레이를 통제하고 있다는 감각을 더 잘 살릴 수 있을 것이다.

- **C-3-4-3.** Modal 기능
: 어떤 GUI들은 플레이어가 확인하거나 종료하기 전에는 다른 입력 명령들을 전부 무시하게 할 수 있어야 한다.

※ 대표적으로 응용 프로그램 내부의 시스템 메시지들은 팝업 윈도우의 형태로 보여줘야 하는 경우가 있다.
게임의 종료 확인 메시지, 필수적인 네트워크 연결 대기를 알릴 때는 그 작업이 완전히 수행이 완료되거나, 혹은 취소되거나, 사용자가 확인하고 그 팝업 윈도우를 닫기 전에는 다른 어떤 입력 명령도 수행해서는 안 된다.
또한, 그러한 입력 명령의 차단은 응용 프로그램의 수행을 유지하는 상태와는 별개이다. 사용자의 입력 명령은 차단해야 하지만, 응용 프로그램은 계속해서 수행하고 있어야 하는 상황도 있을 수 있기 때문이다.

- **C-3-4-4.** Modaless 기능
: 플레이어에게 알리기 위해 GUI가 띄워지지만, 다른 메뉴로의 전환이나 사용자 입력을 막지는 않는다.
이 프로젝트에서는 Modaless 기능이 필요할 것으로 판단하지 않아서, 이에 대한 기능은 구현하지 않기로 한다.

◆ C-3-5. 사용자 입력 처리에 대한 일반적인 방식

- **C-3-5-1.** 모바일 기기에서는 터치 입력 외의 입력 방식은 지원하지 않는다.
: 음성을 통한 입력, 중력 센서를 통한 입력, 기타 터치 입력이 아닌 장치의 특수 기능을 이용한 입력 등이 모두 해당된다.

- **C-3-5-2.** 내 캐릭터의 방향 이동은 가상의 조종 패드 GUI를 띄워서, 그 패드 볼을 굴리는 방향으로 이동하는 방식을 사용한다.

<조종 패드로 사용하는 GUI의 외형 예시>

※ 가상 조종 패드 GUI 의 '볼' 부분이 어떤 시점일 때 캐릭터가 이동할지, 가상 조종 패드 GUI 자체가 항상 보일지, 터치 시에만 보여야 할지 여부 등은 플레이하는 사용자의 '느낌'에 따라 다르다.
이 부분은 현재 확정하기는 어렵고, 사용성 테스트 도중의 요구사항을 반영하도록 한다.

- **C-3-5-3.** 게임의 장면이나 메뉴를 전환하는 방식은, 대체로 지정된 버튼 GUI를 터치하거나 누르고 있어야 한다.

- **C-3-5-4.** 장치 화면의 영역보다 큰 GUI를 표시하는 경우, 사용자는 터치 입력을 유지한 채 그 입력을 '끌어서' 장치 화면보다 큰 GUI 영역을 돌아다닐 수 있다.
: 스크롤(Scroll) 혹은 패닝(Panning) 기능인데, 사용성에 필요한 경우, 이런 기능을 지원할 수 있어야 한다.

C-4. 그래픽 사용자 인터페이스(GUI)

◆ C-4-1. 화면 비율에 따른 GUI의 처리

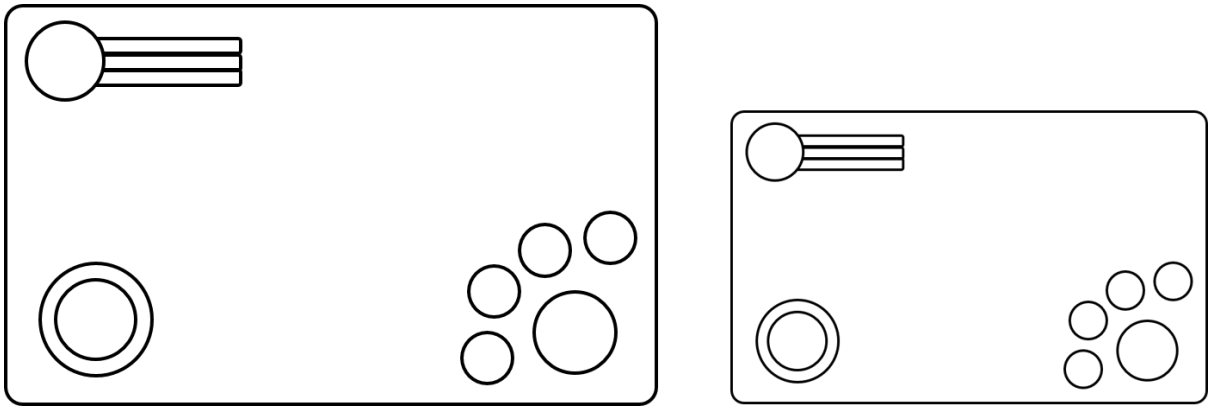
- **C-4-1-1.** 지원해야 할 것으로 예상하고 있는 주요한 화면 비율은 다음과 같다.

화면 비율 (landscape)	해상도	기기
16 : 9	1136 × 640 1280 × 720 1920 × 1080 2560 × 1440	기기의 사실상 표준적인 화면 비율
5 : 3	800 × 480	Samsung Galaxy S2 등
3 : 2	960 × 640	LG Optimus View 등
4 : 3	1024 × 768 2048 × 1536	Apple iPad 등

- **C-4-1-2.** GUI를 디자인하는 해상도는 기본적으로 **16 : 9 화면 비율을 기준**으로 작업한다.
: 전술했듯, 현재 일부 기기를 제외한 모바일 기기의 화면 비율은 이 비율로 통일되는 추세다.
(iPad 가 있는 Tablet 기기 시장은 약간 사정이 다를 수 있지만, 16 : 9 이외의 기준을 별도로 둘 필요까지는 없다고 본다.)

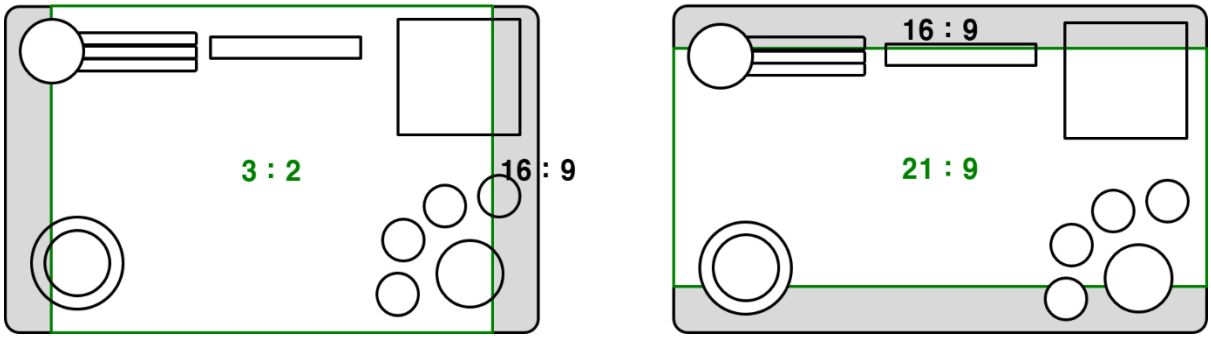
- **C-4-1-3.** 같은 화면 비율 내에서, 화면 영역을 차지하는 GUI들의 상대적인 크기는 해상도의 절대적인 픽셀 수에 관계 없이 동일하게 보여야 한다.

※ 2560 × 1440(QHD 급)와 1280 × 720(HD 급) 해상도는 화면 비율은 둘 다 16 : 9 이지만, 이 비율에 맞춘 GUI 들이 실제로 화면에서 점유하고 있는 픽셀 수에서는 많은 차이가 난다.
이런 경우에, GUI 들이 화면에서 보이는 상대적인 크기 비율이 모두 같게 한다는 뜻이다.
즉, 같은 GUI 인 경우에는, QHD 해상도에서 그려낸 GUI 는 HD 해상도에서 그려낸 GUI 보다 차지하는 픽셀 개수가 더 많을 것이다. 하지만, 둘 다 기기의 화면 영역에서는 같은 비율의 크기로 보일 것이다.

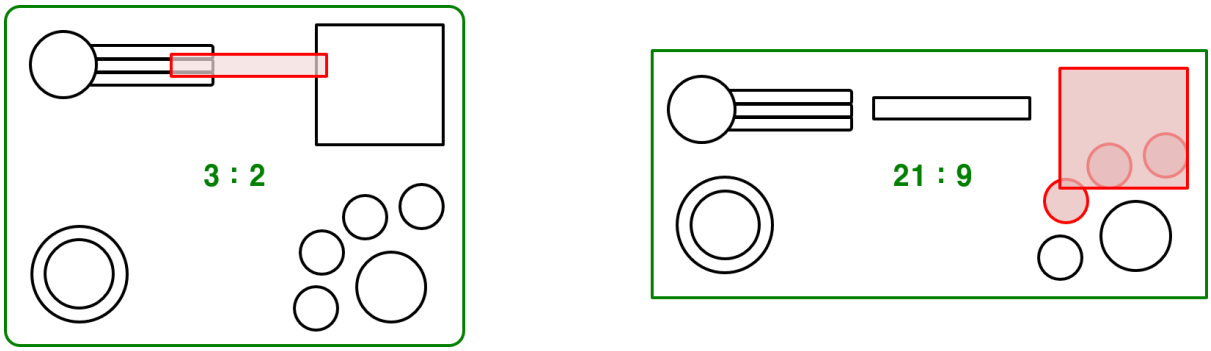


< 화면의 비율이 같고, 크기 차이만 있을 때 >

- **C-4-1-4.** GUI 디자인의 기준이 되는 화면 비율(16 : 9)과 대상 기기의 화면 비율이 다른 경우에는, **개별 GUI들이 좁은 화면 영역에 모여서 겹치게 되거나, 혹은 일부 GUI가 화면 바깥 영역에 그려지는 현상을 방지**해야 한다.

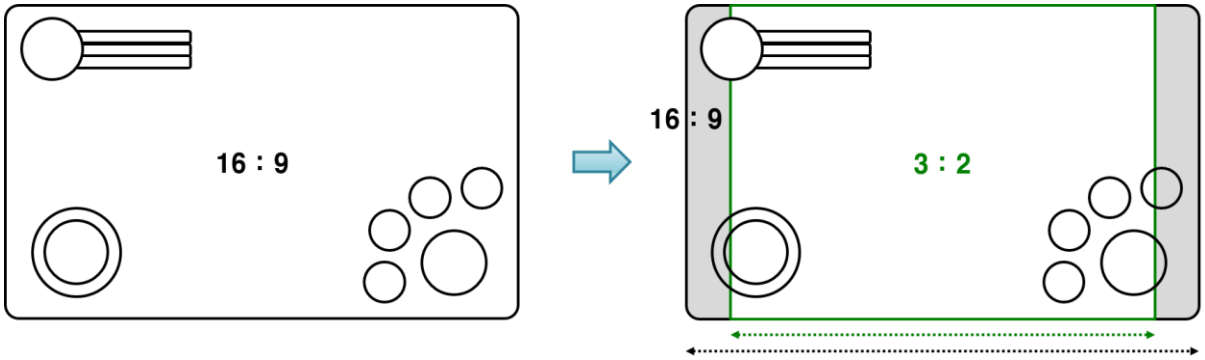


< 화면 비율로 인해, GUI 들의 일부가 화면 영역 밖에 위치하게 되는 현상 >



< 화면 비율로 인해, 좁은 공간에 GUI 들이 몰려서 겹쳐 버린 현상 >

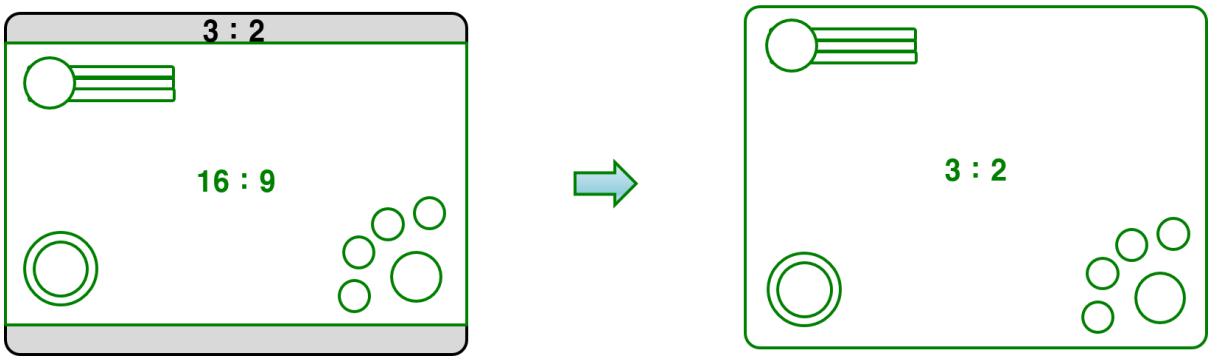
- **C-4-1-5.** GUI 작업 기준의 해상도보다, 대상 기기의 화면 비율이 더 짧은 경우(즉, 화면의 가로 길이와 세로 길이의 차이가 덜 나는 경우)의 처리 방식
: **화면 크기의 세로 맞춤**으로 처리한다.
즉, 상대적으로 좁아진 이전과 현재 화면의 가로 길이에 대한 비율을 구하고, 이를 화면 상의 GUI 들의 크기 비율에 일괄 적용한다.



< 화면이 원래보다 더 좁아졌다. >

기준 비율로부터, 현재 화면이 가로 방향으로 얼마만큼의 비율로 더 좁아졌는지 계산하여, 이를 전체 GUI 들의 크기 배율에 반영한다.

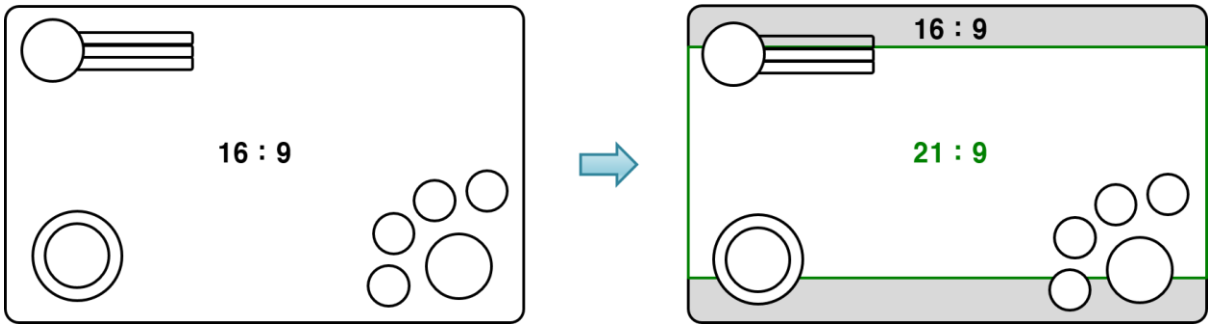
이렇게 하면, 줄어든 가로 길이의 비율만큼 모든 GUI 들의 크기가 같이 줄어들기 때문에, 상대적으로 더 좁아진 가로 길이에도 불구하고 GUI 들이 가로 방향으로 겹칠 일이 없다.



< 크기 비율을 변경한 GUI 들을 각각의 기준점(Anchor)에서의 상대 위치로 변경한다. >

최종적으로는 위에서 보듯, GUI 들이 기준점(0, 0)으로 삼고 있는 화면 상의 방향으로 상대 위치를 이전과 같이 맞춰주면, 화면의 세로 방향으로도 알맞게 GUI 들을 퍼뜨릴 수 있다.

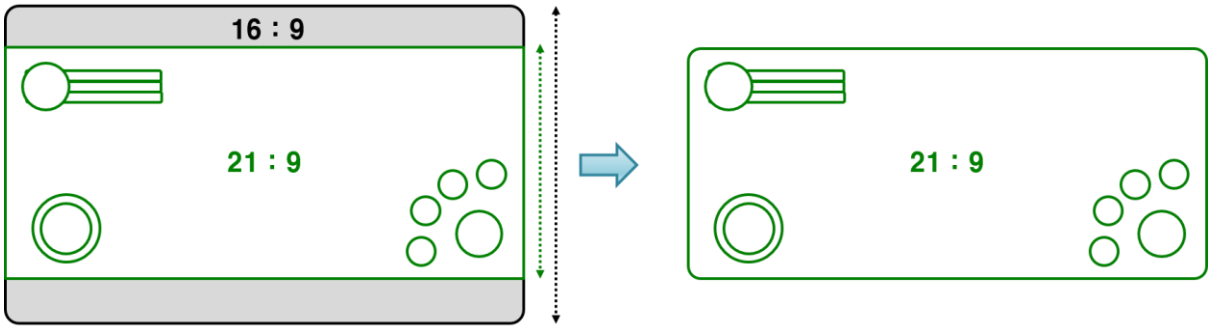
- **C-4-1-6.** GUI 작업 기준의 해상도보다, 대상 기기의 화면 비율이 더 길쭉한 경우(즉, 화면의 가로 길이와 세로 길이의 차이가 더 나는 경우)의 처리
: **화면 크기의 가로 맞춤**으로 처리한다.



< 화면이 원래보다 더 길쭉해졌다. >

기준 비율로부터, 현재 화면이 세로 방향으로 얼마만큼의 비율로 더 좁아졌는지 계산하여, 이를 전체 GUI 들의 크기 배율에 반영한다.

이렇게 하면, 줄어든 세로 길이의 비율만큼 모든 GUI 들의 크기가 같이 줄어들기 때문에, 상대적으로 더 좁아진 세로 길이에도 불구하고 GUI 들이 세로 방향으로 겹칠 일이 없다.



< 크기 비율을 변경한 GUI 들을 각각의 기준점(Anchor)에서의 상대 위치로 변경한다. >

최종적으로는 위에서 보듯, GUI 들이 기준점(0, 0)으로 삼고 있는 화면 상의 방향으로 상대 위치를 이전과 같이 맞춰주면, 화면의 가로 방향으로도 알맞게 GUI 들을 퍼뜨릴 수 있다.

- **C-4-1-7.** GUI 작업 기준의 해상도와 화면 비율과, 실행하는 기기의 화면 비율과 해상도가 가로 세로 전부 다른 경우
: GUI 작업 기준이 가로 : 세로가 16 : 9 비율로 설정하고 있으므로, **실행하는 기기의 화면 비율에서 16 : 9 화면 비율이 가능한 가장 큰 화면 맞춤의 방향**을 찾는다.
그러면 가로 맞춤, 혹은 세로 맞춤이 된다. 이 맞춤 방식을 적용한다.

◆ C-4-2. 문자열의 처리

- **C-4-2-1.** 문자를 그림으로 표현해야 하는 경우, **문자를 표현하는 텍스처와 배경을 표현하는 텍스처는 반드시 분리**해서 제작한다.

※ 흔히 버튼에 사용하는 문자를 좀 더 예쁘게 표현하려고 텍스처로 만드는 경우가 있는데, 이걸 버튼 배경 텍스처와 합쳐서 통으로 제작하는 방식은 심각한 실수가 될 수 있다. 문구가 한 글자라도 달라질 때마다, 비슷하지만 문구만 다른 텍스처를 계속 제작해줘야 하기 때문이다.
심지어 지역화 과정의 일환으로, 번역한 문구를 넣은 버튼으로 바꿔야 한다고 가정한다면... 게임 리소스 양을 불필요하게 늘리는 결과를 가져온다.

- **C-4-2-2.** 문자를 그림으로 표현할 때, 같은 그림인데 색상만 다른 경우에는, 각 색상마다 텍스처를 만들지 말고, 기준이 되는 문자 텍스처에 재질 Shader의 색상 값 설정만으로 색상을 변경이 가능하도록 제작하는 방식을 고려한다.

※ 같은 모양을 가진 문자 그림 텍스처를 단지 색상 변경만을 위해 여러 벌 메모리에 불러오는 건 효율적이지 않다.
더구나 GUI 라이브러리로 사용하는 NGUI 플러그 인은 이미 모든 GUI 에 대해 색조(tint)를 적용하는 재질 Shader 를 공통적으로 사용하고 있다. 가급적 단순한 색상 변경은 Shader 의 내부 기능을 활용하도록 한다.

- **C-4-2-3.** 게임 플레이 스테이지에서는 동적 폰트(Dynamic Font) 데이터를 이용해 문자열을 그리는 횟수를 가급적 줄인다.

※ 정적 폰트(Static Font)는 한 장, 혹은 여러 장의 큰 텍스처 아틀라스에 주어진 글자의 유니코드 번호에 맞춰서 모든 문자들의 형태(Glyph)를 늘어놓고, 해당 문자가 존재하는 특정 영역을 가져오는 형태이다.
동적 폰트(Dynamic Font)는 *.ttf (True Type Font) 등의 폰트 파일을 이용해서, 실행 시간에 문자 형태(Glyph)를 그때그때 생성해서 그리는 방식이다.
정적 폰트는 미리 그려진 텍스처의 UV 좌표만 가져오기 때문에, 직접 모양을 만드는 동적 폰트 방식에 비해 메모리 사용량은 높지만, 수행 성능은 탁월하다. 동적 폰트는 반대로 메모리 사용량은 매우 적지만, 수행 성능을 정적 폰트 방식에 비해 크게 소모한다.
게임 플레이가 이루어지는 도중에는 수행 성능이 핵심적으로 중요한 요소이기 때문에, 동적 폰트로 문자열을 그리는 경우를 가급적 줄이는 것이 좋다.

◆ C-4-3. 텍스처 리소스의 처리

- **C-4-3-1.** 전체 화면에 대한 배경 GUI를 제외하고, 그 외 배경 목적으로 사용하는 GUI들은 반드시 단색이거나, 혹은 확대 / 축소할 때 반복 패턴이 적용 가능한(타일링Tiling이 가능한) 텍스처를 이용해야 한다.

※ 다양한 크기의 팝업 창, 또는 배경 창에 사용하는 텍스처는 실행 메모리를 크게 차지하는 중요한 부분이다.
팝업 창의 배경 GUI 따위에 사용하는 텍스처들을 예쁘게 만들기 위해서 고정된 픽셀 크기에 최적화시켜서 만들어버리게 되면, 그런 GUI 가 많으면 많아질수록 실행 메모리가 크게 늘어날 수 밖에 없으므로, 사용성과 시장 확대에 악영향을 준다.

- **C-4-3-2.** **자주 사용할 것이 확실한 텍스처들만 텍스처 아틀라스로 묶어야 한다.**
: 언제나 필요한 텍스처가 아니거나, 향후 컨텐츠의 확장에 따라 양이 급속히 불어날 가능성이 있는 텍스처들은 텍스처 아틀라스로 묶기 부적절한 게임 자신이다.
대표적으로, 아이템 / 스킬의 아이콘 텍스처가 있다.

※ 텍스처 아틀라스는 같은 재질을 사용하는 텍스처들을 매우 효율적으로 관리하면서 빠른 속도로 이용할 수 있게 하지만, 그 대가로 일정량의 메모리 사용량을 점유한다. 이는 텍스처 아틀라스의 일부 텍스처만 사용한다고 할지라도 마찬가지다. 일부의 텍스처를 사용하더라도, 텍스처 아틀라스 자체는 항상 메모리에 불러오는 상태여야 하기 때문이다.

아이템 / 스킬 등의 아이콘들은 양이 많을 뿐 아니라, 어떤 사용자가 어떤 아이콘 텍스처를 필요로 할지 모른다는 특징이 있다. 결국, 이런 텍스처들을 텍스처 아틀라스로 묶게 되면, 모든 경우의 수를 생각해야 하기 때문에 아이콘 텍스처의 모든 텍스처 아틀라스를 메모리에 불러와야 할 수 있다.
이런 결과는 매우 비효율적일 뿐만 아니라, 구형 기기의 경우에는 가용 메모리의 제약 때문에 불가능할 수도 있다. 이럴 때는 차라리 각 텍스처가 1 개의 아이콘 그림을 나타내는 방식으로 잘게 쪼개놓고, 필요한 아이콘의 텍스처들을 각각 불러들이는 게 낫다.

- **C-4-3-3.** 같은 그림인데 색상만 달라야 하는 경우에는, 각 색상마다 텍스처를 만들지 말고, 기준이 되는 텍스처에 재질 Shader의 색상 값 설정만으로 색상을 변경이 가능하도록 제작하는 방식을 고려한다.
: 문자를 그림으로 표현하는 경우와 마찬가지의 이유다.

◆ C-4-4. Modal 기능 GUI

- **C-4-4-1.** Modal 기능을 가진 창들은 중첩해서 여러 단계로 띄우는 것이 불가능하다.
: 여러 개의 Modal 윈도우는 순차적으로 띄워줘야 한다.

※ 여러 개의 Modal 창을 우선 순위에 맞춰 높이를 조정해서 띄워야 하기 때문에, 구현 난이도가 대폭 상승한다.
그리고 순차적인 방식에 비해 용량이나 속도 면에서도 효율적이지도 않다. 유일한 장점이라면, 각 Modal GUI들의 모양을 다르게 가져갈 수 있고, 이를 겹쳐서 보여줄 수 있는 점이다.

- **C-4-4-2.** 단, Modal 기능을 가진 창들은 우선권 등급에 의해 중간에 '끼어들 수' 있다.

※ 시스템 종료 창과 시스템 오류 창 등은 응용 프로그램의 기본적인 시스템 실행 문제를 처리해야 하기 때문에, 매우 높은 우선권을 가져야 한다.
따라서, 이런 Modal GUI들은 반드시 다른 Modal GUI보다 무조건 우선해서 처리할 수 있게 해야 한다.

- **C-4-4-3.** Modal GUI가 띄워진 경우, 스테이지의 게임 플레이 진행은 Modal GUI가 사라질 때까지 일시적으로 멈춘다.
: Modal GUI가 사라지면, 일시 정지한 게임 플레이 진행을 재개한다.

※ 다른 클라이언트와의 게임 플레이 진행 동기화가 필요한 경우에는 Modal GUI가 띄워졌다고 해서 진행을 멈추게 할 수는 없다.
하지만 이 게임은 다른 클라이언트와 게임 플레이 진행 상황을 동기화할 필요가 없기 때문에, 게임 진행이 일시적으로 멈추더라도 괜찮다. 그리고 이렇게 하는 편이 구현할 때 부작용을 덜 만드는 방법이다.

◆ C-4-5. Modaless 기능 GUI

- **C-4-5-1.** 해당 사항 없음

※ 앞으로도 필요하지 않을 것으로 예상하는 기능이다.
대부분의 모바일 기기의 특성상, 화면 표시 영역이 작기 때문에, 팝업 창이 띄워진 상태에서 배경 GUI를 조작해야 할 경우는 없을 것이다.

C-5. 게임 자산의 식별 체계

◆ C-5-1. 식별 체계 조망

- **C-5-1-1.** 식별 체계는 어떠한 개체의 종류(Kind)를 식별하는 체계와, 개별적으로 생성된 개체 인스턴스들의 고유함(Unique)을 식별하는 체계가 구분되어야 한다.

- **C-5-1-2.** **종류(Kind)를 식별하는 체계**와 **고유함(Unique)을 식별하는 식별 체계**는 혼용해서 쓸 수 없다.

- **C-5-1-3.** 종류 식별 체계에서의 값 부여는 프로젝트 전체에서 유일한 방식이어야 한다.
  : 종류 식별 체계가 여러 개인 경우에는 다른 종류로 구분해야 하는데 같은 숫자 값을 가질 수 있는 모순이 생긴다.

- **C-5-1-4.** 고유함을 식별하는 식별 체계는, 종류를 식별하는 식별 체계가 다른 경우에, 별도의 숫자 부여 체계를 쓸 수 있다.
  : 이를 테면, 같은 100의 값을 가지는 고유 번호라도, 그들의 종류를 분류하는 식별 번호가 하나는 10이고 하나는 30이라면, 100의 고유 번호는 둘 다 유효하게 쓸 수 있다는 뜻이다.

※ 실질적으로, 서로 다른 종류 식별 체계의 고유 번호들이 같은 고유 번호 체계 내에서 섞일 일이 있을 수가 없기 때문이다.
이는 마치 몬스터 분류와 무기 분류의 각 인스턴스들을 같은 ID 체계로 묶으려고 시도하는 것과 같은 이야기이다. 그러니, 다른 종류 구별 체계의 ID들끼리 겹치지 않게 하려는 시도는 전혀 쓸 데 없는 시도이다.

◆ C-5-2. 설계 원칙

- **C-5-2-1.** **타입 코드(Type Code)**
  : 종류를 구분하는 성격의 번호는 타입 코드(Type Code), 혹은 줄여서 코드(Code)로 명명한다.
  즉, 여러 개의 인스턴스들도 종류만 같다면 타입 코드는 일치한다.

- **C-5-2-2.** **고유 식별 번호(Identifier, Identifier Number)**
  : 객체마다 유일함을 보장하고 이를 식별하기 위해 붙이는 번호는 식별 번호(Identifier, 혹은

Identifier Number)라는 이름으로 명명한다.
객체 인스턴스마다 하나씩만 붙어야 하며, 다른 객체 인스턴스끼리 공유할 수 없다.
즉, 같은 타입 코드를 가진 같은 종류의 객체들이 여러 개 있다면, 그 각 객체들의 식별 번호는 서로 달라야 한다.

◆ C-5-3. 범용 식별 타입(General Type Code)

- **C-5-3-1.** 게임의 모든 모듈에서 게임의 어떤 대상이나 객체의 종류를 인식할 때 공통으로 사용하는 타입 코드

- **C-5-3-2.** 최소한 32비트 이상의 정수형 또는 부호 없는 정수형으로 표현해야 한다.

- **C-5-3-3.** 개체 인스턴스의 고유 번호로 사용하거나, 혹은 섞어서 사용해서는 안 된다.
: 이 타입 코드의 성격은 어떤 개체 단위의 '종류'를 구분하기 위함이지, '개별 개체"를 구분하기 위함이 아니다.
이 부분을 혼동해서는 안 된다.

- **C-5-3-4.** 범용 식별 타입은 서버와 클라이언트가 모두 같은 체계를 사용해야 한다.
: 서버에서 사용하는 타입 코드는, 클라이언트에서도 같은 의미로 사용되어야 하고, 반대 역시 마찬가지다.

◆ C-5-4. 고유 식별 번호(Unique Identifier Number)

- **C-5-4-1.** 어떤 객체들이 게임 서비스 기간 동안, 그리고 서비스하는 클라이언트들과 전체 지역에서 유일해야 하는 경우, 고유 식별 번호를 부여해서 그 개별 객체를 구별한다.

- **C-5-4-2.** **고유 식별 번호는 반드시 서버에서만 부여하고 해제할 수 있다.**
: 클라이언트에서 이걸 써야 하는 경우는 전혀 없다.

※ 고유 식별 번호를 클라이언트에서 고유 식별 번호를 부여하는 게 말이 안 되는 이유는, 클라이언트가 다른 클라이언트와 번호가 겹치는 게 있는지 당연히 알 방법이 없기 때문이다.

- **C-5-4-3.** 최소한 64비트 이상의 정수형 또는 부호 없는 정수형으로 표현해야 한다.

※ 유일 식별 번호를 제작하는 문제는 그다지 간단한 문제는 아닐 것으로 판단하고 있다.
이유는, 아이템이란 게 한 번 생성하면 계속해서 유지하기만 하는 성격의 물건이 아니기 때문이다. 사용자들은 끊임없이 아이템을 생성하고, 되팔거나 강화 등의 방법을 통해 '없애버린다.'

이때마다 아이템을 새로 생성해야 하고, 생성한 아이템들마다 고유 번호를 붙여줘야 하는데, 이를 어떻게 해야 잘 관리할 수 있을까?

그냥 일련번호를 순차적으로 부여하는 방식도 있겠지만, 곧 얼마 지나지 않아서 버려지는 번호가 급속하게 늘어나는 부작용이 있다. ~~물론, 64비트 이상의 값이라면, 최대 1844경까지의 숫자를 활용할 수 있으므로, 순식간에 ID에 사용할 숫자를 다 소모해버리는 불상사가 발생하지는 않겠지만, 수십 만 이상의 사용자들이 수백 번 수천 번 아이템을 생성 / 삭제할 때마다 ID 숫자가 계속해서 불어나기만 한다는 건 분명히 문제가 발생할 수 있다~~. (개뿔... 64비트조차도 ID 구조를 방만하게 설정하면, 경우에 따라서는 10년 동안의 안전성을 완전히 담보할 수 없다.)

좀 더 일반적인 방법으로는 UUID(또는 GUID)를 이용하는 방식이다. 이는 ISO 국제 표준 방식이면서, 128비트 방식의 고유 번호를 생성하고 저장하므로 숫자 범위가 엄청나게 크고, '거의' 겹칠 가능성이 없다. 문제는 '거의'가 '100% 확실함'을 보장하지는 않는다는 점이다. (실질적인 고유함Practically Unique이라고 표현하더라.)
(UUID는 Hash 알고리즘을 쓰는데, 이게 겹침을 100% 차단하는 게 아니다.)
그리고 128비트 자료형은 좀 더 처리가 느리고, 저장 공간도 많이 먹는다는 점도 고려해야 한다. (특히, 대량의 데이터를 보관해야 하는 데이터베이스의 경우에는 이 체감이 훨씬 더 크게 다가온다.)

- **C-5-4-4.** 범용 식별 타입(General Type Code)이 같은 객체들일지라도, 그 객체들이 각자 독립적으로 존재해야 하는 경우에는 서로의 고유 식별 번호는 다르다.
: 만약 '블루', '체리', '주시'라는 이름을 가진 세 마리 고양이가 존재한다고 치자.
이 때, 범용 식별 타입은 '고양이'가 되고, '블루', '체리', '주시'는 각각 구분되는 고유 식별 번호를 가지는 셈이다.

◆ C-5-5. 객체 식별 번호(Instance Identifier)

- **C-5-5-1.** 클라이언트에서 객체 인스턴스들을 개별적으로 식별해야 할 때 사용한다.

※ 주로, 객체 참조끼리 정말 같은 객체를 가리키고 있는 건지 알아보고 싶을 때 사용한다.
참조는 C / C++ 식으로 말하자면 메모리의 포인터 주소 혹은 참조자(이건 C++의 경우만)를 말하는데, 포인터나 참조자를 비교하는 경우, 원래 의도와는 다르게 메모리 상에서의 주소 값을 비교하는 경우가 발생할 수 있다.
이럴 때는 객체에게 고유하게 부여한 식별 번호가 같은지를 비교하는 게 더 안전하다.

- **C-5-5-2.** 하나의 클라이언트 내에서만 유일함을 유지해도 된다.
: 다른 클라이언트에서 똑 같은 번호가 나와도 그 클라이언트 내 다른 개체가 중복해서 사용하지만 않으면 상관없다.

- **C-5-5-3.** 최소한 32비트 이상의 정수형 또는 부호 없는 정수형으로 표현해야 한다.

※ 클라이언트의 경우에는 대부분의 상황에서 32 비트 정수형 범위(약 21.47 억, 최대 약 42 억)를 넘어설 정도로 많은 인스턴스 ID 를 부여해야만 할 경우가 거의 없다.
또한, 하나의 프로세스 안에서의 인스턴스들만 신경 쓰면 되기 때문에, 사용하지 않는 ID 를 재활용하게 만들기도 아주 쉽다.

- **C-5-5-4.** 절대로 고유 식별 번호와 혼용해서 사용하면 안 된다.
: 고유 식별 번호는 게임 서비스에 접속하는 각 클라이언트들끼리도 겹치는 값이 있어서는 안 되는 값이다. 그러므로 반드시 발급 권한이 서버에만 존재해야 한다.
반면, 객체 식별 번호는 개별 클라이언트 영역에서만 고유함을 유지하면 되기 때문에, 모든 취급 권한도 클라이언트에게 있다.

◆ C-5-6. 고유 번호의 발급 방식

- **C-5-6-1.** 고유 번호의 자료형은 64비트 부호 없는 정수형을 사용한다.

- **C-5-6-2.** 고유 번호의 **발급은 서버에서만 담당**한다.
: 특히, 데이터베이스에 저장할 필요가 있는 고유 번호들은 원칙적으로 전부 서버에서 생성 및 부여 과정을 담당해야 한다.

- **C-5-6-3.** 클라이언트에서도 인스턴스 ID와 같은 고유 번호 발급 기능이 있지만, 이는 어디까지나 클라이언트의 응용 프로그램을 실행하는 동안에만 유효하고, 장치 독립적으로 고유함을 보장하지도 않는다.

- **C-5-6-4.** 다른 목적으로 사용하는 고유 번호끼리 호환하면 안 된다.
: 즉, 다른 타입 코드 분류에 속하는 고유 번호들끼리는 같은 체계로 사용할 수 없다.

※ 사용자 계정마다 부여한 고유 번호를 아이템 인스턴스의 식별 번호를 담는 매개 변수에 넣는다거나 하는 일이 발생하면 안 된다.
다만, 이런 경우에는 사용처는 달라도 자료형 자체는 같은 것을 쓰는 경우가 많기 때문에 (결국 int 아니면 long 으로 쓸 것 아닌가), 컴파일러에게 구별을 맡기기는 어렵다. 구현 담당자가 혼동하지 않도록 적절하게 타입 별명을 설정하든가, 변수에 사용처에 대해 설명을 할 수 있는 접두어를 붙이는 등의 장치가 필요하다.

- **C-5-6-5.** 고유 번호의 발급 장치나 프로그램을 껐다가 다시 시작한다고 하더라도, 이전에 발급했던 번호를 중복해서 발급하지 않게 해야 한다.

- **C-5-6-6.** 서비스 전체에서 고유함을 보장하는 고유 번호를 생성하는 방식을 다음과 같이 설계한다.

1. 고유 번호로 쓸 64비트 정수형의 자료형을 상위 32비트, 하위 32비트의 자료형으로 나눈다.

2. 상위 32비트는 고유 번호를 생성할 수 있는 권한을 가진 각 ID 팩토리들을 구분하는 번호로 사용한다.
 : 이걸 이제부터 상위코드라고 하자.

3. 하위 32비트는 고유 번호를 생성할 수 있는 권한을 가진 각 ID 팩토리에서 아이템에 순차적으로 부여하는 ID들을 말한다.
: 이걸 이제부터 하위코드라고 하자.

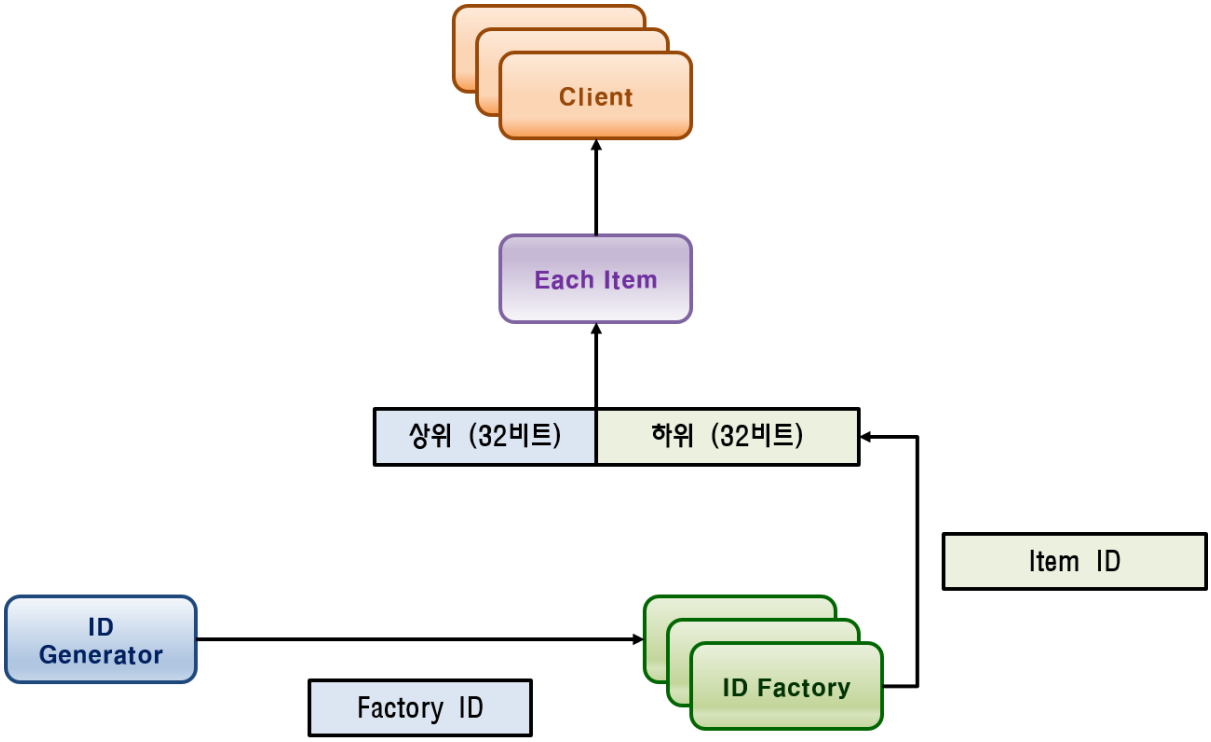
4. 고유 번호는 상위코드와 하위코드의 조합으로 이루어진다.

5. 하위코드들은 분리되어 있는 각 ID 팩토리가 담당한다.
 각 ID 팩토리들은 하위코드들이 겹치지 않게 발급을 하지만, 각 ID 팩토리들은 서로의 하위코드 발급 상황을 알 수 없다. (알려고 하지도 않고, 알 필요도 없다.)
 그 때문에, 각 ID 팩토리에서 생성한 번호가 우연히 서로 겹칠 수 있다.

6. 그렇지만, 각 ID 팩토리에 할당되어 있는 상위코드들은 서로 엄격하게 분리되어 있다.
 : ID 팩토리에 할당하는 상위코드 번호는 프로젝트의 전체 지역 서비스를 통틀어 단 한 군데에서만 발급한다.

7. 전역적으로 보자면, ID 팩토리들이 현재 이용할 수 있는 상위코드들만큼은 유일하도록 분류하기 때문에, 결과적으로 만들어지는 상 / 하위 코드의 번호가 합쳐진 ID 번호는 전역적으로 유일함을 보장할 수 있다.

8. 각 ID 팩토리들은 다음과 같은 상황에서, 전역 상위코드 발급기에게 새로운 상위코드를 요청해야 한다.
 - 현재 상위코드에서는 모든 하위코드 목록이 다 차서, 더 이상 ID를 발급할 수 없다.
 ~~- ID 팩토리가 꺼진 후에 재시작되었다.~~(ID 팩토리의 각 ID 발급 상황은 데이터베이스로 저장해도 되기 때문에 굳이 ID 팩토리의 재부팅 여부를 조건에 포함할 이유가 없다.)
 - 어떤 이유든 간에, 현재 상위코드 내에서, 다음에 발급해야 할 하위코드의 순번을 알 수 없게 되었다.



< 고유 ID 생성이 진행되는 과정 >

※ 위 설계 방식은 서비스 방식이 아주 극단적으로 고유번호의 유일성을 보장해야 하는 경우를 대비한 것이므로, 상황에 따라 훨씬 단순하게 적용할 수 있다.
그리고 그러한 상황은 **서비스 방식에 따라 얼마든지 달라질 수 있다.**
그러니까, 만약 게임 서비스 방식이 여러 개의 '서버'로 이루어져 있고, 각 서버는 단일한 머신이고, 제한된 인원까지만 받아들이며, (대략 활동인원 몇 천명 정도로) 서버 간 캐릭터 이동이 절대로 불가능하다면, 이렇게 엄밀한 ID 발급 방식은 과도한 아키텍처에 불과하다.
그냥 각 서버마다 자동으로 ID를 1씩 증가하는 할당 방식을 써도 ID 고갈, ID 충돌 문제없이 잘 써먹을 수 있다.
다만, 위 생성 알고리즘은 하나의 '서버'가 물리적으로 분산이 되어 있거나, 서버 간 캐릭터의 이전, 또는 서버 자체의 통 / 폐합 상황에서도 문제없이 병합이 가능하다는 있는 장점이 있다.

- **C-5-6-7.** 고유 번호 그 자체에는 고유 번호를 생성한 시간과 관련한 정보를 반영하지 않는다.

※ 64비트 자료형 그 자체는 시간이든 고유 ID든 둘 중 한 가지만 표현하기에는 꽤 충분한 만큼의 공간을 제공한다.
그렇지만 시간과 고유 ID를 한꺼번에 다 효과적으로 표현하기에는 조금 부족하다. 상 / 하위로 32비트씩 쪼개더라도, 32비트로 시간을 세면 두 달을 넘길 수 없고, 아이템의 고유 번호로 쓰기에도 공간이 장기적으로 봤을 때 충분하다고 볼 수 없다.
만약 시간 정보를 넣는다 해도, 정밀도는 크게 떨어질 수 밖에 없다. (10분, 20분 단위라든가...)
그래서 개인적으로는 고유 번호 자체에 생성 시간 정보를 넣는 건 좀 별로라고 보고 있고, 별도로 시간 정보를 저장하는 게 더 좋다고 본다. DB 입장에서는 스토리지 부담이 더 늘어나는

점은 있지만, 어차피 정밀하지도 않은 생성 시간은 보관해도 가치가 없기 때문에, 제대로 된 날짜 자료형을 따로 저장해서 쓰는 게 더 낫다.

◆ C-5-7. 사용자 번호

- **C-5-7-1.** 각 사용자 계정은 그 사용자 계정을 나타내는 고유한 번호를 발급받아야 한다.

- **C-5-7-2.** 사용자의 계정이 존재하는 한, 그 사용자 계정의 번호는 바뀔 수 없다.
  : 물론, 세상 일이 100%인 건 없겠지만...

※ 사용자 번호는 우리나라의 주민등록번호나, 미국의 사회보장번호(Social Security Number)와 유사한 기능을 한다.

- **C-5-7-3.** 여러 개의 사용자 번호가 하나의 계정을 가리키거나, 혹은 그 반대 상황이 될 수 없어야 한다.
  : 하나의 사용자 계정은 반드시 1개의 사용자 번호만 가진다.

- **C-5-7-4.** **사용자 번호는 절대로 클라이언트에게 노출하지 않는다.**
  : 대신, 클라이언트에게는 인증 대용으로 쓸 수 있는 임시 번호를 발급한다.

- **C-5-7-5.** 임시 번호는 만료 조건이 있어야 한다.
  : 만료 기간을 따로 줘도 되겠지만... 만료 기간 설정이 애매할 수도 있기 때문에, 로그아웃이나 접속 종료가 될 때까지 써먹을 수 있는 조건이어도 된다.

◆ C-5-8. 캐릭터 고유 번호

- **C-5-8-1.** 사용자 계정의 캐릭터들마다 고유한 번호가 있어야 한다.

- **C-5-8-2.** 사용자 번호와 캐릭터 고유 번호는 혼용할 수 없다.
  : 사용자 번호는 로그인하는 사용자의 계정에 대한 번호이고, 캐릭터 고유 번호는 그 사용자 계정에 속하는 각각의 캐릭터들을 구분하기 위한 번호이다.

※ 다만, ID 를 생성하는 방식을 '연관성 있게' 만들 수는 있을 것이다.
이를 테면, 사용자 번호가 10000 이라고 가정해보자.
그리고 하나의 계정에는 최대 8 개의 캐릭터를 가질 수 있다고 하면, 각 캐릭터들을 10001 ~ 10008 까지 캐릭터 ID 를 부여하고, 사용자 번호는 반드시 10 단위로만 증가시기는 방식이다.
즉, 다음 계정은 10010 번이고, 캐릭터 ID 는 10011 ~ 10018 까지라는 뜻이다.

물론, 그렇다고 해서 10001 ~ 10008 의 캐릭터 번호가 사용자 번호 10000 과 사용처나 목적이 같아지는 건 아니다. 다만, 캐릭터 번호를 보고서도 어느 계정의 캐릭터인지 알 수 있게 될 뿐이다.

- **C-5-8-3.** 플레이어가 조종하는 캐릭터들에게만 고유 번호를 부여한다.
: 이 게임은 동기화해야 하는 비 플레이어 캐릭터(NPC)들이 없기 때문에, 캐릭터의 고유 번호는 플레이어 계정의 캐릭터들만 부여해도 된다.

◆ C-5-9. 아이템 고유 번호의 발급 방식

~~- **C-5-9-1.** **모든 아이템에 고유 번호를 발급할 수는 없을 것 같다.**~~
~~: 뭔가 획기적인 고유 번호 부여 방식이 떠오르지 않는다면, 중요도가 높은 아이템에만 고유 번호를 발급해야 할 것으로 판단한다.~~

~~※ 현재 생각하는 게임의 대략적인 디자인으로 미루어 짐작해보자면, 소모품 및 낮은 등급의 아이템들은 수없이 생성되고 파괴되기를 반복할 것이다.~~
~~낮은 등급의 재료 아이템들, 낮은 등급의 아이템들은 어떤 스테이지의 보상에서도 계속해서 등장하고, 대부분은 채 한 번도 장착되지 않고 다른 더 중요한 아이템을 위한 재료나 강화 점수를 위해 소모될 것이다.~~
~~이 중에는 장착할 수 있는 성격의 아이템도 존재하지만, 실질적으로는 소모성 아이템에 더 가까운 생애주기를 가지고 있다고 할 수 있다.~~
~~이런 아이템들까지 모두 고유 번호를 발급하고 관리하는 것은 아주 비효율적일 뿐만 아니라, 그렇게까지 해야 할 가치가 거의 없다.~~

~~- **C-5-9-2.** 현재로써는. **전설 등급 장비 아이템에 대해서만 고유 번호를 발급**하는 것으로 한다.~~
~~: 중요도에 대한 판단은, 역시 아이템의 등급이 될 것이다. 희귀한 등급일수록 중요한 아이템일 가능성이 높고, 그런 아이템들은 고유 번호를 발급해서 관리할만한 가치와 이유가 있다.~~

- **C-5-9-3.** 모든 사용자의 모든 보유 아이템에 고유한 번호를 발급한다.

- **C-5-9-4.** 한 번 아이템에 발급한 번호들은 다시 사용하지 않는다.
: 아이템 데이터를 삭제한다고 할지라도 이는 마찬가지다.

※ 이렇게 해도 상관없겠다고 생각한 이유는, **아이템 고유 번호는 그 숫자의 각 자릿수, 또는 비트 자릿수에 대해 어떤 의미도 부여지 않았기 때문**이다.
64 비트 정수형은 대략 -922 경 ~ +922 경, 부호 없는 정수형인 경우에는 대략 1844 경의 숫자를 구분할 수 있다. 이는 일반적으로는 어마어마하게 큰 숫자이다.

하지만 이 정도의 숫자일지라도, 각 자릿수마다 구분을 두거나, 숫자 값의 비트 자릿수마다 어떤 의미 구분을 두고 사용하면 얘기가 좀 달라져서, 경우에 따라서는 숫자 부족에 시달릴 수가 있다.

무슨 말이냐 하면, 다음과 같은 예를 들어보면 이해가 갈 것이다.

아이템의 고유 번호를 설계할 때, '십만 ~ 만 자리 단위까지는 아이템의 장착 부위를 구분하는 용도로 쓰고, 그 이하는 개별 아이템 종류를 구분하는 용도로 쓴다' 같은 규칙을 만든다고 하자.

이렇게 2, 3 개의 자릿수 단위씩 끊어서 하위 의미를 부여해서 구분해서 쓰는 방식이 되면, **실제로 쓰는 숫자의 개수는 의외로 숫자의 전체 범위에 비해 얼마 되지 않는 경우가 많다.**

위의 예를 그대로 사용하면, 각 장착 부위마다 아이템에 부여할 수 있는 실제 ID 의 개수는 겨우 1 만개에 불과하다.

이런 식이라면 장기간 서비스에 할당할만한 숫자의 개수가 나오지 않게 되는 경우가 종종 생긴다. 특히, 일방 통행으로 숫자가 무조건 증가해야 하고, 더 이상 사용하지 않는 ID 를 재활용할 수 없는 경우에는 더욱 그렇다.

그러나 그런 구분 요소가 없이 **단순하게 1 씩 증가하는 방식으로만 쓴다면, 64 비트 정수는 웬만한 상황의 카운트 속도로는 결코 다 소모하기 어려운 엄청나게 큰 수**다.

게임 서비스 상황을 대략적으로 예상해서 소모 정도를 추정해보면 다음과 같은 결론을 내릴 수 있다.

1) 한 사람이 평균 하루 10 게임씩 한다고 가정한다.
2) 1 게임을 할 때마다 대략 10 개의 아이템 번호의 발급이 필요하다고 가정한다.
: 그러면 하루에 평균적으로 100 개 정도의 아이템 번호를 한 사람이 발급받는 셈이다.

위 가정을 토대로 계산을 해보면, 1 년간 한 명의 게이머는 평균 365 × 100 = 36,500 개의 ID 를 발급받는다. (사실 1 년 내내 결코 쉬는 날 없이 게임을 해야만 가능한 숫자이므로, 상당히 하드코어 한 가정이라고 볼 수 있다.)

이런 사람이 1 백만 명이 있다고 가정해도, 1 년 간 약 365 억개 의 아이템 ID 가 발급되는 정도다. (심지어 버려진 ID 는 전혀 재활용하지도 않았다.)

이 정도라면 10 년간 서비스를 한다 해도(모바일 게임 업계의 상황을 본다면 10 년 간 1 백만명의 고정 이용자가 있는 게임 서비스라는 건 거의 비현실적인 상황이다.), 3,650 억개 정도의 아이템 고유 ID 가 발급될 뿐이다.

따라서, 64 비트 부호 없는 정수형의 한계인 1844 경의 범위는 차고도 넘치는 범위가 된다.

위 조건에서 한 사람당 평균 게임 수가 5 배로 늘더라도(매일매일 하루에 50 게임!), 발급하는 ID 의 개수는 약 1 조 8 천억개 정도다. 평균 게임 수가 10 배 이상이라면 약 3.65 조가 되더라도 숫자에는 아직 여유가 넘친다.

아주 극단적으로, 1 인당 하루 평균 1,000 게임을 10 억명의 사용자가 플레이 하더라도 10 년간 필요한 ID 의 개수는 3,650 조개 가량으로, 1 경개에도 미치지 못한다. ~~(과연 돈은 얼마나 벌까?)~~

심지어, 이 기준은 국가와 지역을 초월한 '전체 서비스'의 기준이다. 서비스 지역, 플레이 서버 구분 등 각종 격리 조건 등이 더해지면 한계를 우려하는 게 의미가 없을 정도가 된다.

즉, 64 비트 정수형은 각 자릿수나 비트 자리마다 추가적인 의미를 부여해서 쓰려고 하지 않는 한, 단순 카운트만으로 다 소모하기는 어려운 충분히 엄청나게 큰 숫자라는 점이다.

- **C-5-9-5.** 아이템의 고유 번호는 해당 아이템이 아이템 슬롯 공간을 차지하는지의 여부에 의해 결정한다.
: 완전히 같은 종류와 능력을 가진 아이템이라도, 다른 아이템 저장 슬롯에 배정해야 하는 아이템이라면, 고유 번호를 별도로 발급한다. 즉, 별도의 아이템 인스턴스로 간주하는 셈이다.

※ 개념상 이해하기에 약간 어색한 면이 있을 것이다.
왜냐하면 플레이어 A 의 체력 회복 물약과 플레이어 B 의 체력 회복 물약 아이템은 완전히 같은 아이템이지만, 고유 번호는 다르기 때문이다.
심지어 이 개념 아래에서는, 플레이어가 슬롯 1 개에 10 개씩 적재 가능한 체력 회복 물약을 23 개 들고 있다고 하면, 10 + 10 + 3 개로 나눠서 아이템 슬롯에 적재가 되고, 각 슬롯의 체력 회복 물약의 고유 번호가 전부 달라져야 한다!

C-6. 네트워크 / 서버

◆ C-6-1. 구성 플랫폼

- C-6-1-1. 운영체제
: Microsoft Windows Server를 사용한다.
(현재 Windows Server 2012 R2가 기준)

- C-6-1-2. 웹 서버 플랫폼
: Microsoft Internet Information Service(IIS)를 사용한다.
(현재 IIS 8.0이 기준)

- C-6-1-3. 웹 서버 구현
: C#언어와 ASP.NET을 사용한다.
(현재 .NET Framework는 4.5.2 기준)

- C-6-1-4. 클라우드 플랫폼
: Amazon Web Service(AWS) 또는 Microsoft Azure를 사용한다.
(아직 결정하지 않은 상태이나, 현재로써는 AWS가 좀 더 유력하다.)

※ 솔직히 말하자면, 이 구성은 가장 많은 라이선스 비용을 요구한다.
하지만 기술적으로 가장 안정적이면서, 개발자 풀을 유지하는 데 가장 문제 적은 구성이다.
(ASP.NET 은 여기에서 예외일 수 있겠지만...) 기술지원 역시 가장 안정적이고 확실하다.
참고로, 라이선스 비용 상으로 저렴한 구성으로 고려한 사항은 다음과 같다.

- 운영체제 : CentOS
- 웹 서버 플랫폼 : Apache Tomcat
- 웹 서버 구현 : Java 와 JSP
- 클라우드 플랫폼 : 위와 동일

위 조합은 특히 중국 쪽에 진출할 생각이 있다면 진지하게 고려해봐야 한다. 그쪽 동네에서 좋아한다고 함(...)

※ 사실 Windows Server 를 고집하는 이유는, 서버를 ASP.NET 으로 제작할 경우에 IIS 와의 조합이 좋기 때문이다.

IIS 는 Windows 플랫폼에서만 돌릴 수 있으며, IIS + ASP.NET 의 조합은 빠르고 강력하기로 유닉스 / 리눅스 개발자들도 인정하는 부분이라서...
다만 리눅스에서 아예 ASP.NET 을 돌릴 방법이 없는 건 아니다.
Mono 라는 오픈소스 .NET 프로젝트가 있으며, 이를 활용해서 리눅스에서도 ASP.NET 서비스가 불가능한 건 아니라고 한다. 이에 대해서는 추가적인 조사와 연구가 필요하다. 이게 가능하다면, 소프트웨어 벤더에 대한 라이선스 비용을 최소화하면서 기존 서버 구성을 구동할 수도 있을 것이다.
~~(개인적으로는 .NET Framework 의 리눅스 진출을 기대하고 있다.)~~

◆ C-6-2. 서버 구조

- **C-6-2-1.** 웹 페이지 기반의 요청(Request) – 응답(Response) 방식으로 동작하는 웹 서버로 제작한다.
: 이 방식은 클라이언트에서 이벤트 처리 요청이 있을 때, 서버는 그 요청에 대해 처리하고 즉시 요청자에게 응답을 주는 방식이다. 이 방식에서는 서버가 먼저 클라이언트에게 이벤트를 통지할 수 없다.

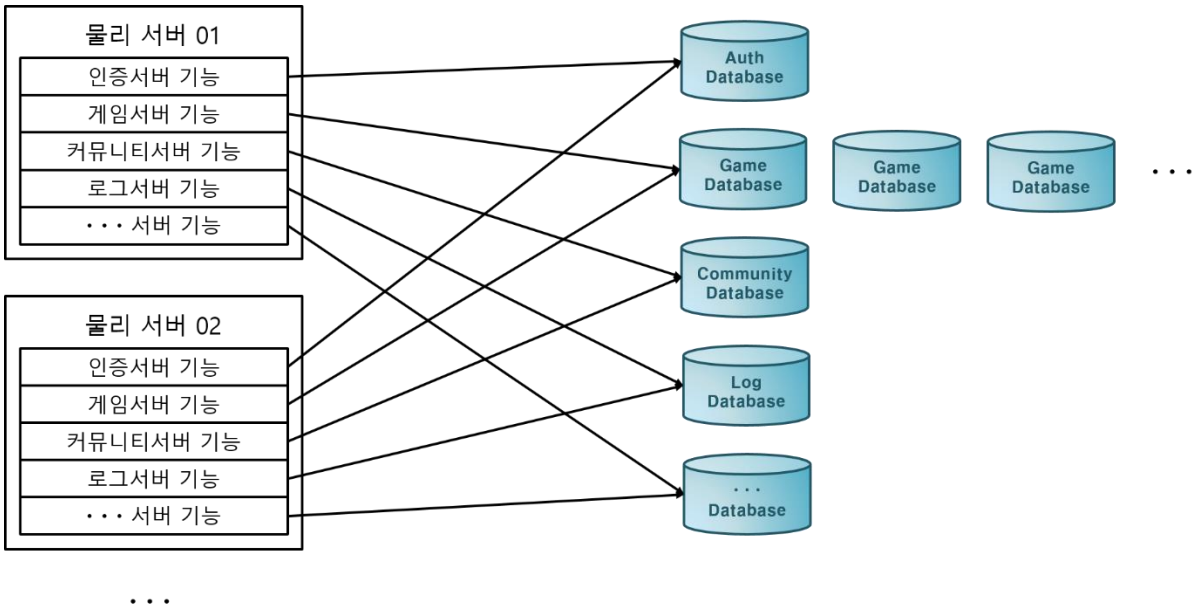
<네트워크 이벤트의 처리 개념>

※ 웹 서버는 상태가 없는(Stateless) 서버이기 때문에, 클라이언트로부터 요청이 있기 전에는 먼저 서버가 특정 클라이언트에게 이벤트 통지를 할 수 없다.

- **C-6-2-2.** 하드웨어 부문 분산 방식
: 각 서버의 기능 모듈은 물리적으로는 하나의 하드웨어에 탑재한다. 즉, 하드웨어 적으로는 서버의 기능을 분산하지 않는다.
하드웨어 입장에서의 서버 분산은, 이렇게 **모든 기능 모듈을 가진 서버를 여러 대 두는 방식**이다.

※ 웹 서버 방식에서는 게임 내부 상태를 서버가 직접 유지 관리하지를 않기 때문에, 잦은 연결(Connection) 유지를 제외하면 서버의 처리 성능 그 자체 때문에 분산을 해야 할 일은 별로 없다.
심지어, 클라이언트의 각 요청마다 요청자 식별 때문에 인증 과정을 거쳐야 하는데, 서버 모듈이 죄다 물리적으로 분산되어 있으면 서버끼리의 네트워크 통신이 증가해서 이게 오히려 성능 병목이 될 수도 있다.

서버들마다 게임 내에서 고정적으로 담당해야 하는 구역이 있고, 그 때문에 서버 자체의 처리 성능에 민감한 MMORPG 소켓 서버들의 경우에는 서버 간 통신 부하를 감당해야 할만한 이유가 있다. 그러나, 웹 서버 방식은 고정적인 담당 구역도 없고, 굳이 그래야 할 필요가 없다.



<물리적으로 서버를 분산할 때, 그 의미는 대략 이와 같다.>

- **C-6-2-3.** 프로세스 부문 분산 방식

: 각 기능별 서버 모듈들은 별도의 프로세스에서 구분해서 동작하도록 한다. (이는 하드웨어 상의 구분이 아니다. 하나의 하드웨어에서도 프로세스 단위로 구분해야 한다는 의미이다.)

프로세스	설명
인증 서버	• 사용자 인증 처리만 담당한다. • 사용자가 어느 게임 서버에 붙어야 하는지를 지정하는 역할을 한다. • 인증 서버 자체는 하드웨어 적으로 분산될 수도 있지만, 인증 데이터베이스는 서비스 단위 내에서 단 1 곳만 존재한다.
게임 서버	• 게임 플레이에 대해, 서버 측에서 필요한 논리 처리를 담당한다. • 실질적인 각 사용자의 게임 정보들은 게임 서버에서 보유하고 있다. • 서비스 단위 내에서 가장 숫자가 많을 것이다.
결제 서버	• 사용자가 결제한 내용을 검증하고, 결제 결과를 인증하고 구매를 반영해주는 기능을 담당한다. • 서비스 단위 내에서 소수만 존재한다. : 다수가 되어야 한다면? ~~(지나친 결제는 감사합니다.)~~
커뮤니티 서버 (파티 서버)	• 사용자들끼리의 채팅 내용을 중계한다. • 사용자들 간의 친구 관계와 길드 관련 요청사항들을 처리한다. • 서비스 단위 내에서 1 ~ 소수만 존재한다..

대전 서버	• 사용자 캐릭터들 간의 전투, 길드 간의 대전 컨텐츠를 담당한다. • 수준에 맞는 사용자들, 길드들을 매치하고, 대전 결과를 처리한다. • 서비스 단위 내에서 소수가 존재한다.

※ 모든 서버 기능 모듈들은 하나의 하드웨어에 탑재하지만, 인증 서버 프로세스 따로, 게임 서버 프로세스 따로, 결제 서버 프로세스 따로 구동한다는 의미이다.

이렇게 하는 이유는 이 편이 장애 대응에 있어 좀 더 유연하기 때문이다. 하나의 기능 모듈이 동작하지 않는데, 그게 게임 서비스 진행에 직접 영향을 끼치지 않는다면, 전체 서버를 정지하지 않고 해당 모듈만 수정하는 식으로 대응할 수도 있다.
이를테면, 채팅 서버 모듈이 고장이 났는데, 게임 서버 모듈과 별도의 프로세스로 분리되어 있다면, 채팅 서버 모듈의 프로세스는 죽어버렸지만, 게임 서버 모듈은 죽지 않고 그대로 서비스를 진행할 수가 있다. 이때는 단지, 채팅 전송 기능만 정지되어 있을 뿐이다.
사소한 기능 하나 때문에 전체 서버 기능을 쓰지 못하는 것보다는, 설계 구조가 좀 더 번거롭고 복잡하더라도 분산되어 있는 편이 장애 대응을 좀 더 '부드럽게' 할 수 있게 한다.

- **C-6-2-4.** 기능 모듈 부문의 분산 방식

모듈	설명
공통 모듈	• 게임 전체 프로젝트에서 공통으로 사용하는 객체와 타입 정의, 상수 정의, 프로토콜 정의 등을 포함한다. • 라이브러리 형태로 제작한다. • 서버와 클라이언트, 저작도구 등이 공통으로 사용한다.
서버 공통 모듈	• 서버 프로젝트에서 공통으로 사용하는 객체와 타입 정의, 상수 정의, 프로토콜 정의 등을 포함한다. • 라이브러리 형태로 제작한다. • 서버 프로젝트들만 사용한다.
서비스 공통 모듈	• 서버와 저작도구, 관리도구에서 공통으로 사용하는 객체와 타입 정의, 상수 정의, 프로토콜 정의 등을 포함한다. • 라이브러리 형태로 제작한다. • 서버와 저작도구, 관리도구프로젝트들만 사용한다.
DB 명령 모듈	• 데이터베이스에 데이터를 기록하거나 가져오게 하는 명령의 실행과 관련된 기능을 구현한다. • 라이브러리 형태로 제작한다. • 서버 프로젝트들만 사용한다.
인증 모듈	• 사용자 인증과 관련된 기능을 구현한다. • 사용자가 어느 게임 서버에 접속해야 하는지를 지정하는 기능을 구현한다. • 서버 측 전용으로 사용하는 라이브러리 형태로 제작한다.

게임 모듈	・게임 플레이에 대해, 서버 측에서 필요한 논리 처리를 담당한다. ・실질적인 각 사용자의 게임 정보들은 게임 서버에서 보유하고 있다. ・서버 측 전용으로 사용하는 라이브러리 형태로 제작한다.
결제 모듈	・사용자가 결제한 내용을 검증하고, 결제 결과를 인증하고 구매를 반영해주는 기능을 담당한다. ・서버 측 전용으로 사용하는 라이브러리 형태로 제작한다.
커뮤니티 모듈	・사용자들끼리의 채팅 내용을 중계한다. ・사용자들 간의 친구 관계와 길드 관련 요청사항들을 처리한다. ・서버 측 전용으로 사용하는 라이브러리 형태로 제작한다.
로그 모듈	・자신을 제외한 모든 서버들로부터 로그 데이터들을 수집해서 보관하는 기능을 구현한다. ・저장되어 있는 로그 정보를 적절하게 가공해서 원하는 요청 형식에 맞게 제공하는 기능을 구현한다. ・ 서버 측 전용으로 사용하는 라이브러리 형태로 제작한다.
인증 서버	・인증 서버 모듈을 서버 응용프로그램 또는 서비스로서 실행할 수 있게 한다. ・웹 서비스 응용 프로그램으로 제작한다.
게임 서버	・게임 서버 모듈을 서버 응용프로그램 또는 서비스로서 실행할 수 있게 한다. ・웹 서비스 응용 프로그램으로 제작한다.
결제 서버	・결제 서버 모듈을 서버 응용프로그램 또는 서비스로서 실행할 수 있게 한다. ・웹 서비스 응용 프로그램으로 제작한다.
커뮤니티 서버	・채팅 서버 모듈을 서버 응용프로그램 또는 서비스로서 실행할 수 있게 한다. ・웹 서비스 응용 프로그램으로 제작한다.
대전 서버	・대전 서버 모듈을 서버 응용프로그램 또는 서비스로서 실행할 수 있게 한다. ・웹 서비스 응용 프로그램으로 제작한다.

※ 크게 보면 모듈과 서버로 나눈 부분이 눈에 들어올 것이다.
이렇게 나눈 이유는, 향후 코드의 재활용에 더 용이한 구조라면 면 외에도, 테스트 과정에서 클라이언트가 서버의 역할을 맡아주도록 만들기에 더 쉬운 구조 때문인 점도 있다.

이를 테면, 스테이지를 완료한 후의 보상 처리 부분은 클라이언트 – 서버 구조를 가지고 있는 게임이라면 어떤 게임이든 서버가 전용으로 처리하며, 클라이언트는 그 코드를 볼 수조차 없다.
그러나, 테스트 과정에서는 매번 서버에 접속하기가 번거로울 때가 많고, 때로는 비즈니스 데모를 시연할 때, 서버에 접속할 수 없는 환경에서 시연을 해야 하는 경우도 왕왕 존재하는데,

이 때 로컬에서 서버 역할을 대행할 수 있는 클라이언트의 존재가 있다면 훨씬 부드럽게 일을 진행할 수 있다.

하지만 유의할 점이 있다. 비록 로컬에서 서버 역할까지 클라이언트가 담당하는 경우가 필요하다고 해서 이를 클라이언트 따로 서버 따로 개발을 해버리면 엄청난 개발력 낭비로 이어진다는 점이다. 이런 방식은 냅다 시작하기는 쉬울지 몰라도, '나중에 진짜 쓸 것'과 '당장 임시로 쓸 것'의 기능을 구분해가면서 코드를 추가해야 하고, 아키텍처를 몹시 지저분하게 만든다. 코드베이스가 커지면 통제가 불가능한 괴물로 성장해버리기 때문에 이런 상황은 절대적으로 피해야 한다.
그보다는 서버의 기능을 하는 '라이브러리 모듈'을 따로 만들고, 서버와 클라이언트가 각각 이를 이용하는 인터페이스만 별도로 제작해서 서버 모듈의 라이브러리를 링크해서 쓰는 게 효율적이고 깔끔한 개발 방법이다.

◆ C-6-3. 장애 대응 방식

- **C-6-3-1.** 예상하는 성능 병목 구간은 대략 다음과 같다.

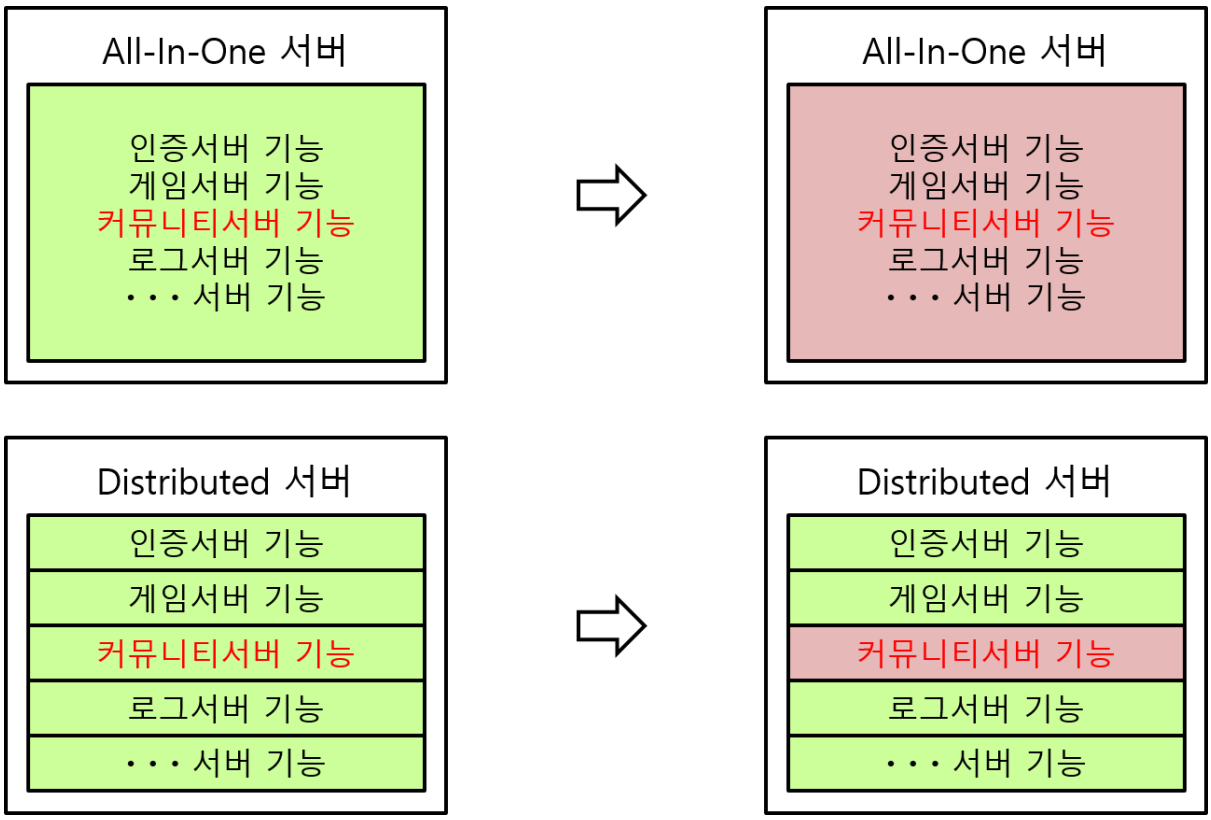
- 특정한 데이터베이스와 지나치게 연결을 자주 맺고 끊음
- 특정한 서버에 클라이언트로부터의 요청 처리가 너무 많음
- 한 번에 시간이 아주 오래 걸리는 요청에 대한 처리
- 중간에 어떤 이유로든 실패하면, 요청 처리 과정의 전체 절차가 취소되어야 하는 경우(주로 결제 데이터와 관련됨)

- **C-6-3-2.** 예상하는 네트워크 이벤트 관련 장애 방식은 대략 다음과 같다.

- 클라이언트가 서버에 처리를 요청했지만 서버가 응답하지 않음
- 클라이언트가 서버에 처리를 요청했으나, 서버가 이를 거부함
- 서버가 클라이언트에게 처리 결과를 전송했으나, 클라이언트는 이를 수신 받지 못함.

- **C-6-3-3.** 위험 분산을 위해, **주요한 서버 모듈들을 별도의 프로세스 및 웹 서비스에서 돌아가도록 분리**한다.

※ 기능 별로 별도의 서버가 프로세스를 할당 받아 돌아간다면, 일부 덜 중요한 모듈에 장애가 발생해도 핵심 서비스는 정지하지 않고 그대로 서비스를 돌리면서, 고장이 난 서버 모듈만 다시 정상적으로 재시작하게 할 수 있다.
뭐, 인증 서버나 게임 서버처럼 핵심 모듈의 장애라면 어쩔 도리가 없겠지만.... ;ㅅ;



<기능이 고장 났을 때, 분산이 되어 있는지 여부에 따라 영향이 미치는 범위가 다르다.>

◆ C-6-4. 버전 점검

- **C-6-4-1.** 서버는 게임 클라이언트의 접속을 승인하고, 게임 클라이언트가 다음 장면으로 넘어가도록 허가하기 전에, 반드시 게임 클라이언트가 서버에서 요구하는 최신 버전에 일치하는 응용 프로그램과 데이터를 가지고 있는지 점검해야 한다.

- **C-6-4-2.** 게임 클라이언트는 서버와 접속할 수 있어야만 버전 업데이트에 대한 안내를 받을 수 있다.
  : 즉, 네트워크에 연결하지 못하는 게임 클라이언트는 게임 응용 프로그램 그 자체를 실행할 수는 있지만, 게임 플레이 장면 단계로 넘어가지 못한다. 그 전에 반드시 버전 점검을 거쳐야만 하기 때문이다.

- **C-6-4-3.** 게임 클라이언트가 버전 **정보를 확인하기 위해 네트워크로 접속해야 하는 도메인의 이름은 서비스가 시작된 이후부터는 변경하거나, 혹은 접속이 아예 불가능한 상태가 되어서는 안 된다.**

※ 이 부분은 절대적으로 중요하다.

게임 클라이언트가 아직 서버와의 네트워크 연결이 수립되지 않은 상태에서, 최초에 어느 주소로 접속을 시도해야 할지에 대한 여부는 클라이언트에 내장되어 있는 도메인 주소를 사용해 판단할 수 밖에 없다.

이 내용은 물론 서버에 접속을 하기도 전에 필요한 내용이다. 접속할 서버의 도메인 주소가 패치 등으로 인해 바뀐다면, 네트워크에 연결되지도 않은 클라이언트가 자신이 접속할 서버의 주소를 알기 위해, 네트워크로 서버에 연결해서 접속할 주소를 업데이트해야 하는 모순이 발생한다.

그러므로, 접속할 주소의 IP 는 바뀔 수 있어도, 접속할 네트워크 주소의 도메인 이름은 절대로 바뀌어서는 안 된다.

- **C-6-4-4.** 게임 클라이언트의 버전 점검의 순서는 다음과 같다.

실행 파일의 버전을 점검한다.
➔ 새로운 실행 파일을 받는다.

⬇

데이터 스크립트의 버전을 점검한다
➔ 새로운 데이터 스크립트들을 받는다.

⬇

리소스 데이터들을 확인한다.
➔ 최신 버전에 맞는 리소스 데이터들을 가져온다.

⬇

필요하다면 응용 프로그램의 첫 번째 프로세스부터 다시 시작한다.

⬇

최신 버전으로 게임 클라이언트의 실행 준비가 완료

- **C-6-4-5.** 반드시 실행 파일의 버전을 데이터의 버전보다 먼저 확인하고, 이를 사용자에게 안내할 수 있어야 한다.
: 새로운 버전의 응용 프로그램 설치가 먼저 필요한 경우에는, 리소스 데이터의 다운로드를 진행해서는 안 된다.

※ 이 과정을 설계할 때의 심각한 실수 중 하나가, 서비스 이후에 응용 프로그램을 새로 설치해야 하는 경우와, 리소스 데이터의 업데이트를 해야 하는 경우를 제대로 분리하지 않는 것이다.

그러다 보니, 버전 1.5 를 설치한 상태에서 오랜만에 게임에 접속했더니, 약 600MB 나 되는 리소스를 모조리 다운로드 받은 뒤, 다시 버전 2.0 애플리케이션으로 재설치가 필요하다는 팝업창을 보는 어처구니 없는 일이 발생한다. 물론, 버전 2.0 애플리케이션으로 '재설치' 한 이후에는 600MB 가 넘는 리소스를 다시 내려 받아야 한다.

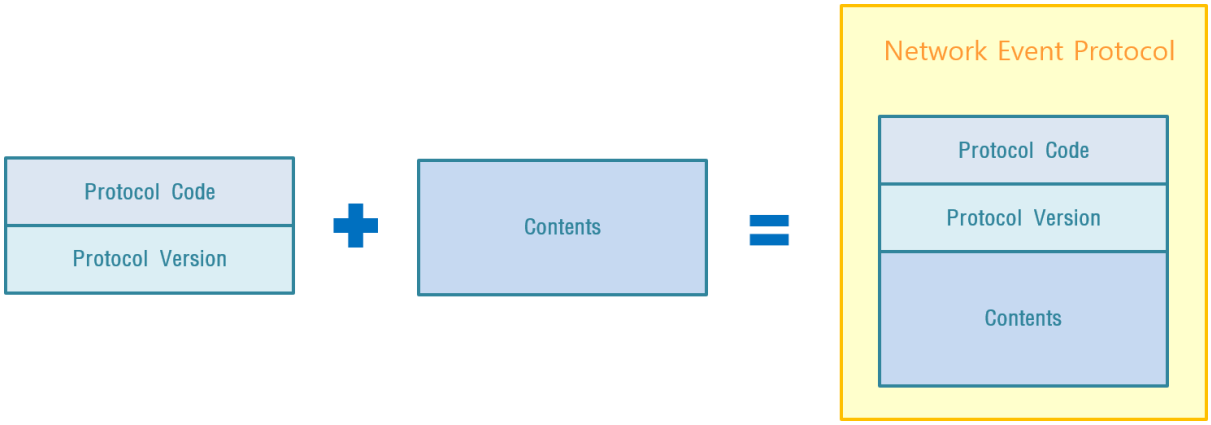
위와 같은 상황은 의외로 많은 게임에서 발견할 수 있으며, 사용자에게 큰 불쾌감을 주는 데서 끝나지 않고, 금전적인 손실을 발생하게 할 수 있다. (모바일 데이터 사용량은 요금제에 따라 사용자에게 추가적인 요금을 발생하게 할 수 있다.) 이러면 제품의 첫 인상에 아주 안 좋은 영향을 미칠 수 밖에 없다.

실행 파일의 버전 점검과 리소스 데이터의 버전 점검은 반드시 분리해야 하며, 많은 네트워크 전송이 필요한 리소스 데이터의 업데이트는 꼭 필요한 상황에서만 한 번씩 작동하도록 설계하는 것이 중요하다.

- **C-6-4-6.** 실행 파일의 버전을 동기화하도록 안내하는 방식
: 새로운 버전의 애플리케이션으로 설치하라는 안내 팝업을 띄우고, 마켓 스토어에 대한 링크를 제공해준다.

◆ C-6-5. 통신 프로토콜 구조와 송 / 수신 규칙

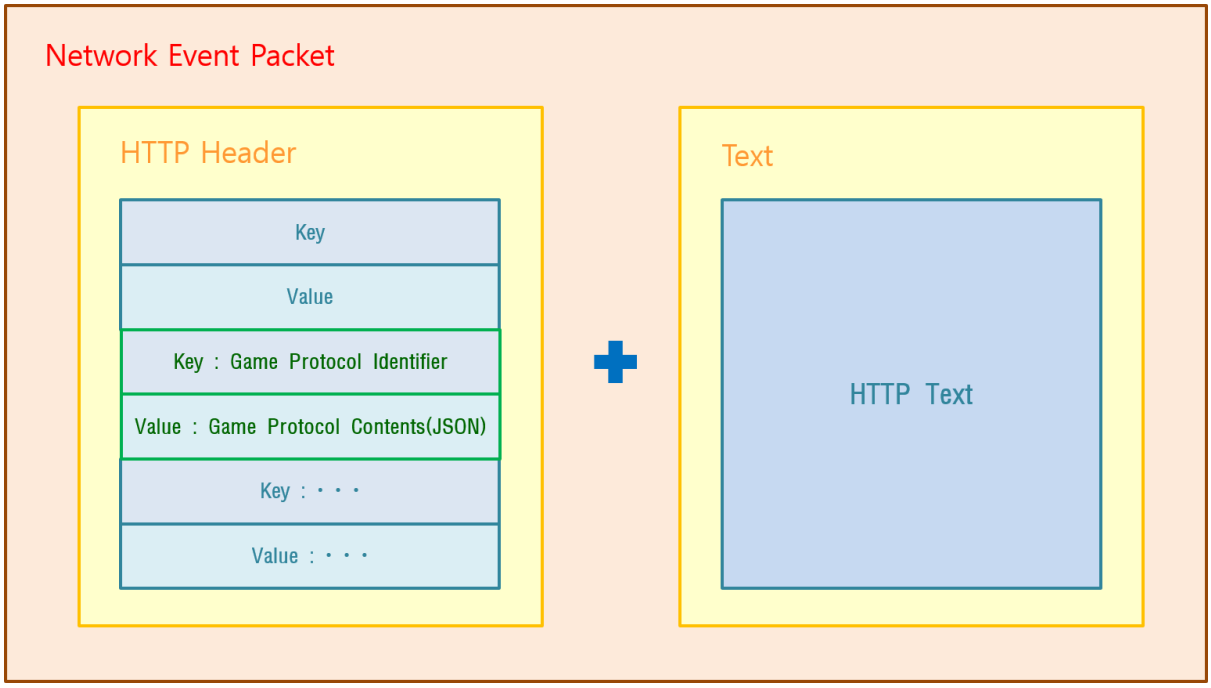
- **C-6-5-1.** 게임의 통신 프로토콜 데이터의 구성
: 기본적으로 HTTP로 통신하며, 모든 패킷들이 공통으로 사용하는 통신 프로토콜의 분류 코드 번호와, 프로토콜 별 버전 번호, 내용으로 이루어진다.



<게임에서 사용할 네트워크 이벤트 프로토콜의 기본 골격>

- **C-6-5-2.** 클라이언트 – 서버 간 데이터 통신은 **JSON 형태의 일반 텍스트 문자열**들을 HTTP로 주고 받는 방식으로 통신한다.

- **C-6-5-3.** 보낼 패킷 데이터는 HTTP 헤더의 키 - 값 쌍(Key - Value Pair) 컨테이너 안에 집어넣어서 송신한다.



<실제 HTTP로 전송할 때는 대략 이런 모습으로 전송이 된다.>

※ 텍스트 부분에 넣지 않고 헤더 컨테이너에 넣은 이유는, 텍스트 부분에서 게임 이벤트 용 패킷과 HTTP 헤더가 섞여 있기 때문이다.
그래서 이를 수신 받는 쪽에서 이 부분을 해석해서 게임 이벤트와 관련된 JSON 텍스트만 쏙 뽑아내야 하는데, 이 과정이 좀 번거롭다. .NET Framework 에서는 헤더 컨테이너에 간단히 접근할 수 있는 인터페이스를 제공하기 때문에, 그냥 이걸 이용하는 게 훨씬 편하다.

- **C-6-5-4.** 패킷을 주고받을 경우의 버전 점검
: 각 패킷마다 고유한 별도의 버전 번호를 둔다. 그리고 서버에서는 패킷 코드가 맞지만 버전 번호가 맞지 않는 패킷이 수신되면 패킷 처리를 거부해야 한다.

※ 통신 프로토콜의 구조가 바뀌는 경우, 서버와 클라이언트 모두 실행 파일을 교체할 수 밖에 없는 상황이므로, **'통신 프로토콜의 버전이 바뀌는 경우가 있다 = 클라이언트 실행 파일의 버전이 바뀐다.'**고 봐도 좋다.

- **C-6-5-5.** 통신 프로토콜 객체들은 JSON 텍스트 형식으로 직렬화(Serialization)와 역직렬화(Deserialization)가 가능해야 한다.

※ C# 언어 기준으로는 class 를 사용할 수 있지만, 직렬화할 멤버들은 모두 public 선언해야 하며, readonly 등의 한정자를 사용할 수 없다.
또한, public property 도 피해야 한다. (멤버 변수와 멤버 변수를 설정하는 프로퍼티가 둘 다 public 이면 직렬화할 때 오류가 난다..)

- **C-6-5-6.** 클라이언트의 네트워크 이벤트 요청을 서버가 처리하는 순서는 다음과 같다.
 : 이것이 가장 일반적으로 네트워크 이벤트를 처리하는 경우이다.

1. 클라이언트가 서버에게 이벤트를 처리해달라고 요청하는 프로토콜을 송신한다.

2. 서버는 수신한 패킷으로부터 프로토콜 식별 코드와 버전이 유효한지 여부를 점검한다.

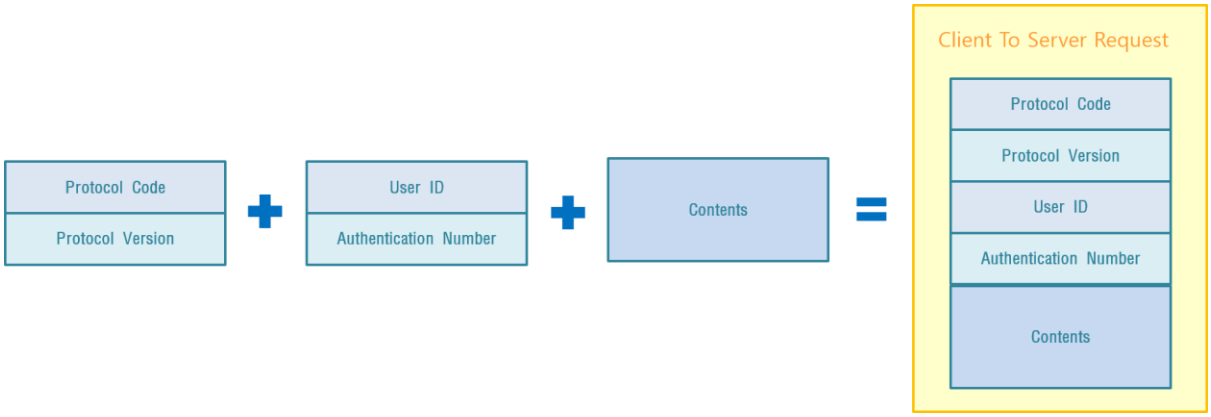
3. 수신한 패킷 내부의 값이 프로토콜 형식에 부합하는지, 값 범위가 정당한지 여부 등의 유효성을 점검한다.

3. 서버의 인증 모듈이 요청한 클라이언트의 사용자 식별 번호와 인증 번호를 검증한다.
 : 계정이 유효한지 여부와 더불어, 정보 갱신을 위해 어느 게임 DB에 접근해야 하는지 알아내기 위해서이다.

3. 해당 프로토콜을 처리해야 하는 서버 모듈(대개는 게임 모듈이지만, 여러 모듈을 번갈아 가며 처리해야 할 수도 있다.)에서 요청 내용을 처리하고, 그 결과를 데이터베이스에 반영한다.
 : 혹은 데이터베이스의 저장 프로시저를 통해 처리를 주문하고, 그 결과를 받는다.

4. 요청자 클라이언트에게, 응답할 결과와 전달할 내용을 각각 결과 프로토콜과 응답 프로토콜로 만들어 전송한다.

- **C-6-5-7.** 클라이언트가 서버에 요청하는 프로토콜에는, 요청자를 식별할 수 있도록 요청한 클라이언트의 사용자 식별 번호와 인증 번호를 같이 보낸다.



< 클라이언트가 서버에 요청하는 프로토콜의 기본 골격 >

- **C-6-5-8.** 클라이언트 대 서버의 통신에서 사용하는 패킷 데이터의 형식과, 서버 대 서버의 통신에서 사용하는 패킷 데이터의 형식을 엄격히 구분한다.

- **C-6-5-9.** 클라이언트가 보낸 패킷 데이터를 그대로 서버 대 서버의 통신에서 사용하지 못한다.
: 서버 대 서버의 통신 프로토콜에 클라이언트가 보낸 패킷 데이터를 사용해야 할 경우, 반드시 서버에서 새로운 패킷 데이터를 만들어서 내용을 채워 넣어서 사용한다.
: 클라이언트가 보낸 데이터는 그게 무엇이든, 기본적으로 신뢰할 수 없는 데이터라고 가정하고 처리해야 한다.

◆ C-6-6. 클라이언트 – 서버 간 실패와 예외 상황의 처리

- **C-6-6-1.** 모든 실패와 예외에 상황은 **오류 번호**로 분류하고, 발생했을 경우 로그를 남긴다.
: 단, 비슷한 경우 및 불가피한 상황에 대해서는, 예외적으로 오류 번호를 추상화하여 통합할 수도 있고, 그냥 텍스트 메시지로만 전달해야 할 수도 있다.

- **C-6-6-2.** 사용자가 게임 또는 장치를 강제 종료한 경우
: 혼자 플레이 하는 경우, 연결 유실에 대한 손실을 본인이 감수한 것으로 판단하고 해당 스테이지를 복구하지 않는다.
실시간 동기화 방식의 멀티 플레이인 경우, 재진입 기능이 필요하다면 해당 게임 모드나 스테이지에 다시 참여하는 기능을 둔다. 다만, 재진입을 하기 전에 해당 게임 모드나 스테이지가 사라진 경우에는 이에 대해 알림 처리를 하고 복원 프로세스는 종료한다.

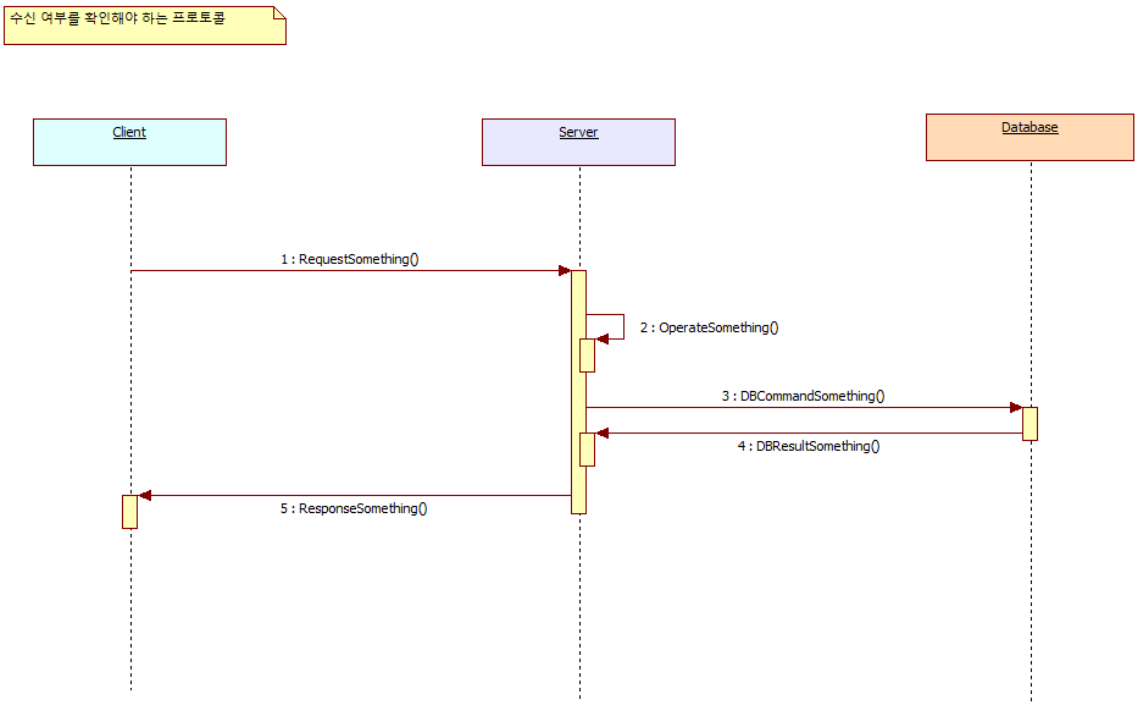
- **C-6-6-3.** 사용자가 바탕 화면으로 돌아간 경우
: 이 때 장치에서 네트워크로 패킷을 주고 받을 수 없는 상태이면, 그냥 **연결을 끊는다.**
연결이 끊긴 사용자의 게임 데이터는, 일반적인 연결 유실 처리 방침에 따라 게임 결과 등을 처리한다.

※ iOS의 경우, Home 버튼을 눌러 OS 화면으로 되돌아가는 경우, 네트워크 패킷을 받을 수 없는 상태가 된다고 한다. 이 경우는, 연결을 살려가면서 처리하기가 매우 복잡한 문제가 되기 때문에, 그냥 연결 유실로 처리하는 게 안전하다고 판단했다.

- **C-6-6-4.** 클라이언트 측이 서버에 네트워크 이벤트 요청을 했는데, 연결을 할 수 없는 경우에는 다음과 같은 절차로 진행한다.

1) 10초씩 3회 재시도한다.
2) 3회 재시도 이후에도 연결이 실패한 경우, 연결을 다시 시도할지, 아니면 애플리케이션을 종료하거나 로그인할지 사용자에게 물어본다.
3-1) 사용자가 연결을 다시 시도하기로 한 경우, 1)부터 다시 시도한다.
3-2) 사용자가 애플리케이션을 종료하거나, 혹은 다시 로그인하기 위한 선택을 한 경우, 애플리케이션을 종료하거나, 혹은 로그인 화면으로 내보내어 접속 과정을 처음부터 다시 진행하도록 한다.

※ 10초 간 기다리거나, 3회까지 시도하는 횟수의 구체적인 수치는 특별한 이유가 있어서 설정한 것은 아니다.
필요하다면 다른 값으로 얼마든지 바꿀 수 있다. 사용자가 느끼기에 지루하지 않은 적절한 값을 경험 법칙으로부터 찾으면 될 것이다.



<비-검증 방식의 네트워크 이벤트 프로토콜의 처리 과정>

- **C-6-6-5.** 서버가 클라이언트 측의 요청을 받아들여 처리한 결과를 클라이언트가 전송했는데, 클라이언트가 이를 수신하지 못한 경우에는 다음과 같이 처리한다.

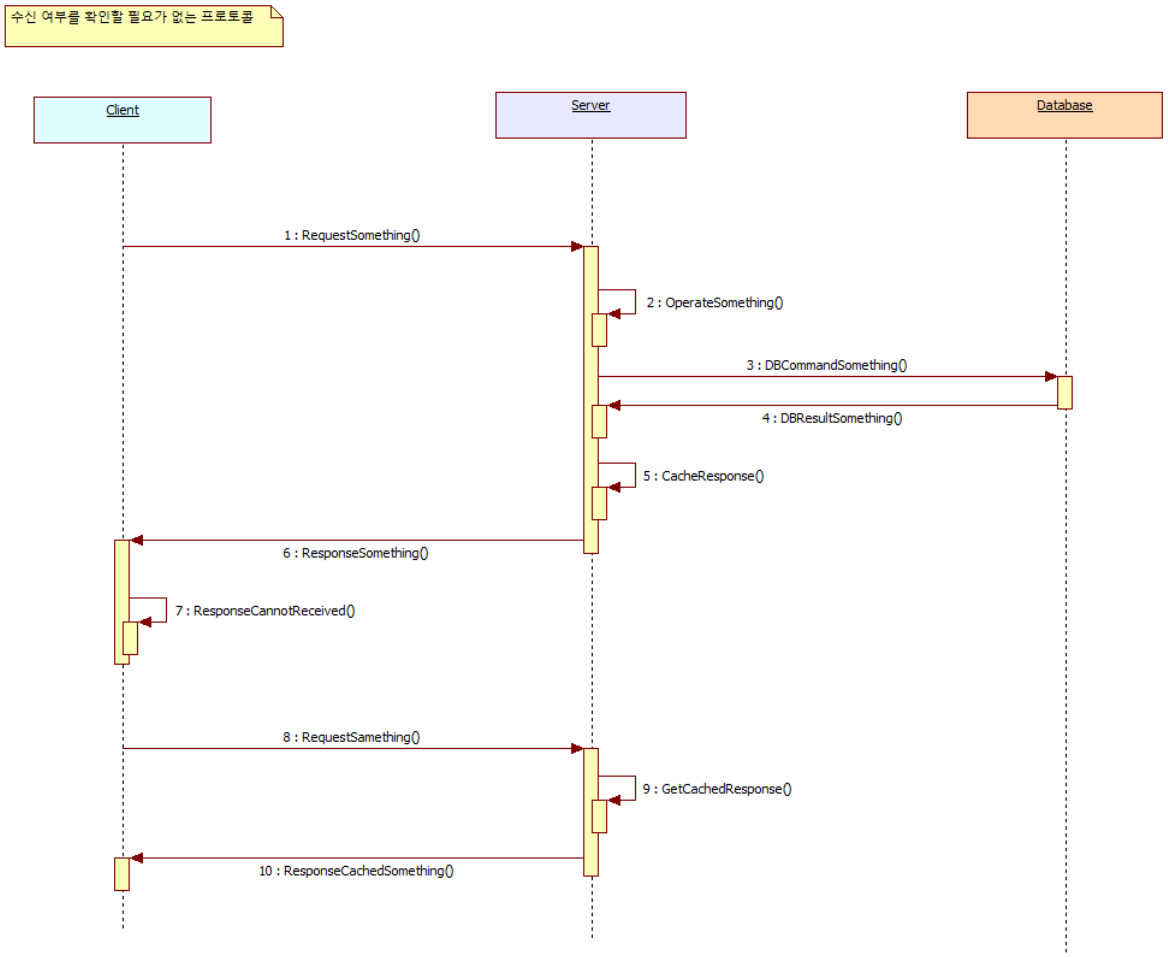
1) 모든 프로토콜에 대해서 다음과 같이 구분을 미리 해줘야 한다.
   - 서버로부터의 응답을 받았는지 검증할 필요가 없는 프로토콜 (Non-confirmative Protocol 비-검정 프로토콜)
   - 서버로부터의 응답을 받았는지 검증을 해야만 하는 프로토콜 (Confirmative Protocol, 검정 프로토콜)

2) 비-검정 프로토콜은 클라이언트가 원할 때마다 서버에 호출하고, 서버는 클라이언트가 이를 응답 받았는지에 대해 확인하지 않는다.

3) 검정 프로토콜을 클라이언트가 호출한 경우, 서버는 클라이언트의 호출로 인한 처리 결과를 저장한다. (어느 정도 과거까지 저장할지 여부는 상황에 따라 다르다. 다만, 적어도 직전 호출에 대한 처리 사항은 저장해야 한다.)

그리고, 만약 최초 요청에 대한 응답을 받지 못한 클라이언트가, 같은 내용 프로토콜에 대해 다시 요청을 하는 경우, 이전에 처리했던 결과를 응답해줘야 한다.

그러면, 클라이언트는 서버가 처리했던 상황에 대해 전달을 받고, 적절하게 현재 처리해야 할 내용을 조절하여 구현한다.



< 검증 방식의 네트워크 이벤트 프로토콜의 처리 과정 >

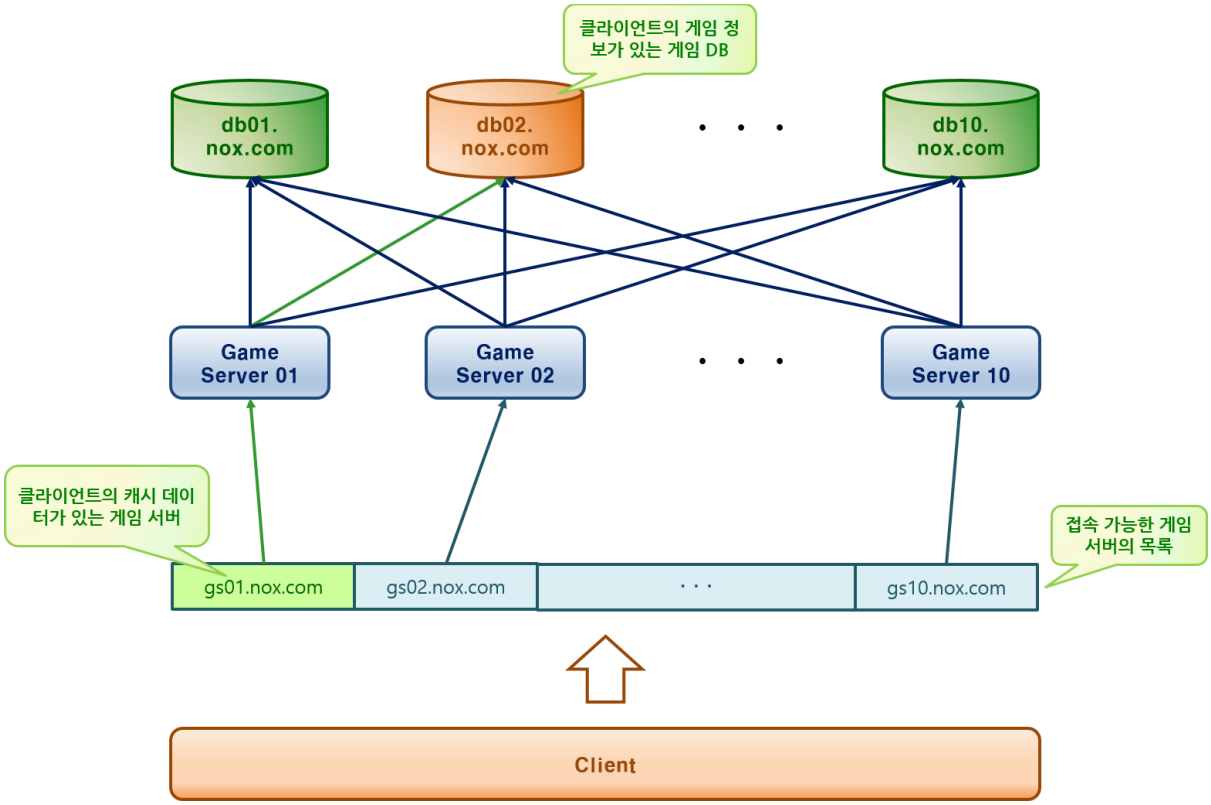
◆ C-6-7. 로드 밸런스 처리

- **C-6-7-1.** 게임 정보를 처리하는 웹 서버에서는 Network Layer 4 단계에서의 로드밸런스를 이용하지 않는다.
  : 각 웹 서버들이 접속한 세션에 대한 캐시 정보를 가지고 있고, 이를 최대한 활용하게 만들기 위해, 접속할 웹 서버를 지정하는 방식을 쓰기 때문이다.

※ 여기서의 '게임 서버'는 프로젝트 전체의 웹 서버를 통칭하는 말이 아니다.
사용자의 인증 정보를 처리하는 인증 서버, 결제를 처리하는 결제 서버처럼, 게임 논리의 처리

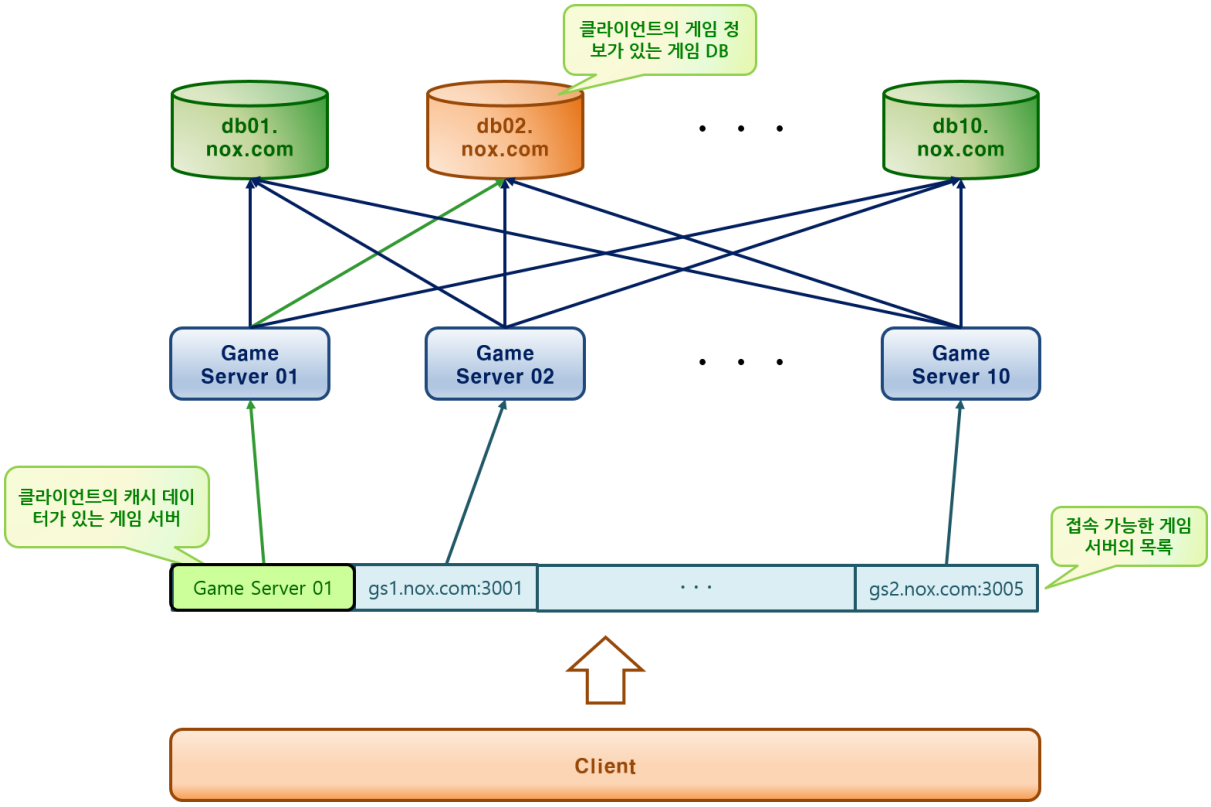
를 전담하는 역할 구분으로서의 게임 플레이 서버를 가리키는 말이다.
게임 프로젝트의 **전체 서버에 대해 로드 밸런싱 자체를 이용하면 안 되는 것으로 오해해서는 안 된다.**

- **C-6-7-2.** 각 클라이언트는 접속할 수 있는 게임 서버의 주소 목록과, 현재 접속한 게임 서버의 주소를 내려 받는다.
: 그래서 가급적 캐시 데이터가 존재하는 게임 서버로 반복해서 접속한다.



<클라이언트는 가급적 캐시가 있는 게임 서버로 접속하도록 유도한다.>

- **C-6-7-3.** 하나의 기계에 여러 게임 서버 프로세스가 존재할 수도 있다.
: 프로세스마다 별개의 게임 서버로 취급한다.



◆ C-6-8. 로그 정책

- **C-6-8-1.** 모든 종류의 로그에는 **로그의 타입 식별 번호**를 붙인다.
  : 이는 타입 식별 코드(General Type Code)와 고유 식별 번호의 차이(Unique ID)와 같다.

- **C-6-8-2.** 모든 로그 인스턴스는 다음의 형식을 바탕으로 남겨야 한다.

분 류	설 명
로그 ID	• 모든 로그를 개별적으로 식별할 번호이다. • 로그 ID 는 서로 겹칠 수 없다. • 로그의 기록 일시가 존재함에도 불구하고 로그 ID 를 별도로 두는 이유는, 로그의 기록 일시는 겹칠 수 있기 때문이다. (같은 시간에 2 개 이상의 로그가 기록될 수도 있다.)
로그 타입	• 로그에 타입을 매겨놔야, 로그를 수집하거나 가공할 방식을 결정하기가 좋아진다. • 단순 기록을 위한 경우도 있고, 시스템에 경고를 하거나, 사용자의 부정 행위를 탐지하고자 하는 등 로그를 남기고자 하는 목적은 다양하기 때문이다. 이를 구별해줘야 한다.
기록 일시	• 로그에 기록 일시가 없다면 아무 짝에도 쓸모가 없다.

	• 그럴 리는 없겠지만, 연도를 두 자릿수로 축약하는 일이 있어서는 안 된다. • 가급적 일시를 기록할 때는 가장 정밀한 단위로 기록하기 바란다.
패킷 코드	• 이 프로젝트의 서버 구조는 요청을 하면 응답을 주는 구조로써, 스스로 상태를 유지하지 않는다. • 따라서, 서버가 어떤 연산을 했다면, 반드시 이에 대한 요청이 있었다는 뜻이며, 이 요청의 종류를 식별할 수 있는 패킷 코드를 공통적으로 추출할 수 있다.
내용	• 로그의 내용 구성은 다양한 방식으로 남길 수 있다. • 검색의 편리를 위해서 테이블의 열을 할당하는 방식, 혹은 그냥 로그의 정보를 일정한 포맷의 텍스트 데이터(아마도 JSON 형태) 형태로 남겨도 무방하다.

※ 로그가 비록 지속적으로 쌓이는 데이터라고는 하지만, 8바이트 정수형 이상의 값 범위를 가지는 자료형이라면, ID를 재활용 없이 순차적으로 고유하게 부여할 수 있다고 본다.
좀 잉여스러운 계산이기는 하지만, 그 근거는 다음과 같다.
1인이 좀 과도하게 하루에 50게임씩 할 수 있고 1게임당 평균 200개의 로그를 남기게 한다면(거의 말도 안 되는 가정이다. 디스크 용량 문제 때문에 이렇게까지 할 수가 없다.), 1인은 하루에 평균 1만 개의 로그를 남긴다고 가정을 한다.
이러면 1년에 1인이 최대로 평균 365만 개의 로그를 남길 수 있다. 10년이면 3,650만 개이다.
이 상태로는 10억 명이 10년간 이 짓을 한다고 해도 3,65경 개의 로그를 쌓을 수 있다. (저장 장치가 몇 개가 필요할까?) 그래도 아직 64비트 unsigned long의 최대치인 1,844경에는 한참 부족하다.

- **C-6-8-3.** **한 번 데이터베이스에 기록한 로그의 원시 데이터(Raw Data)는, 어떠한 경우에도 그 내용을 변경하지 않는다.**
: 다만, 로그의 기록 형식은 데이터베이스의 스키마(Schema)가 바뀌는 경우 등, 피치 못한 경우에는, 한 번 기록한 이후에도 변경이 될 수 있다. 하지만 이는 형식만 바뀌는 것일 뿐, 내용을 바꿔서는 안 된다.

- **C-6-8-4.** 서버와 클라이언트, 서버와 서버 간에 네트워크를 통해 서비스 기능을 호출하거나 이에 응답한 내용들은 모두 일정한 형식의 로그를 남긴다.

- **C-6-8-5.** 서버의 내부 처리에 대한 결과는 필요할 경우에 로그를 남긴다.
: 로그의 형식(Format)은 다른 경우와 다름 없다.

◆ C-6-9. 서버 및 서비스 공격에 대한 대책

- **C-6-9-1.** 잦은 로그인 연결 / 연결 끊기 시도
: 한 번 로그인할 때마다, 실제 연결 시간과는 관계 없이 최소한 몇 초간 대기 시간을 가지게 만든다.
만약, 클라이언트를 개조(?)하든가 해서 최소 대기 시간 이내에 로그인 연결이나 연결 끊기를 시도하면, 최소 대기 시간 이내의 요청은 무시한다. (그리고 요주의 로그를 남긴다.)

- **C-6-9-2.** 같은 의미를 가진 요청 패킷을 짧은 시간에 여러 차례 전송
: 패킷을 1초에 여러 차례 보내야 하는 경우가 있을 수가 없으므로, 패킷을 보내는 행위 자체에도 최소 대기 시간을 둔다.

- **C-6-9-3.** 아주 커다란 크기의 패킷을 보내려고 시도
: ASP.NET의 <web.config>옵션으로 요청 패킷 크기를 제한한다.

- **C-6-9-4.** 패킷 코드 번호는 같은데, 그 외의 값 객체의 형식과 내용을 다른 것으로 바꿔서 보내려고 시도
: 패킷 별로 부여한 버전 번호가 서버에서 처리하는 것과 같은지 비교한다.
또한, 클라이언트가 패킷에 담아 보낸 내용을 데이터베이스에 그대로 반영하는 부분을 찾아서 없앤다.
그 외의 경우는... 서버에서 프로토콜 객체로 변환을 시도할 때 예외가 발생하든지 할 것이므로, 충분히 탐지하고 오류를 반환하는 식으로 처리할 수 있을 것으로 본다.

◆ C-6-10. 클라이언트 계정의 생성

- **C-6-10-1.** 각 클라이언트 계정들은 고유한 식별 번호를 가진다.
: 이 식별 번호는 계정의 이름과는 별도로 구분하는 데이터이다.

- **C-6-10-2.** 클라이언트 계정의 식별 번호는 8바이트 부호 없는 정수형을 사용한다.
: 값 범위는 0 ~ 약 1,844경이다. (단, 값 0은 무효한 값이다.))

※ 계정 이름도 겹치지 않아야 하는 조건이 있으므로, 식별자로 사용할 수 있다.
그러나, 컴퓨터는 문자열보다 숫자 값이 훨씬 다루기 편하고 연산이 빠르기 때문에, 각 계정마다 고유한 숫자를 할당하는 것이다.

- **C-6-10-3.** 클라이언트 계정의 식별 번호는 한 군데의 데이터베이스에서만 발급을 담당한다.
: 생성한 계정의 식별 번호는 여러 군데에서 보관할 수도 있으나, 발급 자체는 반드시 한 군데에서만 담당한다.

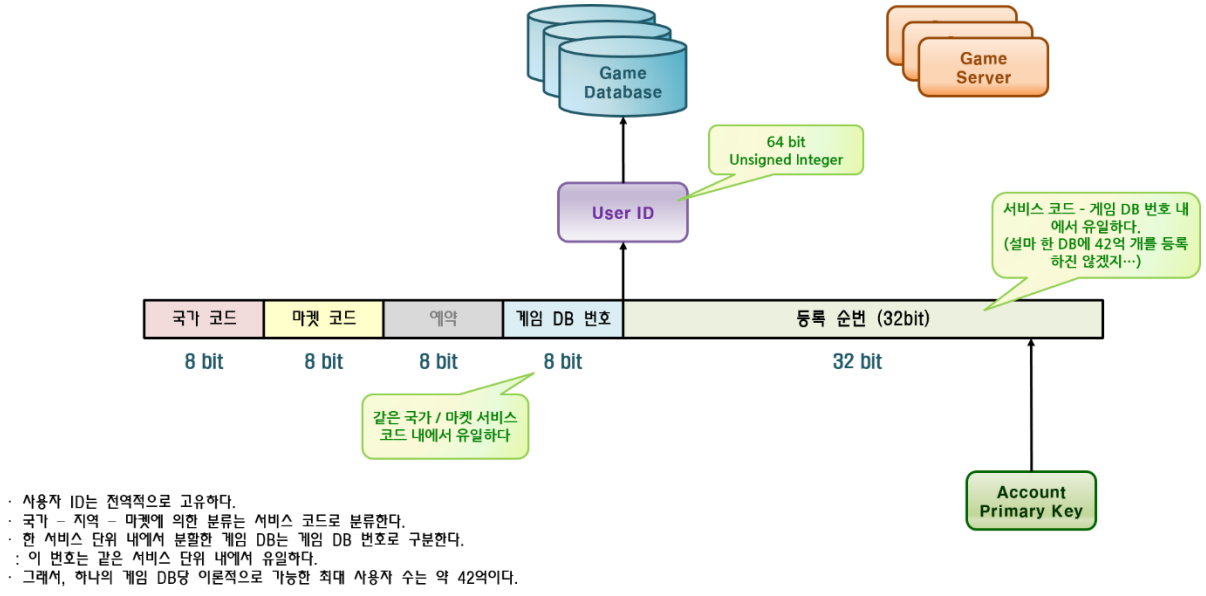
※ 여기서의 의미는, 계정을 고유하게 구분해야 할 필요가 있는 서버 지역 별로 한 군데에서만

담당해야 한다는 의미이다.
서버 지역은 전 세계에 하나만 있는 경우도 있지만, 몇 개로 쪼개어져 있을 수도 있다. 이 때는 쪼개어진 각각의 서버 지역마다 계정 발급 담당이 하나씩 존재한다. 물론, 이 경우에는 서버 지역끼리는 기본적으로 완전히 분리해서 게임을 서비스한다고 가정하고 있다.

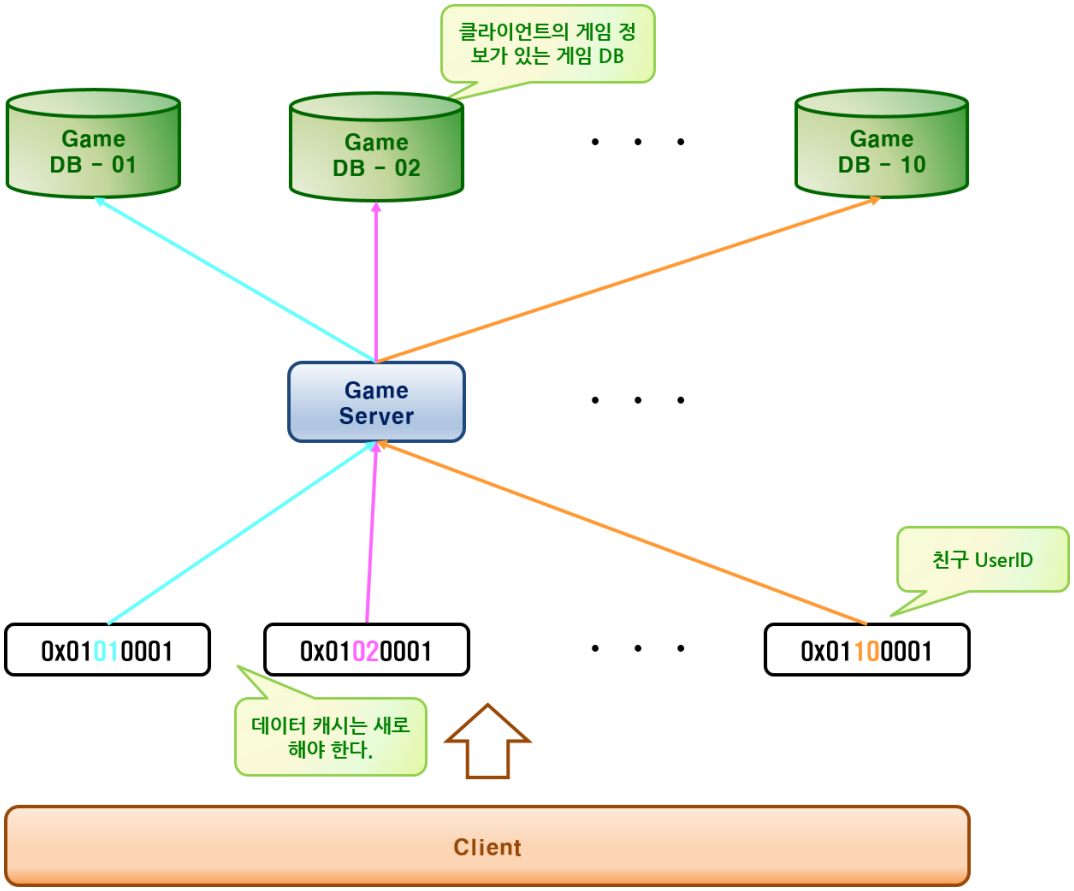
- **C-6-10-4.** 한 번 생성한 클라이언트 계정의 식별 번호는 **도중에 다른 값으로 변경하거나, 다른 계정 식별을 위한 식별 번호로써 재사용하지 않는다.**

- **C-6-10-5.** 인증 서버 및 데이터베이스를 제외한 다른 서버 모듈들과 데이터베이스에서는, 각 플레이어를 식별하는 데, 이 클라이언트 계정의 식별 번호를 이용한다.
: 즉, 인증 데이터베이스는 로그인 한 사용자의 계정 정보를 통해서 클라이언트의 식별 번호를 가져올 수 있어야 한다.

- **C-6-10-6.** 각 사용자의 고유 식별 번호는, **그 사용자의 실제 게임 데이터가 저장되어 있는 게임 DB를 식별할 수 있는 정보를 포함**한다.
: 게임 DB는 접근 빈도 및 사용 성능상의 문제로 인해 분산할 가능성이 매우 높다.
그런 상황에서 이런 기능이 없다면, 사용자의 실제 게임 데이터를 찾으려고 할 때마다 인증 DB를 뒤져야 하는 상황이 될 수 있다. 이는 인증 DB를 성능 병목 지점이 되게 만들 수 있으므로, 인증 번호 그 자체에 게임 DB 번호를 포함할 수 있는 수단을 주는 게 좋다.



<사용자 번호 디자인 방식>



<같은 게임 서버에 접속하더라도, 데이터에 접근하는 게임 DB는 다를 수 있다.>

◆ C-6-11. 클라이언트 계정의 삭제 및 정지

- **C-6-11-1.** 클라이언트의 계정은 사용자가 삭제를 요청할 경우, 접근할 수 없는 상태로 변경하는 식으로 삭제 처리한다.
  : 즉, **실제로 지우지는 않는다**는 뜻이다.

※ 사용자의 마음이 바뀌어서, 다시 계정을 원래대로 복구하기를 바랄 경우에는 이러한 방식의 삭제 처리는 도움이 된다.

※ 계정 DB 뿐 아니라, 삭제가 필요한 DB 데이터의 경우에는 실시간으로 데이터를 제거하는 것보다는, 삭제를 허가하는 표식을 해두고, 나중에(주로 이용자들이 거의 없는 시간대에) 일괄 삭제하는 편이 더 좋다고 한다.
이는 DB 스토리지의 파편화를 줄여주어 최적화 상태를 유지하기에 더 좋다고 함.
다면, 현재 시점(2016년~)에서는 이미 DB 서버에 SSD가 대중화된 상태이기 때문에, 이런 접근 방식이 정말로 효과가 큰지 조사를 좀 더 해봐야 할 수 있다.

- **C-6-11-2.** 단, 삭제를 요청한 계정 정보를 일정한 기간 동안 보관하다가 영구히 삭제하거나,

(복구 불가능) 활성화한 계정과 별도로 분리해서 보관해야 할 수 있다.

※ 이는 제품을 서비스하는 지역에 따라 개인 정보에 관한 법률과 정책에 따라야 할 필요가 있기 때문이다.

- **C-6-11-3.** **징계에 의한 계정 접속 정지 처리**를 할 수 있어야 한다.
: 접속 정지 상태임을 식별할 수 있는 타입 값과, 접속 정지가 해제될 시간을 지정할 수 있으면 된다.

- **C-6-11-4.** 특정 계정에 대한 접근 정지는 사용자 계정의 단위로만 할 수 있다.
: 사용자 계정 중에서 특정 캐릭터만 접속할 수 없는 식의 처리는 하지 않는다.

◆ C-6-12. 클라이언트 계정의 인증 처리

- **C-6-12-1.** 계정 식별 및 인증 관련 도구들의 정의와 특징

인증 도구	설 명
계정 이름 (Account)	• 사람이 구분할 수 있는 계정의 식별 명칭 • 계정을 처음 생성하거나, 수동으로 로그인해야 하는 경우에 사용한다. • 사용자 클라이언트에 저장하지 않는다.
계정 암호 (Password)	• 개인 계정을 보호할 용도로 사용하는, 사용자만 기억할 수 있는 문자와 숫자, 특수 기호의 조합으로 이루어진 비밀 문자열 • 계정을 처음 생성하거나, 수동으로 로그인해야 하는 경우에 사용한다. • 해당 계정과 대응하는 암호 문자열을 정확하게 틀리지 않고 입력해야 로그인할 수 있다. • 사용자 클라이언트에 저장하지 않는다.
전자 우편 (E-Mail)	• 계정 관련 정보의 수정 사항이나, 암호를 잊어버리는 등의 비상 상황에서 연락 도구로 사용할 전자 우편 수단 • 전자 우편 계정에 대해 별도로 인증하지 않으며, 전자 우편의 수신이 가능한 주소로 입력하는지 여부는 각 계정 사용자의 책임이다. : 단, 전자 우편 주소의 형식을 검증하거나, 전자 우편의 송 / 수신이 가능한지 여부를 점검하는 모듈이 들어갈 수 있다. • 사용자 클라이언트에 저장하지 않는다.
계정 식별 번호 (UserID)	• 계정 이름(Account)마다 부여하는 식별 번호. • 서버에서는 검색 속도를 위해 수동 로그인 과정을 제외하고 이 식별 번호로 계정을 구분한다. • 한 번 부여하면 변경되지 않는다.

	• 클라이언트는 계정 식별 번호를 (암호화한 형태로)들고 있다가, 자동으로 로그인하는 데 사용할 수 있다.
계정 인증 번호 (AuthNumber)	• 사용자가 자신의 계정에 로그인하면, 그 계정에는 UserID 와는 다른, 별도의 인증 번호가 자동으로 부여된다. • 형식은 64 비트 이상의 정수형 숫자, 또는 128 비트 기반 UUID • 사용자가 계정에 접속하거나, 계정 정보를 변경할 때마다, 인증 서버는 그 계정의 인증 번호를 새로 발급한다. • 클라이언트는 이 인증 번호를 가지고 있다가, 게임 서버에 뭔가 요청을 할 때 함께 제출한다. • 게임 서버들은 계정 식별 번호와 계정 인증 번호가 모두 유효한 경우에만 이벤트를 처리하도록 한다.

- **C-6-12-2.** 로그인
: 사용자가 자신에게 할당하도록 신청한 계정과, 그 계정의 암호를 직접 입력하는 방식이다.

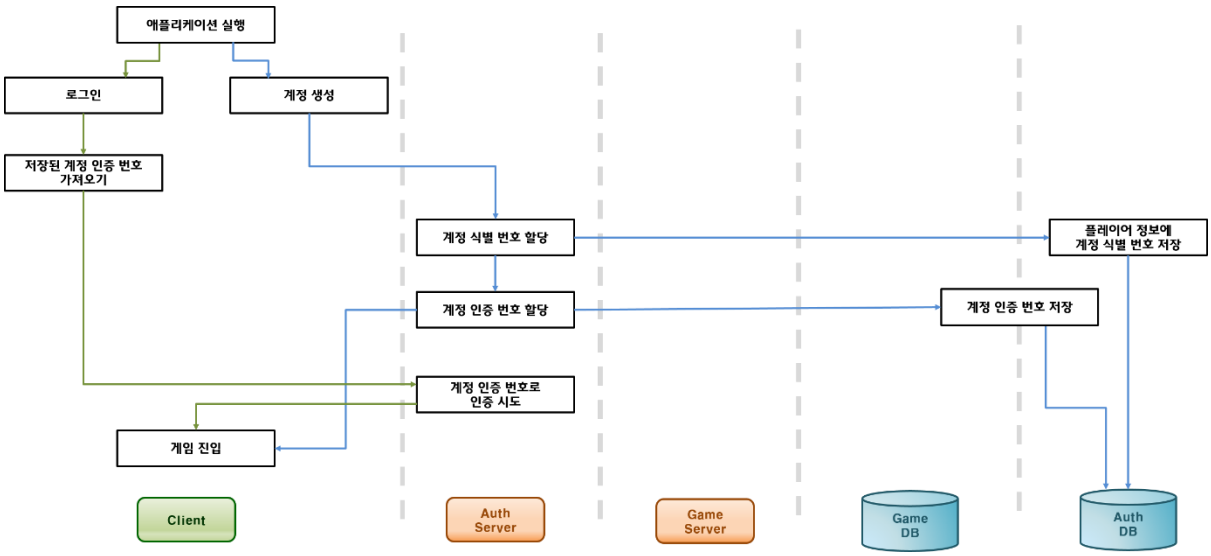
- **C-6-12-3.** 인증
: 사용자의 클라이언트 프로그램에 사용자가 사전에 획득한 계정의 식별 번호(ID) 인증 번호(Authentication Number)가 존재하는 경우, 그 사용자의 식별 번호와 인증 번호를 이용해서 사용자를 식별하는 과정이다.

※ 사용자 클라이언트에 사용자가 등록한 계정 이름과 비밀번호를 저장하지 않게 하기 위한 조치이다.
또한, 로그인이든 인증이든, 클라이언트 -> 인증 서버 -> 게임 서버를 거치는 방식 자체는 동일하다.

- **C-6-12-4.** 상황에 따라 필요한 사용자 로그인과 인증 처리

상 황	처 리
• 처음 게임 애플리케이션을 구동함	• 클라이언트에 저장되어 있는 인증 정보를 이용해서 수동으로 확인하여 로그인 • 계정과 암호를 사용자가 직접 입력해서 로그인
• 로그인 하기 전에, 게임 애플리케이션이 어떤 이유로든 간에 종료가 됨	• 클라이언트에 저장되어 있는 인증 정보를 이용해서 수동으로 확인하여 로그인 • 계정과 암호를 사용자가 직접 입력해서 로그인
• 로그인 한 이후, 게임 애플리케이션이 어떤 이유로든 간에 종료가 됨	• 클라이언트에 저장되어 있는 인증 정보를 이용해서 자동으로 인증을 시도

• 로그인 시도를 할 때, 네트워크 연결이 유실됨	• 클라이언트에 저장되어 있는 인증 정보를 이용해서 자동으로 인증을 시도
• 로그인 한 이후, 게임 진행 과정에서 네트워크 연결이 유실됨	• 클라이언트에 저장되어 있는 인증 정보를 이용해서 자동으로 인증을 시도



< 사용자 계정의 생성과 인증 처리에 대한 흐름도 >

- **C-6-12-5.** 클라이언트가 계정에 로그인 한 이후에는, 서버는 클라이언트에게 사용자 식별 번호(User ID)와 인증 번호(Authentication Number) 값을 부여해준다.
  : 이 정보는 암호화 하지 않아도 무방하다.

※ ~~이 문제에 대해 결정하지 못한 이유는 데이터베이스의 물리적인 특성과 암호화 특성에 대한 문제 때문이다.~~

사용자 ID와 인증 번호를 단 방향 암호화(MD5, ASE 알고리즘 등)로 클라이언트에게 제공해야 하는 경우를 생각해보자.

이 때는 (일반적으로는)복호화가 안 되는 방식이기 때문에, DB에서도 암호화한 방식으로 저장을 해야 한다. 아예 사용자 ID 자체를 암호화한 결과물로 저장을 하든, 별도의 열(Column)을 유지하든 해야 한다.

이렇게 되면, 사실상 인증 과정에서 ID 검색은 실제 User ID가 아니라, 암호화한 Crypt User ID로 검색을 해야 한다. User ID의 경우, 검색의 고속화를 위해 자동 증가(Auto Increment)로 할당하고 최우선 키(Primary Key)로 설정해놓는데, 실질적으로는 거의 사용하지 않게 되어버리니, 뭔가 주객이 전도된 느낌이다. 더구나, 암호화한 키의 크기는 User ID와는 비교할 수 없이 크기 때문에 사용자가 매우 많다면 인증 DB에 막대한 용량 부담을 줄 수 있다.

장점이라면, DB와 서버 어느 쪽도 인증할 때마다 복호화에 대한 성능 부담을 지지 않아도 되는 점이다.

또 다른 방법은 DB에서는 사용자 ID를 암호화해서 저장하지 않고, 대신 클라이언트에 사용자 ID를 전달할 때, 공개키 기반 암호화(PKI 방식, ASE 알고리즘이 대표적임)로 암호화해서 하는 방법을 생각해볼 수 있다.
이 방식을 쓴다면 서버에서는 클라이언트의 요청이 올 때마다, 요청자의 인증을 확인하려고 공개 키로 암호화된 사용자 ID를 서버의 비공개 키로 풀어야 한다. 이 경우의 단점은, 공개 키 기반 암호화가 단 방향 암호화보다 겁나게 느리다는 거.... 그 대신 암호 푸는 건 서버가 담당하면 되기 때문에, DB에는 성능적인 문제든, 용량 문제든 부담이 갈 일은 없는 게 장점이다.

- **C-6-12-6.** 클라이언트에서는 자동으로 서버에 로그인을 해야 하는 경우, 계정 이름(Account)와 암호(Password) 대신, 인증 서버가 발급한 사용자 식별 번호와 인증 번호를 이용해서 인증을 시도해야 한다.
: 일반적인 문자열을 이용한 인증 시도는 사용자가 수동으로 입력해서 인증할 때만 사용한다.

- **C-6-12-7.** 클라이언트가 게임 서버에 뭔가 네트워크 이벤트 요청을 보낼 때마다, 서버에서는 요청한 클라이언트의 인증 과정을 거쳐야 한다.
: 이벤트 하나 처리할 때마다 로그인 -> 이벤트 처리 -> 로그아웃이 반복된다고 보면 된다.

※ 이벤트 하나 처리할 때마다 로그인(인증) -> 이벤트 처리 -> 로그아웃을 반복한다고 보면 된다. (엄밀히 말해, 로그아웃 그 자체는 별도의 과정으로 존재하지는 않는다.)
그런데 인증 과정이 사용자 계정과 암호를 입력하는 방식으로만 되어 있다면, 클라이언트에서 서버에 어떤 요청을 할 때마다 사용자 계정과 암호를 입력하는 인증 과정을 거쳐야 한다는 말인데 이건 실질적으로 불가능한 일이다. ~~(아이템 하나 갈아입을 때마다 계정, 암호를 치고 싶은 사람?)~~
그렇기 때문에, 한 번 사용자가 올바르게 인증을 받은 계정에 대해서는, 게임 서버가 간단하게 사용자를 식별하고 인증하게 할 수 있는 수단이 필요하다. 그래서 계정 식별 번호가 생기게 되었다.

- **C-6-12-8.** 사용자의 계정과 비밀번호, 전자 우편 주소는 **암호화를 했는지 여부와 관계 없이, 클라이언트의 기기에 파일 형태로 저장해서는 안 된다.**
: 그 대신, 인증 서버에서 발급한 사용자 식별 번호(User ID)와 인증 번호(Authentication Number)를 저장해야 한다.

※ 계정 식별 번호와 계정 인증 번호 등을 두는 이유 중의 하나는, 사용자가 직접 입력하는 계정 이름들을 노출하는 경우를 없애기 위해서다.
특히, 이들이 암호화되지 않은 상태로 저장되어 있는 경우, 제 3자 제품 서비스의 인증 계정을 탈취하는 접근수단이 될 수도 있다. (실제로 많은 게임 계정들이 전혀 관계가 없는 웹사이트 서비스 계정 탈취로 인해 같이 털렸다.)

- **C-6-12-9.** 인증 서버에서 부여하는 계정 인증 번호는 영구적이지 않다.

: 게임 서버가 어떤 이유로든 인증할 수 없거나, 새로운 계정 인증 번호가 필요하다고 판단하는 경우에는, 인증 서버로부터 다시 계정 인증 코드를 부여 받아야 한다.

- **C-6-12-10.** 사용자의 계정 접속 정보가 바뀐 경우, 계정 인증 번호는 즉시 파기되고, 다음 번부터는 인증 서버로부터 새로운 계정 인증 번호를 부여 받아야 한다.

- 사용자가 명시적으로 로그아웃 처리함. (다른 계정으로 접속하기 위함이든 무엇이든...)
- 사용자가 계정의 암호를 교체함

- **C-6-12-11.** 여러 기기에서 같은 계정으로 접속 시도를 할 경우에는, **같은 계정으로는 한 번에 하나의 로그인 세션만을 허용**한다. 즉, 마지막으로 로그인한 기기의 세션을 제외하고, 나머지 이전에 로그인한 기기들의 세션은 자동으로 로그아웃 처리한다.

※ 그러므로, 사용자가 로그인을 할 때마다, 이미 로그인을 한 상태인지 점검하고, 이전 로그인 세션을 종료해야 한다. 사실, 새로운 로그인이 발생하면 계정 인증 번호가 바뀌기 때문에, 이전 세션의 인증 번호는 무효화되어서 쓸 수가 없기 때문이기도 하다.
서버에서 강제로 로그아웃 처리를 했을 때, 그 이유를 구분해서 로그를 남긴다면 운영에 있어 도움이 될 것이다.

◆ C-6-13. 캐시 정책

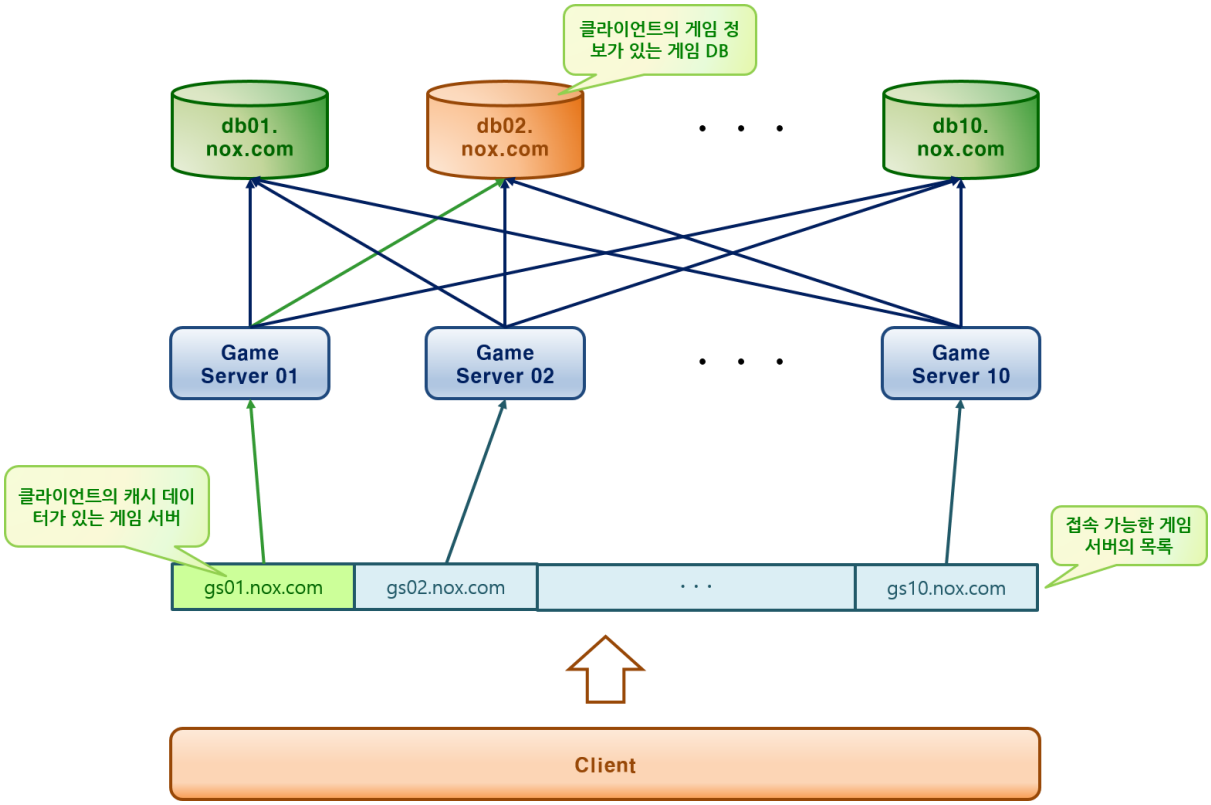
- **C-6-13-1.** 각 계정의 플레이어 데이터들은 한 번 데이터베이스에서 가져온 뒤, 쉽게 버리지 않고 캐시하여 사용한다.

- **C-6-13-2.** 서버에 캐시하는 데이터는 플레이어 단위로 이루어진다.
: 그래서, 게임 서버에서 가장 빈번하게 데이터 캐시가 이루어진다.

※ 게임 서버는 웹 서버로 구현하고 있기 때문에, 세션 데이터에 플레이어의 캐시 정보를 저장하는 방식을 사용한다.. 이는 다음과 같은 과정을 통해 이루어진다.

1) 클라이언트가 게임 서버에 네트워크 이벤트를 요청한다.
2) 서버로부터 응답을 받을 때, 웹 서버가 발급한 세션 키를 저장한다.
3) 2)의 과정에서 저장한 세션 키를 1)의 과정을 반복할 때 HTTP 헤더 정보에 넣어서 네트워크 이벤트를 요청한다.
4) 그러면, 웹 서버는 3)의 과정에서 입력된 세션 키를 이용해 세션 데이터로부터 캐시 된 플레이어의 데이터를 가져와서 처리한다. 이 과정은 데이터를 가져오기 위해 DB에 접근할 때보다 훨씬 고속으로 처리할 수 있다.

- **C-6-13-3.** 이를 위해, 클라이언트는 로그인할 때, 접속할 수 있는 서버의 주소 목록과, 현재 캐시 데이터가 있는 서버 주소(보통 마지막으로 접속한 서버가 된다.)를 받는다.



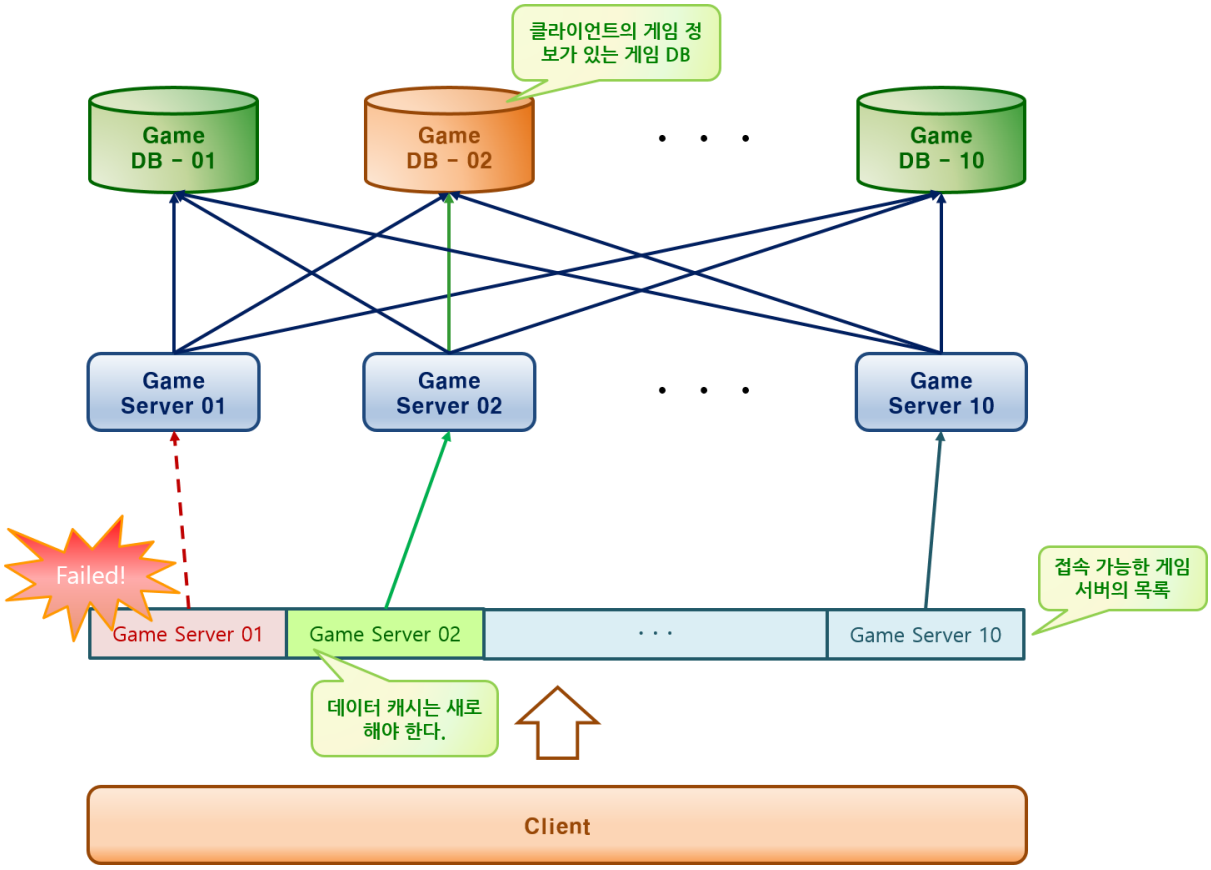
<캐시 데이터가 존재하는 서버로 접속하면, 네트워크 요청에 더 빠르게 응답한다.>

- **C-6-13-4.** 캐시 데이터가 존재하는 서버는 반드시 고정적인 것은 아니다.
  : 캐시 데이터는 수명 주기가 있어서, 장시간 네트워크 이벤트를 요청하지 않으면 사라질 수도 있다. 이럴 때는 같은 게임 서버에 접속하더라도, 네트워크 요청을 처리할 때 DB에서 다시 데이터를 가져오고, 이에 대해 데이터 캐시를 다시 하게 된다.

- **C-6-13-5.** 데이터가 캐시 되어 있는 서버에 뭔가 장애가 발생해서 접속할 수 없는 경우에, 다른 게임 서버를 통해 네트워크 이벤트를 처리하도록 요청해도 마찬가지로 데이터 캐시 과정이 다시 일어난다.

※ 데이터를 가져오기 위해 접근하는 게임 DB는 사용자 ID에 의거하여 변하지 않지만, 접속하는 게임 서버는 바뀔 수 있다. 다만, 고속으로 요청을 처리하기 위한 데이터 캐시가 사라졌기 때문에, 이를 재구축하느라 응답이 좀 더 느려질 뿐이다.

이런 과정은 사실 전혀 어려울 것이 없는 개념이다.
데이터에 접근할 때, 최초 1회만 디스크로부터 가져오고(DB로부터 가져오고), 메모리에 캐시한 뒤(서버에 캐시 한 뒤) 재활용하는 것과 같은 개념이기 때문이다.



<다른 게임 서버로 접속하면 캐시 데이터를 새로 만들어야 한다.>

◆ C-6-14. 게임 플레이 동기화

- **C-6-14-1.** 해당 사항 없음

※ 향후 실시간 동기화가 필요한 시점에서 이와 관련한 이슈가 새로 등장할 수는 있다.
실시간 동기화 관련된 이슈들은 그 시점에 재검토한다.

◆ C-6-15. 클라이언트에서의 계산 결과 검증

- **C-6-15-1.** 해당 사항 없음

C-7. 데이터베이스

◆ C-7-1. 플랫폼(Platform)

- **C-7-1-1.** 운영체제
  : 데이터베이스를 탑재하는 운영체제는 Windows를 사용한다.

※ 서버용 리눅스 시장에서는 CentOS가 주류라서 이걸로 하기로 함. 딱히 다른 이유는 없다.
개인적으로는 그래도 조금이라도 사용해 본 Ubuntu 계열이 더 익숙하지만 현장에서는 정작 CentOS가 주류인 게 걸린다. ~~(어차피 Core 옵션으로 깔면 둘 다 명령 창만 볼 텐데 뭐...)~~

- **C-7-1-2.** 관계형 데이터베이스(RDBMS)
  : Oracle MySQL을 사용한다.

※ MySQL은 자료가 방대하고, 검증되었다.
~~다만, 서비스할 때는 공짜가 아닐 수 있다. (공짜인 건 Community Edition임)~~
MySQL Community Edition과 Enterprise Edition의 차이는 지원 서비스 유무이며, MySQL 소스 코드를 수정하지 않는 한, GPL 전염이나 라이선스 비용을 부담할 필요가 없다.

MariaDB는 MySQL의 fork 프로젝트로, 자기들 주장으로는 MySQL보다 훨씬 성능이 뛰어나다고 하는데, 아직 국내에서는 그렇게 널리 퍼져 있지는 않은 듯... (단, 외국은 사정이 다름. 다들 MariaDB로 갈아탄 분위기...이건 Oracle의 탓이 더 큰 걸 수도 있다.)

※ 상당수의 발행사들이 Scale-out할 때 라이선스 비용이 덜 들어가는 리눅스 + MySQL 조합을 선호하는 경향이 있다.
또, 한국 외에는 Unix-like 기반 운영체제가 게임 서버 용으로 대세인 면도 있고...

- **C-7-1-3.** NoSQL
  : 이 프로젝트에서 게임 데이터를 저장하는 용도로의 사용을 고려하고 있지 않다.
  만약 NoDQL DB를 사용해야 하는 경우, MongoDB 혹은 Couchbase를 사용하기로 한다.

※ MongoDB는 (내 생각에...;;)대중적이고 라이선스 비용이 저렴한 선택이다.
Couchbase는 NoSQL 데이터베이스 중에서 고성능인 대신 라이선스 비용이 상당히 비싸다.(라

고 하더라..;;;)

NoSQL을 사용하지 않는다면, 그 이유는 일단 NoSQL DB를 잘 알고 서비스에 효과적으로 적용할 수 있는 사람이 현재까지는 적기 때문이다.
또한, 기존의 RDBMS가 구현하고 있는 트랜잭션이라든가, 테이블 파티셔닝 등의 유용한 기능을 사용할 수 없는 이유도 있다. NoSQL 기반 DB에는 자동 수평 분할(Auto Sharding)같은 기능이 있기는 하지만, 역사가 깊은 RDBMS 엔진들이 구현하는 기능들을 따라잡지 못한 면이 아직은 많다.

- **C-7-1-4.** In-Memory DB
: Redis를 사용한다.
단, 이 DB는 캐시 용도로 사용하며, 일차적으로는 RDBMS를 이용해서 구현한 후에 적용한다.

※ 가장 일반적으로 많이 사용하는 추세인 것 같아서 선택했다.
Memcached도 오랫동안 검증되었고, 특정 상황에서는 Redis보다 강력한 면이 있다고 해서 고려해 본 바 있으나, 결과적으로 Redis로 마음이 기울었다.
워낙 Redis의 성장세가 빠른데다, memcached의 자료구조가 (많이) 빈약하고, 그렇다고 항상 Redis에 비해 성능이 앞선다고 할 수도 없어서...

◆ C-7-2. 데이터베이스 구성

- **C-7-2-1.** 물리적 구성

• 첫 개발 단계에서는 RDBMS만으로 구성한다.
• In-Memory DB는 데이터베이스 캐시 용도로 사용한다.
• 복제 구성을 하는 경우를 제외하고, 물리적으로 DB가 분리된 경우에는 사용자들이 별도의 서버 지역으로 사용하도록 한다.
• 결제 기록, 로그 기록, 대전 기록 등 기록의 개수와 양이 많고, 기록 시간이 주된 탐색 범위에 해당하는 경우에는 일정한 기록 시간에 따라 기록 테이블을 분할하여 저장한다. (Table Partition)

- **C-7-2-2.** 논리적 구성

종류	설명
인증 DB	• 사용자 계정의 인증 정보들을 저장한다. • 사용자 계정이 어느 서버군으로 연결되어야 하는지에 대한 정보를 저장한다. • 서버군 내에서 사용자 계정의 클라이언트에게 필요한 패치 파일, 접속해야 할 게임 서버 주소 등을 알맞게 제공하는 역할을 한다.

게임 DB	• 각 사용자 계정의 게임 플레이 정보들을 저장하는 DB 이다. • 사용자가 선택한 서버에 따라 테이블을 분할하도록 한다.
결제 DB	• 사용자 계정들의 결제 정보들을 저장하는 DB 이다. • 기간에 따라 기록 테이블을 분할하도록 한다.
커뮤니티 DB	• 사용자 계정들의 채팅 활동, 친구 / 길드 활동을 저장하는 DB 이다. • 기록의 정확성은 그렇게까지 중요하지 않고, 대신 기록 속도가 가장 중요하다.
랭킹 DB	• 랭킹 관련 정보들을 기록한다. • 대전 전보들을 기록한다. • 대전 정보의 경우, 기간에 따라 랭킹 이력 테이블을 분할하도록 한다.
로그 DB	• 서버에서 일어난 요청과 처리 과정들을 기록한다. • 기간에 따라 랭킹 이력 테이블을 분할하도록 한다.

◆ C-7-3. 설계 정책

- **C-7-3-1.** 데이터베이스의 버전은 각 테이블 항목 별로, 별도로 가진다.
: 게임 버전과 상관 없이, 데이터베이스가 바뀔 때만 변경되는 별도의 버전 체계를 가지며, 이 버전들은 전체 데이터베이스 스키마들이 공통으로 소유하는 것이 아니라, 구분되는 각 항목들마다 별도로 판올림 한다.

※ 이런 방식의 장점은, 실제로 데이터베이스의 규칙이나 내용이 변경될 때만 버전을 판올림 하기 때문에, 데이터베이스의 변경 사항을 추적하기가 좀 더 쉽다는 장점이 있다. 그리고 게임 컨텐츠의 이유로 게임 버전을 판올림 하더라도, **데이터베이스의 내용이 변경되지 않는다면 굳이 데이터베이스도 같이 판올림할 필요 없이**(내용은 바뀌지 않았으니까), **그대로 기존 버전을 사용해도 무방**하다.
단점이라면, 게임 버전과 데이터베이스 버전이 별도로 분리되어 있기 때문에 **버전 추적 및 관리**가 쉽지 않다는 점이다. 어느 게임 버전과 어느 데이터베이스 버전이 맞는 것인지 찾기 어려울 가능성이 있다. 또한, 버전이 여러 벌 존재함으로써 생기는 의사소통의 혼란 또한 무시할 수 없을 것이다.

◆ C-7-4. 데이터 타입 사용

- **C-7-4-1.** 어떤 항목에 대해 가변적인 크기와 개수, 종류를 담아야 할 경우, 일정한 규칙에 의해 작성하는 **JSON 텍스트 형식**으로 저장한다.
: 해석에 대한 책임은 전적으로 DB에서 이 데이터를 얻어오는 애플리케이션의 코드에게 위임하는 방식이다.

- **C-7-4-2.** 날짜와 시간은 **협정세계시(Coordinated Universal Time, UTC)로 저장**해야 한다.

※ DB서버의 운영체제에 설정된 지역 시간대를 이용해서 저장을 하게 되면, 나중에 DB 서버가 많아질수록 이를 관리하는 데 큰 노력을 소모해야 한다. 또한, 한 번 저장한 날짜와 시간 값을 고치기도 매우 어렵기 때문에, 애초에 UTC만 사용하도록 표준을 규정하는 게 더 낫다.
지역 시간대가 필요한 순간에만 변환을 해줄 수 있으니까... 이미 이와 관련한 라이브러리와 설비들도 표준화되어 제공이 되고 있다.

클라우드 플랫폼으로 서비스하는 경우가 늘고 있기 때문에, 서비스 대상 지역의 시간대와, DB 서버가 위치한 지역의 시간대가 일치한다는 보장도 없게 된 경우도 많다. 이러니 UTC 위주로 날짜와 시간을 다루는 건 불가피하다고 볼 수 있다.

◆ C-7-5. 데이터베이스 분산(Database Partitioning)

~~- **C-7-5-1.** 가급적 DB를 물리적으로 분할하지 않는 구조와 정책으로 설계한다.~~
~~: 자동 수평 분할(Auto Sharding)을 사용하기는 어려울 것 같기 때문이다.~~

~~※ 현재 주력 기반 DB로 사용하려고 하는 MySQL의 경우, MySQL Cluster와 NDB 엔진을 사용하지 않으면 DB의 자동 수평 분할을 할 수 있는 제품이 없다. MySQL Cluster는 유료 제품이라서 서비스에 적용할 가능성이 별로 없다. 또한 MySQL Cluster의 기능을 이용하려면 반드시 NDB 엔진만 사용해야 하기 때문에, 필요에 따라 기본 엔진인 InnoDB 같은 DB 엔진을 섞어서 쓸 수 없는 점도 있다.~~

※ 2016. 08. 24 삭제
- 말 자체는 틀리지 않으나, 현실적으로 게임 DB와 같은 경우에는, 하나의 DB 서버에서 모두 처리하기에는 성능 부담이 너무 크기에, 분할을 고려하지 않을 수가 없다.

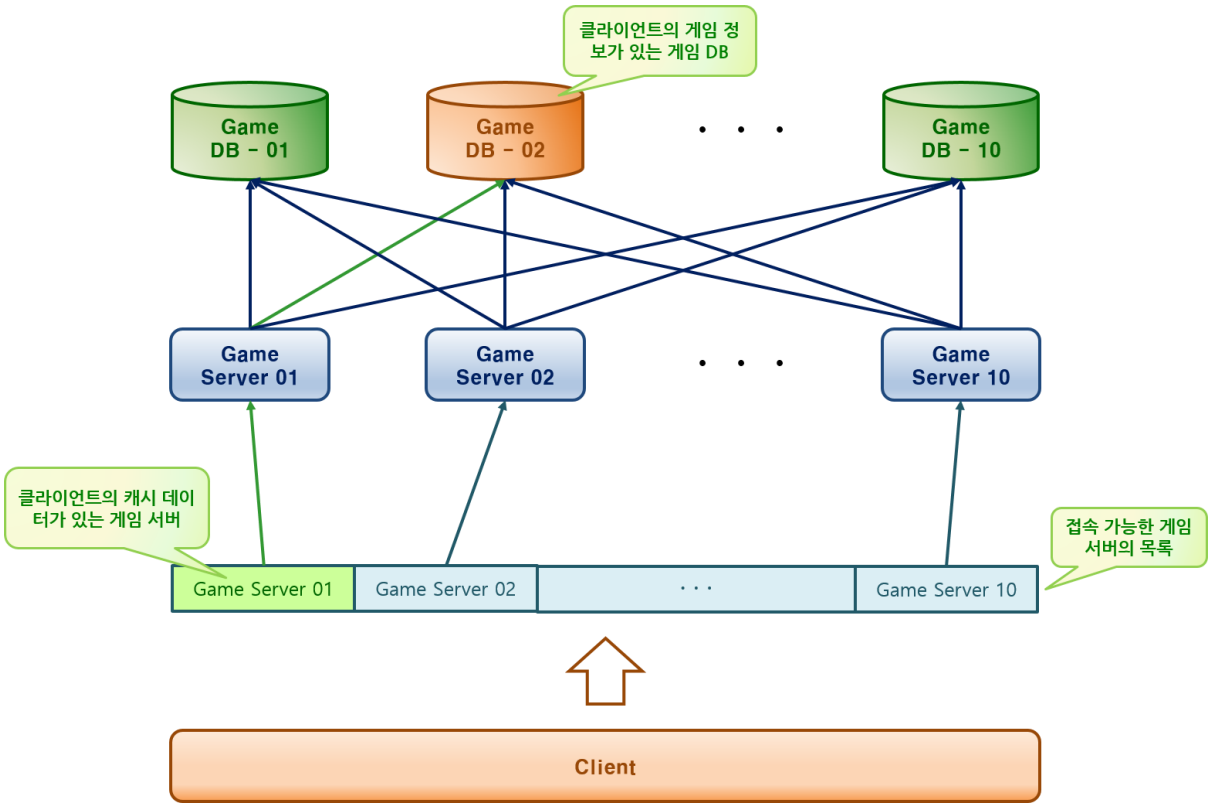
- **C-7-5-2.** 하나의 테이블 크기가 지나치게 커질 경우, 이를 경감하기 위한 테이블 분할은 가능하다.
: 대표적으로 기록이 몰릴만한 게 결제 기록, 대전 기록, 로그 기록 등이다. 과거 이력이 자주 필요하지 않으면서도, 지속적으로 쌓이는 데이터라는 특성을 지닌다.

- **C-7-5-3.** 관계형 DB를 쓰기 때문에, 분산은 가급적 게임 DB에만 집중한다.
: 인증 DB라든가 결제, 커뮤니티와 관련한 DB들은 분할하지 않는다.

※ 인증 DB 같은 것도 꼭 물리적으로 분산을 하지 못하라는 법은 없기는 한데, 분할을 하게 되면 이들 DB의 데이터에서 유일성(Unique)을 검증하기 위해 네트워크 비용(지연 시간, 트래픽

등)이 과다해지는 문제가 있을 것이다.
인증 DB나 결제 DB, 커뮤니티 DB와 같은 DB들은 데이터에 사용하는 ID, 이름 등에 대해 유일함을 검증해야 할 경우가 많기 때문에, 가급적 서버군 내에서 하나만 있는 게 좋다고 본다.

- **C-7-5-4.** 게임 DB의 분산은, 플레이어의 사용자 ID를 기준으로 이루어진다.



<사용자 ID에 따라, 어떤 게임 서버로부터 접근하더라도 접근하는 게임 DB는 고정적이다.>

◆ C-7-6. 복제(Replication) 전략

- **C-7-6-1.** Master – Slave 관계
: 이 관계는 전적으로 MySQL NDB Cluster 제품의 내부적인 분산 기능에 의존한다.

◆ C-7-7. 캐시 정책

- **C-7-7-1.** 데이터베이스의 내용을 메모리에 캐시 할 것인지 여부
: 각 DB 테이블에 대해 SELECT를 싹 다 돌리고 서비스에 투입할 것인지...

◆ C-7-8. 장애조치(Failover)와 복구 정책

- **C-7-8-1.** 각 데이터베이스들은 지정한 시간마다 백업을 생성한다.

- **C-7-8-2.** 클라우드 플랫폼에서 서비스하는 경우, 해당 플랫폼이 지원하는 장애복구 기능을 사용한다.
  : 단, 이런 서비스를 사용할 때는 추가 비용에 대해 고려해야 한다. 같은 서비스에 대해 2배의 기계를 투입하는 것이기 때문이다.

◆ C-7-9. 백업 정책

- **C-7-9-1.** 물리적으로 분리되어 있는 각각의 DB마다 각각 백업 DB를 둔다.

- **C-7-9-2.** Master – Slave 관계인 경우에도 백업 DB는 별도로 두어야 한다.

※ Slave DB는 Master DB의 내용을 즉각적으로 반영하기 때문에, 잘못된 데이터가 Master에 반영된다면 Slave DB도 결과적으로 똑같이 오염될 수 밖에 없다. 그래서 Slave DB를 백업 용도로 사용하지 못하는 것이다.

- **C-7-9-3.** Master – Slave 관계인 경우에는 Master DB를 기준으로 한 백업 DB만을 둔다.
  : Slave DB는 어차피 Master DB의 쌍둥이 복제판이기 때문에, 굳이 Slave DB까지 백업할 필요는 없다.

◆ C-7-10. 사용자 데이터 정보

- **C-7-10-1.** 서비스를 이용하는 개별 사용자들의 누적된 정보를 저장하기 위한 데이터 테이블들이 있어야 한다.

- **C-7-10-2.** 사용자 데이터를 적재하는 용도로 사용하는 데이터베이스 테이블들은, 테이블을 정의하는 이름의 맨 앞에 'table_' 접두어를 붙여서 이름을 짓는다.

◆ C-7-11. 게임 데이터 정보

- **C-7-11-1.** 서버에서 사용해야 하는 게임 데이터들을 데이터베이스 테이블로써 정의한다.

※ 사실 이 부분은 제한된 구조의 XML이나 CSV 등의 테이블 기반 데이터들을 서버에서 파일로 읽어 오는 방식으로 구현해도 된다.
다만, 이를 데이터베이스를 이용해서 정의하는 이유는, 서버 용 게임 데이터가 변경이 될 때마다 배포와 관리가 좀 더 용이할 거라는 판단 때문이다.
게임 서비스를 할 때 서버의 개수가 매우 많을 수 있는데, 이들 서버마다 일일이 변경된 데이터 파일을 배포하는 작업은 지루할 뿐만 아니라 오류가 발생하기도 쉽다. 그에 비해, 데이터베이스들은 상대적으로 물리적으로 개수가 적기 때문에 관리하기가 상대적으로 더 쉽다.

- **C-7-11-2.** 서버에서 게임 데이터로 사용하는 데이터베이스 테이블들은, 테이블을 정의하는 이름의 맨 앞에 'data_' 접두어를 붙여서 이름을 짓는다.

- **C-7-11-3.** 게임 데이터로 사용하는 데이터베이스 테이블('date_'로 시작하는 테이블)들은 사용자 데이터를 적재하는 용도로 사용하는 데이터베이스 테이블('table_'로 시작하는 테이블)은 **서로 외래 키 제약(Foreign Key Constraint)을 걸지 않는다.**

※ 이렇게 하는 이유는 'data_' 테이블들은 특성상 자주 구조가 변경될 수 있는 여지가 있기 때문이다. 그런 테이블에 외래 키 제약이 많이 걸리게 되면, 외래 키로 연결된 종속 테이블을 삭제하기 전까지는 테이블의 변경이나 삭제가 불가능하다. 그런데 데이터 용 테이블들은 그 내용이나 구조가 빈번히 변경될 수 있고, 그 때문에 자주 테이블을 제거했다가(DROP TABLE) 재등록(CREATE TABLE) 해야 하는 경우가 많을 수 있다.
이래서는 데이터 변경에 대해 제약이 너무 많고, 이를 회피해서 구현하는 건 매우 힘들다. 더구나 사용자 테이블 데이터와 외래 키 제약이 걸려 있은 경우에는 사용자 테이블의 해당 열(Column 데이터를 제거해야 하는 상황이 올 수도 있는데, 그런 건 절대로 받아들여질 수 없는 상황이다.
그렇기 때문에, 게임 데이터 테이블과 사용자 테이블끼리는 외래 키 제약 조건을 사용하지 않는다.

C-8. 게임 플레이 동기화

◆ C-8-1. 정책

- **C-8-1-1. 실시간으로 게임 플레이 동기화가 필요한 어떤 요소도 기획 단계에서부터 배제**한다.

※ 네트워크 동기화에 대한 이슈와, 재참여, 게임 진행 상황 복구에 대한 시스템과 네트워크 분야의 이슈가 필요하기 때문이다.
특히, 이 요소들은 구현을 위해서는 아예 게임의 기반 설계부터 다르게 만들어야 할 수 있는 요소들이다.
하나하나가 대단히 구현 비용이 큰 이슈들이기 때문에, 위 기능을 나중에라도 추가하고자 하는 경우에는 프로젝트 전체의 설계 및 구현 비용 자체를 다시 추정해야 한다.

※ 매우 유감스러운 점은 멀티 유저 플레이를 위한 실시간 동기화에 필요한 비용이나 설계 / 구현 난이도가 높음에 비해, 이를 실제로 활용했을 때의 이용 빈도나 매출액에 대한 공헌도는 그다지 크지 않다는 점이다.

우선 모바일 기기는 네트워크 패킷 송수신 빈도가 PC 등의 기기에 비해 훨씬 떨어지며, 연결 지연 / 단락 횟수도 훨씬 잦다. 이렇게 상대적으로 PC에 비해 열악한 네트워크 현실을 보완하기 위한 방책을 충실히 구현하기는 쉬운 일이 아니다.
더구나, 실시간 동기화는 웹 소켓으로 처리하기 어려운 문제이기 때문에 별도의 서버를 제작하고 이를 항상 유지해야 한다. 당연히, 웹 서버 한 가지로 네트워크 관련 처리를 몽땅 담당하는 방식에 비해 훨씬 구현 비용이나 유지 비용이 비쌀 수 밖에 없다.

그리고 경험상, 게임에서 싱글 플레이와 네트워크 멀티 플레이를 둘 다 지원한다고 했을 때, 이용 빈도는 싱글 플레이와 네트워크 멀티 플레이가 8 : 2 이상의 편중을 보일 정도로 네트워크 멀티 플레이는 인기가 별로 없다. 무엇보다도, 아직 모바일 네트워크 환경이 PC 환경에 준할 만큼 쾌적하지가 않기 때문이다. (전 세계에서 가장 네트워크 기기들이 밀집해 있는 대한민국에서조차도 현실이 이러하다.)
결국, 싱글 플레이에서 수익을 창출하기가 어려운 기획 방식이라면 모를까, 과금 체계가 싱글 플레이와 네트워크 멀티 플레이에서 차이가 별로 없다면, 네트워크 멀티 플레이를 완벽하게 지원하기 위한 비용들의 크기에 비해 대부분 얻는 결과가 초라하다.

◆ C-8-2. 동기화 개념과 구조

~~- C-8-2-1. 결정론적 업데이트(Deterministic Update)~~
~~: 앞으로 할 행동 단위의 연산 결과가 사전에 결정적으로 작동해서, 각 응용 프로그램이 같은 순서의 같은 입력을 넣는다면, 출력 결과가 동일함을 보장한다.~~

~~- C-8-2-2. 게임 상에서는, 플레이어의 입력이 같은 시간에 같은 순서로 들어간다면, 그 결과로 진행되는 게임의 양상은 항상 동일한 진행을 보이게 된다.~~

~~※ 긴 말 필요 없이 스타크래프트 등의 RTS에서 쓰이는 다시 보기(Replay) 시스템을 떠올리면 된다. 이 기능들은 플레이어가 진행했던 게임에서 플레이어의 입력 순서와 타이밍을 그대로 흉내를 낸다.~~
~~비록 플레이어가 실제로 게임을 플레이 한 현실 시간과, 다시 보기를 재생한 현실 시간은 같지 않지만, 다시 보기 기능은 플레이어의 입력 순서와 (게임이 시작된 뒤부터 센)입력 시간만 가지고서 원래의 게임을 그대로 재현해낸다.~~
~~이는, 게임 내부의 프로그램 구현이, 사용자 입력(직접 사용자가 입력하든, 프로그램이 흉내 내서 입력하든)이 상대적으로 같은 시간에 같은 순서로 들어오면, 같은 결과를 내도록 미리 프로그램 되어 있기 때문이다.~~

~~- C-8-2-3. 네트워크 게임의 입장에서는, 서버에서는 각 클라이언트의 게임 자산들의 상태와 위치를 매번 갱신하지 않더라도, 각 클라이언트들의 입력을 동기화하는 것만으로도 게임 내용이 자동으로 동기화된다.~~
~~: 각 클라이언트가 동일한 타이밍과 순서로 입력을 받아들이도록 보장해준다면, 각 클라이언트들은 각자 연산을 하더라도, 각 연산 결과와 그 순서들이 동일하므로, 게임 진행 결과가 똑같아진다.~~

※ 2017. 02. 14 삭제
지형 맵이 아주 넓은 RPG에서 이런 방식의 동기화 개념은 사용하기 어렵다.

◆ C-8-3. 공통 구현 사항

- C-8-3-1. 해당 사항 없음

◆ C-8-4. 서버 측 시스템

- C-8-4-1. 해당 사항 없음

◆ C-8-5. 클라이언트 측 시스템

- **C-8-5-1.** 고정 업데이트 단계 구현
  : 1초 당 특정한 횟수만큼만 업데이트하는 기능

- **C-8-5-2.** 각 업데이트 단계를 게임 플레이 장면에서의 모든 객체가 판정 단계에서 연동하게 하는 기능
  : 게임 플레이 장면에서 모든 객체들이 고정 업데이트 단계에 연동할 수 있어야 각 클라이언트의 기기 성능이나 현재 부하 상황에 따라 연산 결과가 달라지지 않음을 보증할 수 있다.

- **C-8-5-3.** 고정 업데이트 단계가 정지할 때, 게임 플레이 장면에서 연동된 객체들의 판정 업데이트가 같이 정지하는 기능
  : 업데이트 단계는 일종의 가상 턴처럼 동작하는데, 이 턴이 진행되지 않는다면, 게임 내 판정 기능이나 내용 진행 역시 고정 업데이트 단계가 다시 시작할 때까지 정지해야 한다.

- **C-8-5-4.** 고정 업데이트 단계는 렌더링 업데이트와는 별개로 작동해야 한다.
  : 고정 업데이트 단계는 판정에 관련한 시스템이고, 실제 기기 시간이 필요한 렌더링이나 애니메이션 내부 동작과는 관련이 없다.

※ 렌더링이나 애니메이션은 기기의 성능이 좋을수록 더욱 '부드럽게' 보여야 한다.
그러나 고정 업데이트 단계의 각 단계별 수행 시간은 기기의 각 1프레임 수행 속도보다 일반적으로 훨씬 길게 잡는 게 보통이다. 그런 시간을 기준으로 애니메이션이나 렌더링과 관련된 작업에 적용을 하면, 뚝뚝 끊기듯이 보이게 된다.

C-9. 게임 플레이 데이터 검증

◆ C-9-1. 정책 및 제한사항

- **C-9-1-1.** 스테이지 내부에서만 유효한 정보들은 서버가 검증하지 않는다.
  : 오직 저장해서 유지해야 하는 정보와 관련된 것들만 서버가 검증한다.

- **C-9-1-2.** 서버 측에서는 클라이언트의 검증 결과를 어떤 형태로든 직접 이용하지 않는다.

◆ C-9-2. 스테이지 참여 검증

- **C-9-2-1.** 각 스테이지들은 스테이지의 시작 시간을 저장하고, 스테이지 완료 요청 시점에 시작 시간의 유효성을 점검한다.

- **C-9-2-2.** 각 스테이지들은 스테이지 시작 시점에서 인증 번호를 생성하고, 완료 요청 시점에서 인증 번호를 검사하여 이를 통과해야만 결과 및 보상 처리를 해준다.
  : 이를 통해, 실제로 입장을 하지도 않았음에도 불구하고 스테이지 보상 요청 프로토콜만 요청하여 무한정 보상을 받을 수도 있는 상황을 방지한다.

◆ C-9-3. 전투 판정 검증

- **C-9-3-1.** 게임 서버는 전투 판정에 대해서 검증하지 않는다.

- **C-9-3-2.** 클라이언트에서는 각 판정에 사용하는 변수 값들에 대해 검증하는 기능을 구현해야 한다.

- **C-9-3-3.** 하지만, 클라이언트에서의 검증과 관계없이, 서버 측에서는 클라이언트의 검증 결과를 어떤 형태로든 직접 이용하지 않는다.

◆ C-9-4. 미션 / 퀘스트 완료 여부의 검증

- **C-9-4-1.**

◆ C-9-5. 업적 달성 여부의 검증

- **C-9-5-1.** 업적 달성 여부를 검증하기 위해서, 업적 자체는 서버에서 검증할 수 있는 요소들만을 포함해서 만든다.
  : 클라이언트의 내부 계산으로 밖에 검증할 수 없는 요소들은 업적으로 만들지 않는다.

- **C-9-5-2.** 클라이언트의 내부 계산으로만 검증할 수 밖에 없는 요소들을 사용해야 한다면, 그 업적의 보상은 비즈니스 모델에 타격을 주지 않도록 제한해야 한다.

C-10. 리소스 제작

◆ C-10-1. 3D 개체 모델의 리소스 정책

- **C-10-1-1.** 모델 데이터의 뼈대(Bone) 개수 제한
: 모든 캐릭터 모델들은, 인간의 기본 형태보다 더 많은 뼈대 개수를 사용하지 않도록 한다.
영웅 캐릭터인 경우는 예외적으로 일부 장식적 요소에 한정하여 추가적인 뼈대를 사용할 수도 있으나, 플레이어가 조종할 가능성이 없는 다른 NPC 모델들은 아무리 많아도 인간 형태를 표현하기 위한 뼈대 개수를 넘어서면 안 된다.

※ Rigging 개수의 여부도 성능에 큰 영향을 미친다고 한다.
모델 리소스를 제작할 때, 인간형 Biped의 기본 Rigging 개수를 넘지 않는 편이 좋다고 한다.

- **C-10-1-2.** 모델 데이터의 뼈대 가중치 개수 제한
: 뼈대 가중치(Bone Weight)의 단계는 Unity3D 엔진에서 권장하는 수치인 **2단계**까지로 한다.

※ Unity 엔진에서는 뼈대 가중치가 최대 4 단계까지만 적용할 수 있다. Unity 엔진의 공식 문서에 따르면, 모바일 장치에서는 2 단계까지 적용하는 것이 시각적으로나 성능상으로나 가장 적절한 타협 지점이라고 한다.

- **C-10-1-3.** LOD(Level of Detail)를 적용하지 않는다.
: Mesh LOD와 Bone LOD 둘 다 그렇다.

※ 게임 디자인이, 카메라의 높이 각도를 조절할 수 있어서 멀리 바라보거나, 화면을 확대 / 축소 하지 않기 때문에, LOD를 적극적으로 적용하는 게 그렇게 큰 이득을 주지는 못한다.
(이는 Mipmap을 적용하는 데에도 영향을 주는 이슈이다.)
어차피 가까이 있는 캐릭터들은 원래대로의 LOD를 가져야 하고, 화면 밖에 있는 렌더링 개체들은 LOD 적용 이전에 카메라의 Frustum Culling에 의해 그리지 않게 되므로, LOD를 적용하는 의미가 퇴색한다.

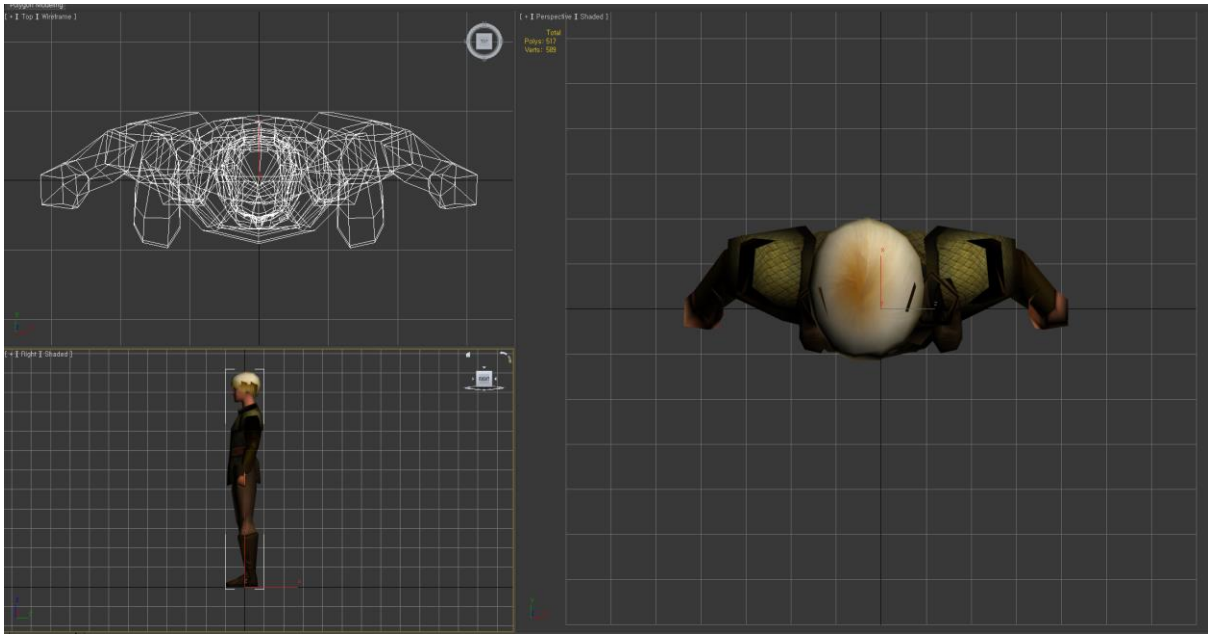
- **C-10-1-4.** 배율(Scale) 제한
: 게임을 실행할 때, 특별히 그래야 하는 경우가 아니라면, **모든 모델은 FBX 파일을 불러들일 때 사용한 Scale Factor에서 상대적으로 1.0 크기를 유지**한다.

※ Unity 엔진에서, 모델의 스케일은 FBXImporter의 Scale Factor 항목을 통해서 조절하지 않고

Transform에서 조절하는 경우, 동적 배칭(Dynamic Batching)이 동작하지 않을 수 있다. 모델의 배율 값이 서로 다른 경우, 동적 배칭이 일어나지 않기 때문이다.
동적 배칭 역시 성능 향상에 도움을 주는 기능이기 때문에, 이게 안 되는 상황도 역시 피하는 것이 좋다.

- **C-10-1-5.** 중심축 기준
: 모든 모델의 **회전 중심축은 그 모델 단면의 (가로 중앙, 세로 중앙, 높이 0)에 위치**해야 한다.
그래야 회전할 때, 자연스럽게 회전하는 것처럼 보이기 때문이다.

※ 회전의 중심이 모델의 한가운데가 아니라, 한쪽으로 치우쳐 있을 경우에는, 게임에 올려놓고 회전시키면 마치 태양을 도는 지구처럼 보이지 않는 어떤 중심을 두고 빙빙 둘러서 회전하는 것처럼 보이게 된다.



<회전 중심축은 캐릭터의 (가로 중앙, 세로 중앙, 높이 0)에 놓여야 한다.>

◆ C-10-2. 객체 구조 정책

- **C-10-2-1.** 게임에서 불러오는 모델 중에, '땅 위에 세워야 하는' 모델들은 중심축을 바닥 부근에 놓는다.

※ 사실, 중심점을 모델의 완전한 정 중앙에 놓는다고 해도 상관없다. 그렇지만, 대표적으로 플레이어가 조종하는 캐릭터처럼 땅 위에 늘 서 있다고 가정하는 모델들의 경우에는 중심점 높이가 발바닥으로 잡혀 있는 게 더 다루기가 쉽다. 그렇게 만들면, 게임에서 모델을 불러오면 일단 (회전은 차치하고)땅 위에 서 있는 것처럼 불러올 수 있기 때문이다.

물론, 몸의 중심점을 찾아야 하는 경우도 발생하지만, 일반적으로 땅 위에 세우는 물체들은 캐릭터 모델들이고, 캐릭터들은 대개 어떤 이유로든 (보통 캡슐 모양의)충돌체를 하나씩 가지고 있다.

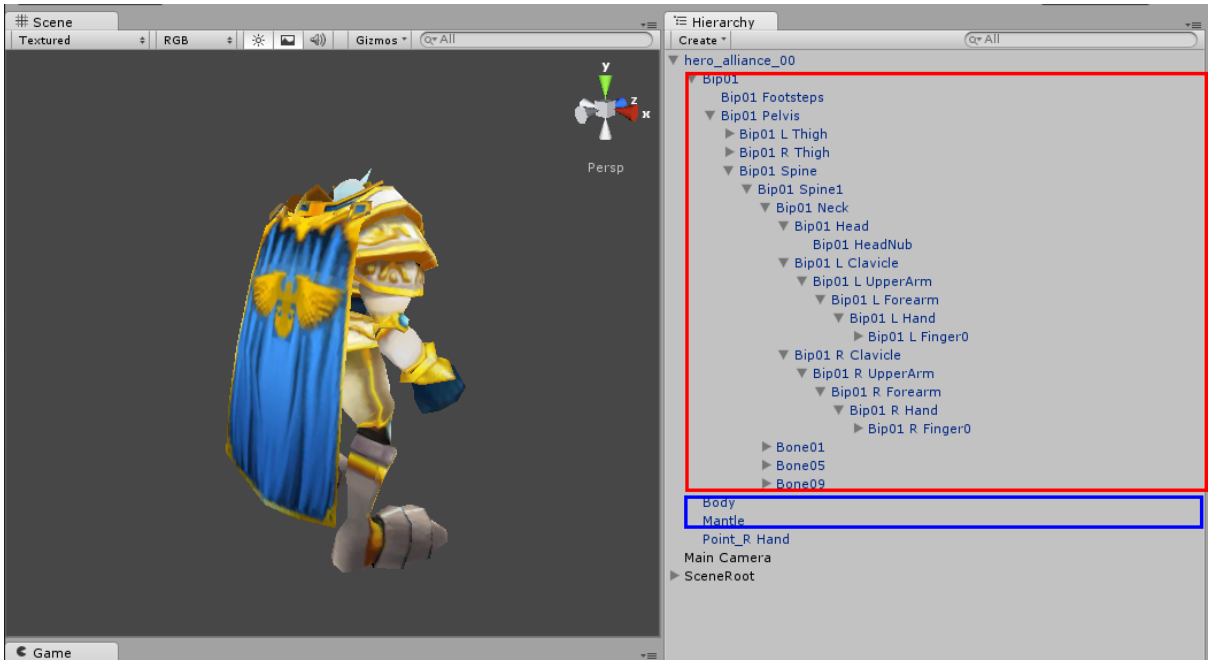
몸의 정 중앙을 찾고 싶은 경우, 충돌체의 중심을 찾아도 된다는 의미다.

물론, 아예 처음부터 모델의 정 중앙에 포인트나 더미 객체를 삽입해놓는 것도 방법이다.

- **C-10-2-2.** 애니메이션이 들어가는 3D 모델들을 제작할 경우, 뼈대와 Skinning Mesh가 별도의 계층으로 구분되도록 구성해야 한다.

: 이렇게 해야 게임 논리를 프로그래밍할 때, 게임 논리적으로 구분하는 각 파츠를 손쉽게 관리할 수 있다.

파츠 시스템에 대해서는 **[C-15. 파츠 시스템]** 을 참조할 것.



<Unity3D에 적용된, 뼈대와 Skinning Mesh가 분리되어 있는 모델 객체 구조>

※ 위 그림에서 보는 것처럼, 붉은 색 사각형 안에 있는 객체는 뼈대 객체들 뿐이고, 파란색 사각형에 포함된 객체들은 실제 눈에 보이게 될 Skinning Mesh들이다.

보다시피 뼈대 객체는 인간형일 경우, 일반적으로 3DS Max에서 Biped를 통해 구성하는데, 이 구성 요소의 개수나 객체 깊이가 꽤나 많고 깊다. 만일 Skinning Mesh들이 이러한 뼈대 객체의 각 구성 요소 안에 흩어져 있는 경우를 생각해보자. 이럴 경우에는 게임 진행 중에 특정한 Skinning Mesh를 찾아서 어떤 효과를 가하기 어려운 상태가 된다.

게임 논리를 프로그래밍할 때, 캐릭터 몸 전체를 열 받았다는 표시로 빨갛게 채색하거나, 잠깐 투명 인간이 되었다고 해서 일시적으로 눈에서 안 보이게 해야 한다거나, 팬티를 입히거나 벗길 수 있게 해야 할 경우 등을 표현하는 유사한 모든 행위는 원하는 특정한 Skinning Mesh를 찾아서 조작해야 한다.

그러나 **Skinning Mesh들이 뼈대 객체 속에 흩어진 경우, 이를 일반적으로 찾아내기는 매우 어려워**진다. 아주 불가능한 것은 아니지만, 사전에 약속된 이름을 짓거나, 프로그래밍 적으로 더 느릴 수 밖에 없는 깊은 뼈대 객체의 구조를 일일이 탐색해야 하는 등의 부작용이 있다.
반면 위 그림처럼 **Skinning Mesh 객체들이 뼈대 객체들의 집합과 아예 구분되도록 제작한 경우**에는, 프로그래밍 과정에서 아주 손쉽게 Skinning Mesh만 골라낼 수 있게 된다.

◆ C-10-3. Mesh 분류 정책

- **C-10-3-1.** 모든 모델은 이하에 열거되는 예외 사항을 제외하고, 가급적 Skinning Mesh를 한 개만 사용하여 표현해야 한다.

※ Unity 공식 문서의 최적화 관련 부분에서 이와 같은 내용을 찾을 수 있다.
Skinning Mesh가 1개일 때와 여러 개일 때의 성능 차이가 가장 두드러지고, 일단 1개를 넘어서게 될 경우, 개수가 많아지거나 적어짐에 따른 성능 격차가 별로 없다고 한다.
애니메이션 모델 하나가 Skinning Mesh를 2개를 쓰든 5개를 쓰든 성능 격차는 별 차이가 없지만, 1개일 때와 2개일 때는 성능 격차가 많이 난다는 뜻이다.

- **C-10-3-2.** **모델이** 정적 **/ 신체형 파츠를 사용해야 하는 경우**, 구분이 필요한 Mesh들을 분리해서 표현할 수 있다.
: **[C-15-3. 파츠의 한계 및 적합성]** 내용 참조.

※ 무기의 3D 모델이 필요에 따라 검 모델에서 활 모델로 바뀌는 모습이 보여야 하거나, 가죽 갑옷을 미니스커트로 갈아 입혀야 하는 경우를 들 수 있다. 실제로 게임에서 '아이템을 바꾼다.'는 데 대한 일반적인 이미지는 이렇게 파츠의 Skinning Mesh가 교체되는 모습이라고 할 수 있다.

- **C-10-3-3.** Occlusion Culling을 사용하고 있고, 모든 Mesh가 한 화면에 다 표시되지 않는 경우에도 Mesh를 분리해서 표현할 수 있다.
: 이 경우는 애니메이션 모델에 붙는 Skinning Mesh만 해당하는 것은 아니다. 항상 고정된 위치에 존재하는 Mesh(Static Mesh)들도 모두 해당한다.

※ 가장 대표적인 경우는 지형 맵을 표현하는 경우다. 지형 맵은 일반적으로 한 화면에 모든 Mesh를 다 그릴 수 없다. 그래서 카메라에 보이는 Mesh만 그리고, 나머지는 그리지 않는 Occlusion Culling을 사용하기에 적합하다. (특히 카메라가 원경(遠景)을 보도록 각도가 설정되어 있는 경우는 더더욱 그렇다.)
자세한 내용은 **[D-4. 지형]** 내용 참조할 것.

<Occlusion Culling을 사용하는 경우, 지형 Mesh는 분리하는 게 좋다.>

◆ C-10-4. 재질 정책

- **C-10-4-1.** 하나의 모델이 여러 개의 재질을 사용해야 하는 경우를 줄인다.
  : 가급적 캐릭터 모델에 대해서만 사용해야 하며, 개체 수가 많은 졸병 모델들은 **1개의 개체 모델 = 1개의 재질 사용**의 원칙을 준수하는 것이 성능에 좋다.

※ 여러 개의 재질을 사용하는 경우는 주로 Unity 엔진의 **Draw Call** 횟수 부분과 관련이 깊다.

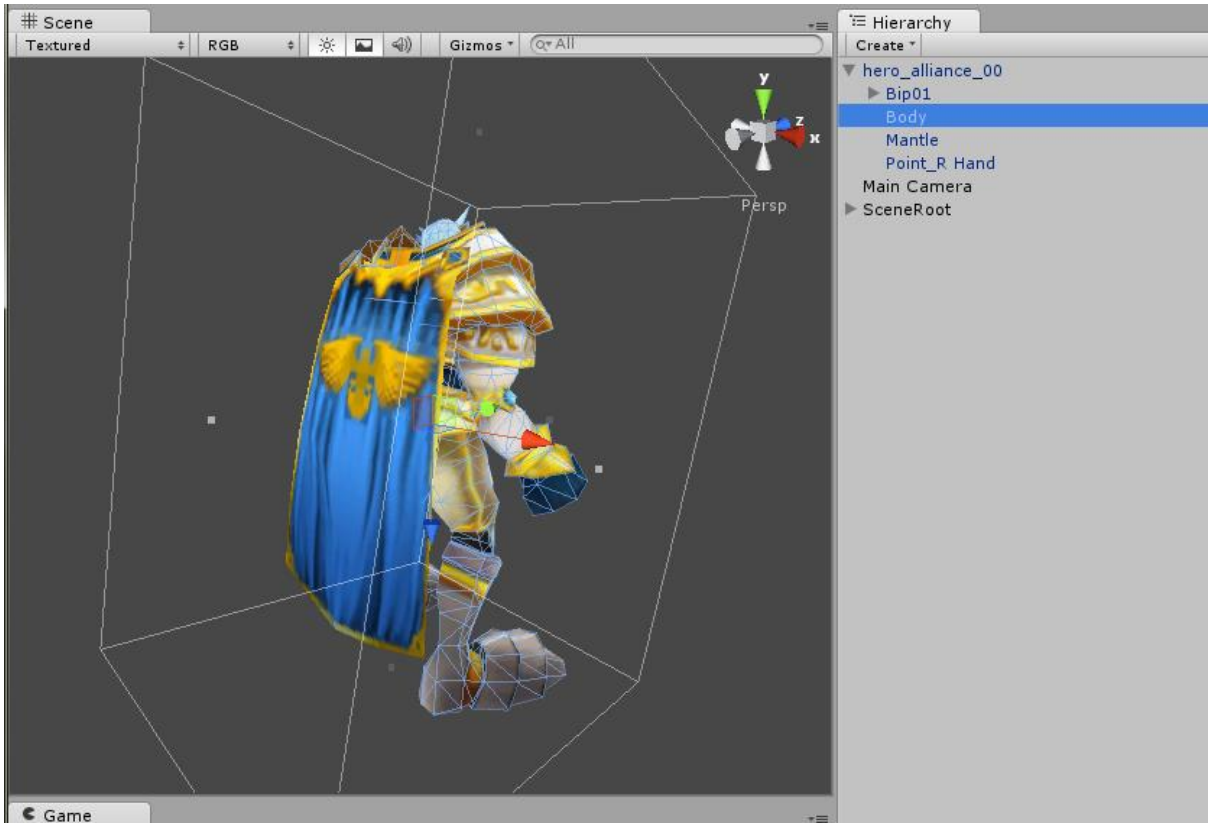
- **C-10-4-2.** 동적으로 불러오는 개체를 표현하는 모델들은 광원을 받지 않는 재질을 적용해야 한다.
  : 그 이유는 Battlefield Heroes 게임에는 **실시간 광원 계산을 적용하지 않을 것이기 때문**이다.
  (**[C-7-1. 조명 처리 정책]** 내용 참조.)

※ 실시간 광원을 이용하는 Shader를 적용하고, 광원을 하나도 사용하지 않는 경우, Unity 상에서는 모든 사물이 매우 어둡게 보인다.
이를 회피하기 위해 Unity Editor에 전역으로 존재하는 환경광(Ambient Light)의 밝기를 조절하는 것도 가능하지만, 조명 맵(Light Map)을 제작할 때 광원의 세기에도 영향을 미치기 때문에, 환경광을 이용해 전체적으로 밝게 만드는 방식을 사용하는 것 자체가 어려울 수도 있다.
그렇기 때문에, 조명 계산을 하지 않는 Shader를 사용하는 방식을 우선 고려한다.

- **C-10-4-3.** 특정 Mesh 부분만 별도의 재질로 표현해야 하는 경우, 전체 모델의 Mesh 객체는 하나로 합치되, 해당 Mesh 부분만 재질을 따로 준다.

※ Unity3D에서는 **하나의 재질에 여러 재질 속성이 들어가지 못한다. Unity에서의 재질은 텍스처 1장과 연결**되어 있기 때문에, 다른 재질로 사용하고 싶으면, 재질 별로 텍스처는 따로 파일을 분리해서 제작해줘야 한다.

※ Mesh의 부분 별로 다른 재질을 주고 싶은 경우, 재질 별로 텍스처를 나누면 된다. 텍스처만 나뉘어 있다면, 1개의 Mesh에서 여러 개의 재질을 사용하는 것은 가능하다.
재질을 여러 개 사용하는 것도 성능에 악영향을 미치지만, Mesh 분리 + 재질 분리로 쌍으로 작용하는 경우는 더더욱 그렇다.



<망토의 경우, 앞 / 뒤를 모두 그리게 해야 해서, 망토만 별도의 재질을 적용했다.>

◆ C-10-5. 3D 월드에서의 크기 관련

- **C-10-5-1.** Unity 3D 월드에서의 **1.0 = 1미터(m)의 공식으로 기준**으로 삼는다.

◆ C-10-6. 텍스처 관련

- **C-10-6-1.** 텍스처 사이즈 제한
: 각 오브젝트 종류에 따른 텍스처 사이즈의 제한은 다음과 같다.

- **지형 맵의 바닥 텍스처 사이즈는 장당 최대 2048 × 2048 픽셀 이내**이다.
- **플레이어 캐릭터에게 사용하는 텍스처 사이즈는 최대 256 × 256 픽셀 이내**이다.
- **비 플레이어 캐릭터(NPC)에게 사용하는 텍스처 사이즈는 최대 128 × 128 픽셀 이내**이다.
- **지형에 사용하는 텍스처의 사이즈는 최대 128 × 128 픽셀 이내이다.**
- **시각 이펙트에 사용하는 텍스처의 사이즈는 64 × 64 픽셀 이내이다.**

- **C-10-6-2.** **텍스처** 포맷은 다음과 같다.

• iOS의 경우
: **PVRTC 포맷을 사용**한다. iOS에서는 포맷이 가장 효율이 좋다. (속도 + 용량)
옵션은 **Compressed 2Bit + Normal의 사용을 권장**하나, 품질이 너무 떨어져 보일 경우, Compressed 4Bit + Best도 허용할 수 있다.

• Android의 경우
: 기본 포맷은 **ETC1**이다. OpenGL ES 2.0 이상 지원하는 기기에서는 가장 효율이 좋다.
다만, 이 포맷은 RGB 포맷이기 때문에, Alpha 값이 들어간 텍스처에 사용할 수는 없다. **투명도가 있는 텍스처는 RGBA 16Bit 포맷**이 기본이다.
옵션은 Compressed ETC1 또는 RGBA 16이며, **압축 품질은 Normal을 우선**으로 하되, 너무 품질이 나빠 보일 경우 Best도 가능하다.

※ 현재 iOS에서의 성능이 특히 나쁘기 때문에, iOS에 최적화된 **PVRT Compress**를 아예 텍스처 원본부터 적용시키는 연구도 진행 중이다. (2012.년 7월)

- **C-10-6-3.** 텍스처의 압축 방식 및 품질이 반드시 균일해야 할 의무는 없다.
: 더 작고 눈에 띄지 않는 텍스처는 더 저품질의 옵션으로, 더 크고 중요한 텍스처는 고품질의 옵션으로 주는 방식도 허용한다.

- **C-10-6-4.** 모든 텍스처는 Mipmap을 사용하지 않는다.

※ 렌더링할 텍스처에 Mipmap을 생성하게 되면, 생성하지 않는 경우에 비해 33% 정도의 메모리를 더 차지한다. 밉맵은, 렌더링되는 객체에 대해 확대 축소가 빈번하고 광활한 시야를 가지는 게임인 경우에 성능 향상을 극대화할 수 있다.
하지만 이 게임은 기본적으로 아주 먼 방향을 바라볼 필요가 없도록 제작되어 있고, 카메라와 객체 간의 거리 역시 큰 변화는 거의 없다. 따라서 밉맵을 사용해서 얻을 수 있는 이득은 미미

한 반면, 추가적인 축소 텍스처 생성으로 인한 메모리 부담은 그대로이기 때문에 손해가 훨씬 크다.
이 게임에서는 밉맵 기능을 필요로 하는 상황이 거의 없으리라 본다.

- **C-10-6-5.** 가급적 텍스처 아틀라스(Texture Atlas)를 사용한다.
: 여러 텍스처를 모아서 하나의 큰 텍스처로 만들고, 이를 통해 여러 Mesh가 하나의 재질을 공유하게 될 경우, **Draw Call을 획기적으로 줄여서 성능을 높여주는 효과**가 있다.

※ 단, 주의할 점이 있다.
텍스처 아틀라스가 성립하려면, 텍스처 아틀라스에 있는 텍스처로 만드는 재질의 Shader는 동일해야 한다. 왜냐하면, 같은 파일에 있는 텍스처의 어떤 부분은 A라는 Shader로, 저쪽 부분은 B라는 Shader로 재질을 사용할 수는 없기 때문이다.
따라서 **사용할 재질의 Shader 별로 구분해서 텍스처 아틀라스를 만들어줘야 한다.**

- **C-10-6-6.** 투명도를 가진 텍스처의 사용 제한
: 투명도를 가지는 텍스처는 RGB 값만 가지는 텍스처보다 용량이 크다. (당연한 사실...)
따라서, 투명도가 굳이 필요하지 않은 경우, 텍스처는 투명도가 없는 포맷으로 제작하는 것이 좋다.

※ Android의 OpenGL ES 2.0에 최적화된 ETC1은 투명도를 지원하지 않는 텍스처 형식이다.
iOS에 최적화된 PVRTC 포맷은 투명도를 지원하지만, 투명도를 실제로 사용하는 텍스처를 그릴 때, 성능이 유의미할 정도로 더 많이 하락하는 것을 확인하였다.

iOS4를 기준으로 RGB 색상만 있는 텍스처 및 재질을 사용할 때는 90K의 정점을 30 FPS로 구동할 수 있으나, RGBA 색상의 텍스처 및 투명도를 표현하는 재질을 사용하는 경우에는 40K까지만 30FPS로 동작시킬 수 있었다.

C-11. 컨텐츠 데이터

◆ C-11-1. 다운로드 가능한 컨텐츠 데이터

- **C-11-1-1.** 게임 컨텐츠는 사전에 실행 파일 패키지에 탑재하는 컨텐츠 요소와, 다운로드해서 설치하는 컨텐츠 요소로 나눈다.

- **C-11-1-2.** 실행 파일 패키지에 탑재하는 컨텐츠 요소는 게임의 최소 실행부터 컨텐츠를 다운로드할 때 필요한 사용자 인터페이스를 표시하는 컨텐츠까지이다.

- **C-11-1-3.** 그 이후의 게임 진행 내용에 대한 리소스 데이터들은, 게임 서비스에서 규정하는 최신 버전의 리소스 데이터를 네트워크로 내려 받아서 적용하는 방식으로 구성한다.

◆ C-11-2. 컨텐츠 데이터의 업데이트 방식

- **C-11-2-1.** 게임이 처음 실행될 때, 컨텐츠 데이터의 버전이 현재의 게임 서비스에 적합한 버전인지 점검해야 한다.

- **C-11-2-2.** 현재 게임 컨텐츠가 최신 버전이 아닌 경우, 최신 컨텐츠에 대한 리소스를 내려 받기 위한 안내를 해야 한다.

- **C-11-2-3.** 필요하다면, 갱신한 컨텐츠를 올바르게 적용하기 위해서 게임 실행의 과정을 처음부터 다시 진행하는 것도 가능하다.
  : 주의할 점은, 실행 과정을 첫 단계부터 다시 밟는 것이지, 응용 프로그램 자체를 완전히 종료했다가 다시 시작하는 게 아니라는 점이다.

※ 특히, 자동으로 특정 응용 프로그램을 자동 실행하는 행위는 경우에 따라 아예 불가능하게 되어 있거나, 마켓 정책상 허가하지 않을 수 있다.

- **C-11-2-4.** **응용 프로그램 자체의 버전 점검과, 리소스 컨텐츠에 대한 버전 점검은 분리해서 점검**을 해야 하고, **반드시 응용 프로그램 자체의 버전 점검을 먼저 해야** 한다.

※ 대부분의 대형 게임들의 경우, 응용 프로그램 패키지의 용량에 비해 리소스 컨텐츠의 데이터

용량이 압도적으로 크다.
만일 버전 점검이 분리되지 않은 경우에는, '구 버전 응용 프로그램의 최신 리소스 컨텐츠'들을 수백 MB 다운로드 받은 뒤, '최신 응용 프로그램으로 교체하라'는 안내를 받는 상황이 만들어질 수 있다.
이렇게 되면 많은 용량의 리소스 데이터를 받느라 들인 시간(심지어 회선 비용)은 모두 헛수고가 되기 때문에 사용자에게 대단한 불쾌감을 준다. (사실 이건 매우 순화한 표현이다. 당해 보면 욕이 저절로 나온다.)

- **C-11-2-5.** 응용 프로그램의 버전 점검 과정에서, 새로운 응용 프로그램을 내려 받을 수 있는 URL을 제공하는 인터페이스를 내장해야 한다.
: 특히, 이 인터페이스는 가급적 최초에 공개된 인터페이스에서 URL 데이터 외에는 변경하지 않을 수 있도록 주의 깊게 설계해야 한다.

※ 버전을 점검하는 인터페이스 그 자체가 새로운 버전이 나올 때마다 업그레이드해야 한다면, 이를 효과적으로 알리기란 상당히 난해한 문제다.
가급적 버전을 점검하고, 새로운 버전을 안내하는 인터페이스 구조는 그 구조 자체를 변경하는 일을 최소화할수록 좋다.
물론, 서비스가 장기간 지속되면 불가피하게 버전 업데이트 인터페이스도 바뀌어야 할 경우가 생기는데, 그래도 응용 프로그램의 버전 점검 인터페이스만은 처음 상태를 유지할 수 있다면, 버전 변경으로 인한 부정적인 여파를 최소화할 수 있을 것이다.

◆ C-11-3. 리소스 데이터 불러오기 방식

- **C-11-3-1.** 로딩할 때, 화면을 GUI로 가리는 경우, 진행 상황을 사용자에게 알릴 수 있게 표시해야 한다.
: 현재 진행률이 어느 정도 되는지 알 수 있으면 좋고, 그게 아니더라도, 뭔가 로딩을 하고 있다는 사실을 알 수 있으면 된다.

- **C-11-3-2.** 응용 프로그램을 시작했을 때 불러들여야 하는 리소스의 종류. 이 리소스들은 (거의 대부분) 응용 프로그램이 종료되기 직전까지 가지고 있어야 하는 데이터들이다.
: 나중에 추가함.

- **C-11-3-3.** 그 외 게임 플레이 장면에 필요한 리소스 데이터들(특히 배경, 모델 데이터 등을 말한다.)은 **해당 스테이지가 시작하기 직전에, 해당 스테이지와 관련된 데이터들만 모두 불러들인다**.

◆ C-11-4. 리소스 데이터 로딩 구조

- **C-11-4-1.** **순차적인 로딩(Sequence Loading)**
: 로딩의 순서가 정해져 있고, 완료 순서는 로딩을 호출한 순서를 준수한다.

- **C-11-4-2.** **비동기적 로딩(Asynchronous Loading)**
: 로딩에 대해 순차성을 가지지 않는다. 경우에 따라, 먼저 로딩을 시도했더라도, 로딩이 끝나는 순서는 다를 수 있다.

- **C-11-4-3.** 순차적인 로딩 방식에서는 한 번 로딩 과정이 시작되면, 중간에 새로운 로딩 과업을 끼워 넣을 수 없다.
: 현재 순차 로딩 목록이 모두 작업이 끝난 뒤, 새로운 순차 로딩의 목록에 넣는 식으로 추가해야 한다.

- **C-11-4-4.** 비동기적 로딩 방식에서는 새로운 로딩 과업을 로딩 중에도 집어넣을 수 있다.
: 그게 가능해야 비동기적 로딩이 의미가 있다.

◆ C-11-5. 프리팹(Prefab) 정책

- **C-11-5-1.** 게임에서 사용하는 의미 있는 객체들과 컴포넌트들의 조합은 프리팹(Prefab)으로 구성해서, 게임을 시작할 때 동적으로 불러오게 한다.

- **C-11-5-2.** 스크립트가 작동하는 컴포넌트와, 리소스 모델을 보여주거나 애니메이션하는 컴포넌트는 가급적 다른 프리팹으로 분리한다.

※ 이렇게 하는 이유는, 비주얼을 위한 리소스 데이터(FBX 등)를 바꾸기 위해서는 프리팹 자체를 새로 만들어야 하는 경우가 있기 때문이다.
이렇게 되면, 스크립트를 포함한, 나머지 전체 컴포넌트들도 새로 만들 프리팹에다 전부 새로 설정을 해줘야 하는데, 이게 정말 번거로운 일이다.

※ GameObject의 계통 구조를 나눠서 컴포넌트를 설정하는 방식이 성능에 악영향을 미치는데 대해 정식으로 테스트한 경우는 아직 없다.
하지만 보다 단순한 계통 구조를 가진 경우에도 실제 모바일 대상 기기에서는 정점 개수나 실시간 그림자 등이 훨씬 더 성능에 결정적인 영향을 미치는 것으로 판명되었기 때문에, 계통 단계가 약간 더 복잡해진다고 해서 성능이 눈에 띌 만큼 떨어지지는 않는다고 판단한다.

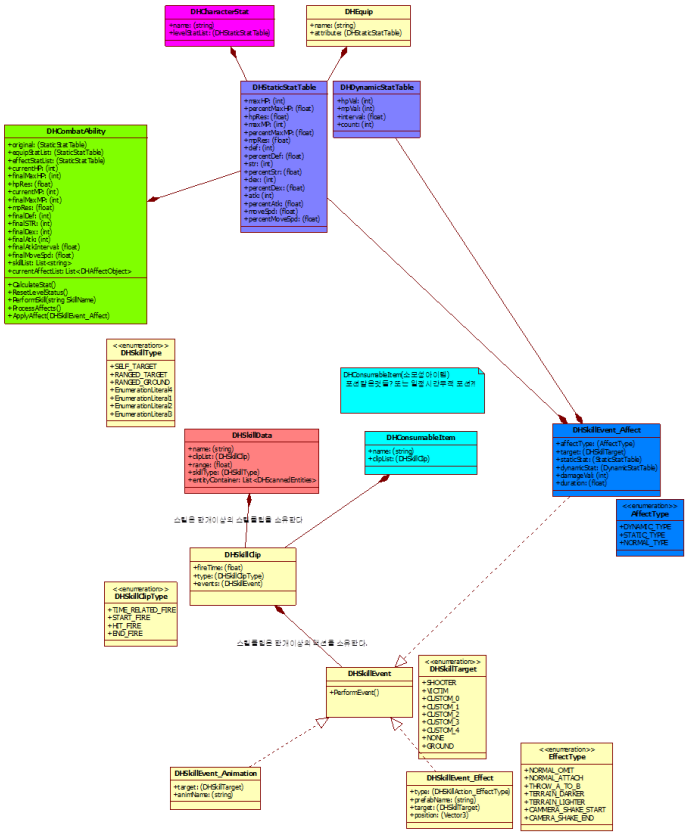
- **C-11-5-3.** 프리팹은 객체와 컴포넌트의 조합 상태를 담당하게 하고, 가급적 **변수 값을 초기화하는 목적으로는 사용하지 않는다.**

※ 프리팹에 포함된 초기값은 텍스트 데이터만큼 쉽게 찾아보고 알아보기가 어렵다.
또한, Unity 엔진의 프리팹은 그 개념과 활용도로 볼 때, 리소스 모델 등의 시각 데이터와 강하게 결합될 경우가 많다.
리소스 모델에 사용할 Mesh 데이터 등은 메모리를 많이 차지하는데다, 게임 플레이 장면으로 들어 갈 때 빼고는 사용할 일이 없다. 그런데 캐릭터 데이터가 프리팹에 결합되어 있다면, 게임 플레이가 아닌 장면에서는 캐릭터 데이터를 사용해야 하기 때문에, 필요하지도 않은 정보까지 결합된 프리팹의 객체 정보를 불러와야 할 수가 있다.

반면, XML에 저장할 스크립트의 데이터는 오직 숫자와 문자열 등의 작은 데이터만으로 이루어져 있고, 게임을 시작할 때 한꺼번에 불러온다. 또한, 전역으로 관리하기 때문에, 현재 불려온 장면(Scene)과 상관없이 사용할 수 있다.

이런 이유 때문에, 프리팹은 게임 플레이에 직접 사용할 큰 용량의 데이터를 결합하는 용도로 사용하고, 게임 데이터 스크립트(XML)는 그 프리팹이나, 3D 모델 등의 데이터가 필요하지 않은 다른 장면에서 활용할 수 있도록 변수 데이터의 초기 값을 저장하도록 사용한다.

<캐릭터의 모델은 게임 플레이 장면에서만 필요하다.>



<그러나 이러한 캐릭터의 능력 수치는 게임 플레이 외, 다른 장면에서도 필요하다>

C-12. 컨텐츠 데이터 스크립트

◆ C-12-1. 게임 데이터 스크립트(Script)

- **C-12-1-1.** 게임 데이터 관련 스크립트는 키 – 값(Key – Value) 쌍으로 구성하는 **튜플(Tuple)들의 관계형 모델**로 작성한다.
  : 소위, 행(Row)과 열(Column)을 가지는 테이블 방식의 집합으로 구성하는 방식의 데이터를 말한다.

※ 여러 개의 스프레드 시트 데이터들을 예로 들 수 있다.
관계형 데이터베이스 모델 역시 이와 같은 개념이다.

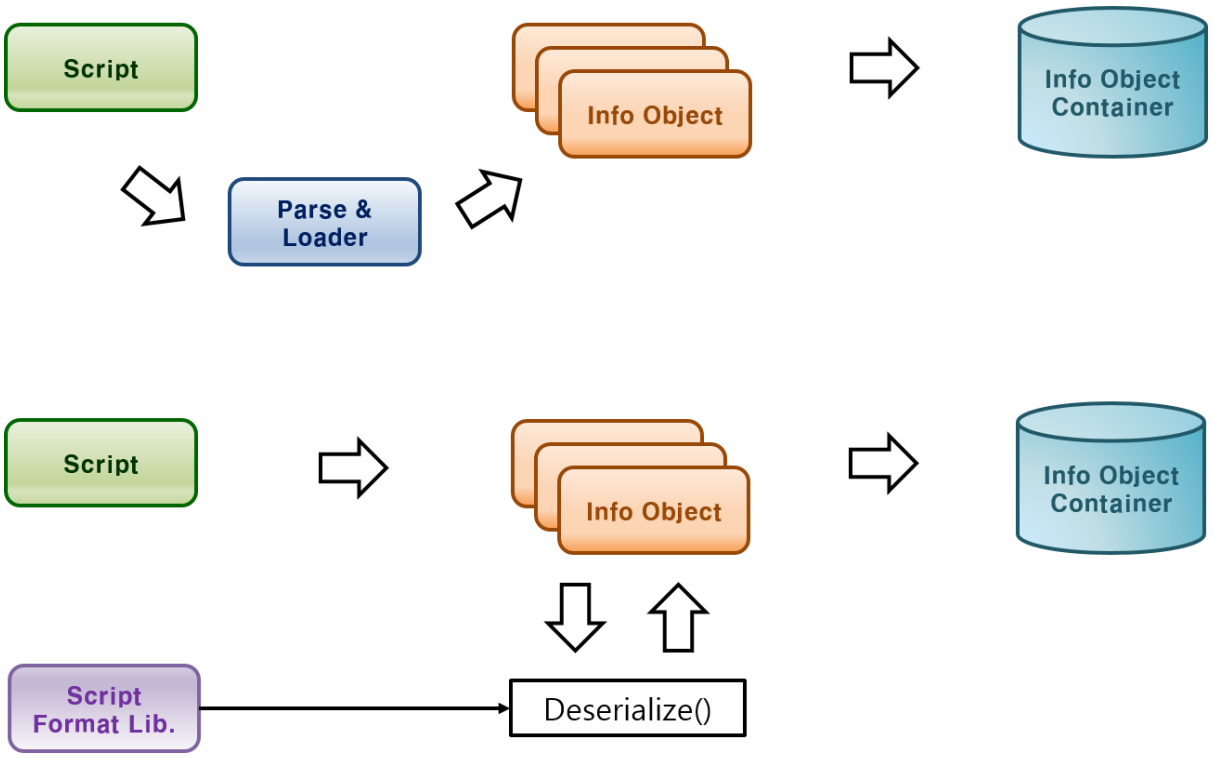
- **C-12-1-2.** 게임 데이터에 쓰이는 스크립트는 **일반 텍스트를 기반으로 하는 형식**이어야 한다.
  : XML, JSON 등의 표준화되고 널리 쓰이는 형식을 추천한다. CSV(Comma-Separated Values) 방식도 나쁘지는 않지만, 상대적으로 전자의 포맷이 좀 더 최근 추세에 알맞다.

※ 여기에서 말하는 스크립트는, Unity3D 엔진에서 게임 이벤트를 구동하는 소스의 코드로 사용하는 C#이나 Javascript와는 다르다. 그냥 순수하게 리소스 데이터를 불러오기 위한 정보를 담고 있는 좀 더 단순한 형태의 텍스트 데이터를 의미한다.

※ 이진 형식(Binary Format)을 기반으로 하는 파일을 쓴다면 데이터의 크기를 더 작게 유지할 수 있고, 훨씬 최적화되어 있기 때문에 읽어 들이는 속도도 더 빠르게 할 수 있다.
하지만 게임을 서비스해야 하는 플랫폼이 여러 개이고, 각각의 운영체제와 최종 번역하는 프로그래밍 언어가 다르다는 점이 문제이다. (iOS는 Objective-C, Android는 Java를 쓴다.) 즉, 하나의 이진 파일 형식이 여러 운영체제에서 똑 같은 바이트 순서와 형식 크기를 가진다고 보장하지 못할 수도 있다. 게다가 이진 파일 형식은 디버깅 하기도 더 까다롭다. (일반 텍스트 기반 파일은 심지어 컴파일러와 디버거를 동원하지 않더라도, 그냥 텍스트 편집기에 띄워놓은 상태로 문제점을 찾을 수도 있다.)
정말 피치 못할 이유가 있는 게 아니라면(최종 패키지 용량을 단 1MB라도 줄여야 할 정도로 용량이 가장 최우선이라거나, 형식 보안이 최우선이라거나), 이진 형식을 쓰는 게 그렇게까지 큰 이득이 된다고 보기는 어렵다. 차라리 그냥 텍스트 기반 파일들을 압축하는 방법을 시도하는 게 더 낫다.

- **C-12-1-3.** 리소스 데이터 정보를 담는 스크립트의 해석기를 구현해야 할 경우, **각자 그 내용을 읽어 들이는 전용 해석기 클래스를 가져야** 한다.

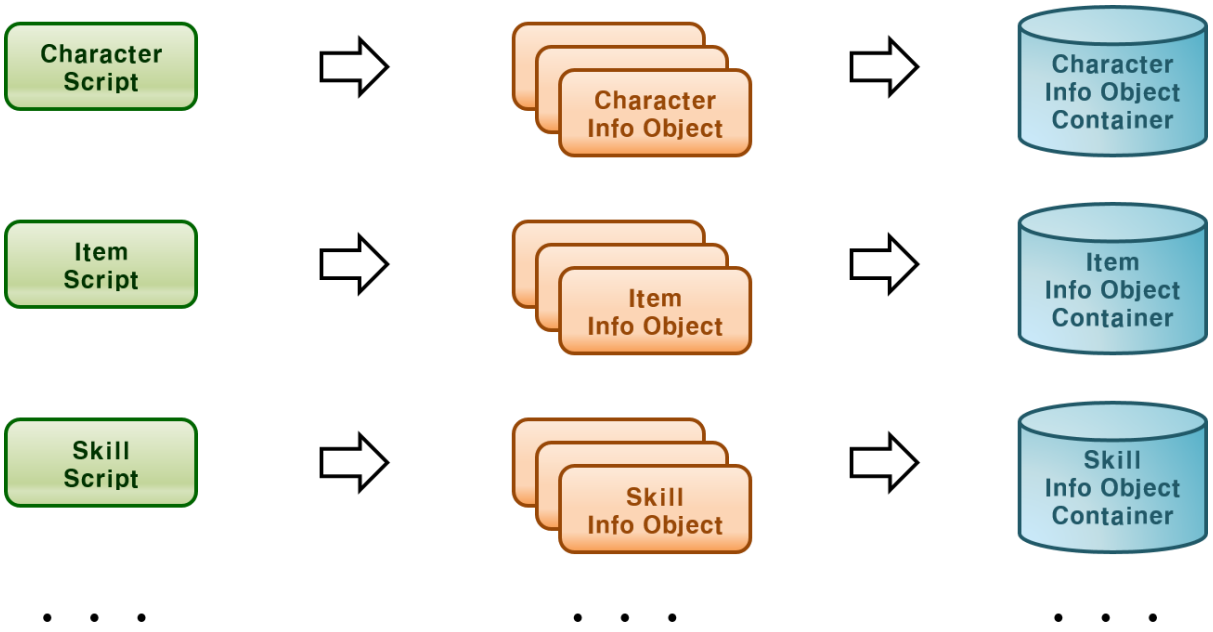
※ XML이나 JSON은 여러 언어에서 직렬화 – 역직렬화(Serialization – Deserialization) 기능을 구현하고 있다.
이러한 기능을 이용하면, 다수의 해석기 클래스를 작성하지 않고도, 간편하게 해당 언어로 데이터 스크립트의 내용을 저장하게 만들 수 있다.
하지만, 이 방법이 모든 면에서 다 장점만 있지는 않다.
복잡한 데이터 모델을 테이블 형식으로 표현하기 위해, 일부 필드를 특수한 방법으로 해석하게 만드는 요령을 부릴 수 없다. 언어적으로 허용하는 '표준적인' 방식으로만 테이블의 데이터 형식을 구성해야 하는 점은, 때로는 테이블 데이터의 모델링에까지 영향을 주게 만든다. 때로는 이런 제약 사항이 테이블 모델을 지나치게 복잡하고 비직관적으로 보이게 만들기도 한다.
특히, 필드에 들어가야 하는 데이터가 일종의 컨테이너 형식이어야 하거나, 가변적인 형식과 길이를 가져야 하는 데이터일 경우에는 직렬화를 활용해서 무사하게 데이터를 들여오거나(Inport) 내보내도록(Export) 하기가 쉽지 않다.



<전용 해석기를 구현하거나, 혹은 직렬화 기능을 이용한다.>

- **C-12-1-4.** 여러 XML에 대해 공통적으로 동작하는 해석기 클래스는 피하는 것이 좋다. 유지보수 하는 사람이, 어떤 클래스가 어떤 XML들에 대해 작동하는 것인지 혼동하기 쉽다.

※ 직렬화 기능을 이용하더라도, 각 스크립트 테이블마다 전용으로 직렬화 대상이 될 객체를 분리하는 방식이 좋다.
만약 직렬화 기능을 쓰지 않고 테이블 데이터를 직접 해석하는 소스를 구현한다면, 해당 해석기 객체가 테이블 데이터의 종류만큼 존재하는 게 좋다.



<스크립트 – 정보 객체 – 컨테이너가 정보의 종류마다 하나씩 대응한다.>

- **C-12-1-5.** 모든 게임 데이터 스크립트의 정보들은 게임이 시작할 때, 한꺼번에 순차적으로 불러서, 그 내용을 저장해두어야 한다.
: 이를 위해서, 게임 데이터 스크립트는 메모리 상에서 가장 단순하고 작은 형식의 정보들, 즉, **숫자와 문자열 기반의 데이터들만으로 구성**하도록 한다.

- **C-12-1-6.** 게임 데이터 논리에 대한 변수 값 설정은, 가급적 게임 데이터 스크립트를 통해서 초기화하도록 한다.

※ 이 부분이 중요한 이유 중 하나는, 내 캐릭터의 경우에는, 게임 플레이 할 때 사용하는 리소스 모델이 불러지지 않는 장면(Scene)에서도 그 데이터가 필요하기 때문이다.
게임 데이터와 시각적인 모델 데이터를 손쉽게 분리하기 위해서는, 게임 프로그램이 시작할 때부터, 게임 데이터에 대한 부분만 먼저 불러올 수 있어야 한다.

◆ C-12-2. 구현 정책

- **C-12-2-1.** 서버와 클라이언트가 각자 필요한 부분만을 데이터 스크립트로 정의해서 보유한다.
: 각자의 모듈에서 알 필요가 없는 정보들을 중복해서 포함하지 않는다.

※ 클라이언트에서 서버에서만 필요한 데이터 스크립트들을 들고 있는 경우가 특히 문제가 된다.
또한, 대개 보안 문제와 관련이 깊다.

서버에서만 필요한 데이터 스크립트들은 대개 능력치, 전투 판정에 관련한 값들이기 때문에, 치트, 변조를 통해서 이득을 취하려고 할 때, 최고의 목표 대상이 된다. 직접 이용하거나, 그렇지 않더라도 서버와의 통신 내용을 분석하는 근거로 쓸 수 있는 등 악용될 수 있다.
그러니, 특히 불필요하게 서버에서만 가지고 있어야 할 데이터를 외부 사용자가 접근할 수 있는 응용 프로그램 패키지 안에 포함하지 않는 게 좋다.

- **C-12-2-2.** 상태 값과 판정에 관련한 원시 값(Primitive Data)들과 관련한 사항들은 서버의 데이터 스크립트에서 관리한다.

- **C-12-2-3.** 모델 데이터, FX 데이터, 리소스 데이터, 프리팹 데이터 등 클라이언트가 적재한 리소스와 직접 관련이 있는 사항들은 클라이언트의 데이터 스크립트에서 관리한다.

- **C-12-2-4.** 클라이언트의 데이터 스크립트에서는 검색 키는 일반 타입 식별코드 (General Type Code, GTC)를 사용하지 않아도 된다.
: GTC는 엄밀히 말하자면, 모든 모듈에서 공통적으로 사용할 키 값을 정의하기 위한 도구이기 때문에, 클라이언트 내부에서만 사용하는 키 값을 GTC에 꼭 맞춰서 쓸 필요는 없다.

※ 경우에 따라, 문자열을 키 값으로 사용하는 방식이, 대량의 스크립트를 편집하는 사람의 입장에서는 더 편할 수 있다.
단, 자료형의 비교에서 문자열의 비교는 정수 값 비교에 비해 훨씬 느리다는 사실은 염두에 둬야 한다.

C-13. 국제화(Internationalization)

◆ C-13-1. 기본 사항

- **C-13-1-1.** 게임 프로젝트에서 사용하는 **모든 문서와 데이터의 문자열은 유니코드로 작성**한다.
  : 멀티 바이트 형식(운영체제의 특정한 인코딩 설정에 따라 각 바이트의 문자열 해석이 달라지는 방식)을 사용해서는 안 된다.

※ 특별한 제약 사항이 없는 한, 작성하는 텍스트 파일의 인코딩은 모두 UTF-8 인코딩으로 통일한다.

- **C-13-1-2.** 특별히 필요한 경우가 아니라면, 각 지역 별로 배포하는 응용 프로그램이 달라지지 않도록 **하나의 응용 프로그램 빌드에서 모든 지역과 언어를 지원하게 하도록 설계**한다.

- **C-13-1-3.** 다만, 하나의 응용 프로그램 빌드에서 모든 지역화를 지원하는 게 불가능한 경우라면, 최소한 하나의 프로젝트에서 모든 지역별 빌드를 만들 수 있는 방식은 되어야 한다.

※ 아쉽게도, 게임은 그 특성상 텍스트 데이터 뿐 아니라, 각종 미디어 컨텐츠들도 지역화에 따라서는, 아예 리소스 자체를 다르게 사용해야 할 경우가 있다.
 또한, 실행 파일의 용량 제한이 민감한 이슈가 되기도 한다. 또한 이동 통신 기기들은 통신 요금이 중요한 비용 중 하나이기 때문에, 되도록 대용량 리소스들의 다운로드가 적을수록 좋다.
 그러다 보니, 하나의 응용 프로그램 빌드로 모든 언어와 컨텐츠를 전환하게 만들기는 아직 시기적으로 어렵다.

- **C-13-1-4.** 서비스 지역에 따라 다른 프로그램 코드를 동작시켜야 하는 경우, **하나의 프로젝트 소스 코드 내에서 조건에 따라 분기**해야 한다.
  : 프로그램의 설계 구조를 적절하게 설정하거나, 사용하는 프로그래밍 언어의 조건부 실행 / 조건부 컴파일 설비를 활용해서 실행 논리(또는 실행 빌드 결과물)를 구분해줘야 한다.

※ 가장 좋은 방법은 서비스 지역 전용의 실행 논리를 별도의 객체에 몰아넣고, 조건 분기에 따라 그 전용 객체를 호출해서 처리하는 방식이다.
 하지만, 경우에 따라서는 그런 처리가 불가능할 수도 있다. 이 때는 #ifdef 등의 전처리기를 이용하거나, Conditional Attribute(C# 언어의 경우)를 이용해서 소스 코드가 사전에 설정된 전처리 지시문 또는 조건 애트리뷰트에 의해 별도로 컴파일하도록 한다.
 요점은, **프로젝트의 소스 코드는 한 벌인 상태를 유지하는 게 중요**하다는 점이다.

- **C-13-1-5.** 응용 프로그램 내부에서 지역 및 언어 설정에 따라 해당 지역에 맞는 게임 자산을 불러올 수 있게 해야 한다.

- **C-13-1-6.** 불가피한 이유로 지역 및 언어 설정을 전환할 때, 응용 프로그램을 재시작 해야 할 필요가 있다면, 변환이 이루어진 시점에, 사용자에게 이를 알려야 한다.

※ 보통, '게임을 재시작할 때 적용됩니다.' 같은 내용을 써 있는, 확인 버튼이 하나 있는 팝 업 GUI를 띄우는 경우가 많다.

◆ C-13-2. 시스템

- **C-13-2-1.** 숫자 표시
: 모든 숫자는 예외 없이 아라비아 숫자로만 표기한다.
단, 일부 컨텐츠에서 한 자리 숫자를 로마 숫자(I ~ X)로 표기하는 경우까지는 허용한다.

※ 특정 문화권에서만 쓰는 숫자 표기를 남발하게 되면, 다른 지역 버전을 만들 때마다 그 부분을 전부 해당 지역 표기로 고쳐야 하기 때문에 일이 아주 힘들어진다.

- **C-13-2-2.** 날짜 표시
: 게임이 실행되고 있는 기기의 운영체제 설정을 그대로 따른다.
하지만, 만약 그러한 설정을 활용하기가 어렵다면, **[YYYY]-[MM]-[DD]** 형식의 국제 표준 방식을 사용한다. 특히, 연도(Year)는 2 자리로 축약해서 표현하지 말 것.

※ 지역 문화권마다 00 / 00 / 00 식으로 되어 있는 날짜의 해석 방식이 다르다. 국제 표준과 일치하게 사용하는 곳도 있지만, 완전히 반대 순서로 사용하는 곳도 있다.
특히, 연도를 2자리로 축약하는 관례는 일 / 월의 단위와 혼동할 여지가 있기 때문에, 가급적 안 쓰는 게 좋다.

- **C-13-2-3.** 통화 표시
: 서비스하는 플랫폼 마켓에서, 상품의 가격을 표시할 때 특별히 통화의 제한을 두지 않는 한, 게임이 실행되고 있는 기기의 운영체제 설정을 그대로 따른다.

※ 기기 운영체제의 언어와 관계 없이 가격을 미화 달러(USD, $)로만 표시해야 할 수도 있고, 서비스 지역에 맞는 가격 단위로 환산해서 표기해야 할 수도 있다.

- **C-13-2-4.** 버전 표기
: 배포하는 응용 프로그램의 버전을 표기하는 방식은 국가, 언어, 지역에 관계 없이 단 하나의

형식만을 사용한다.

◆ C-13-3. 텍스트 데이터

- **C-13-3-1.** **사용자에게 노출해야 할 필요가 있는 모든 텍스트들은 소스 코드에 직접 적혀 있으면 안 된다.**

- **C-13-3-2.** 사용자에게 노출해야 할 필요가 있는 모든 텍스트들은 사전에 정해진 외부의 스크립트 파일에서 불러오도록 설계해야 한다.

- **C-13-3-3.** 모든 텍스트 데이터는 그 텍스트 데이터를 지칭하는 고유한 번호가 있어야 한다.

- **C-13-3-4.** 같은 텍스트 코드 번호의 텍스트 내용들은, 언어 분류와 국가 분류의 코드 번호를 이용해서 각 언어 / 국가 별 내용으로 분류할 수 있어야 한다.

- **C-13-3-5.** 텍스트 데이터를 담는 스크립트 파일은 하나 또는 제한된 개수여야 하고, 다른 여타 스크립트 여기저기에 흩어져 있으면 안 된다.
  : 텍스트 데이터가 따로 모아져 있지 않고 흩어져 있으면, 번역 작업 등을 할 경우에 수정 비용이 매우 커진다.

◆ C-13-4. 기호 및 이미지

- **C-13-4-1.** 전 세계적으로 공통적으로 쓰이는 기호와 이미지를 위주로 사용한다.
  : 특정한 문화권에서 의미가 달라지는 기호와 이미지, 특정한 문화권에서만 알아볼 수 있는 기호나 이미지는 가급적 피한다.

- **C-13-4-2.** 응용 프로그램에서 언어와 문화권 설정에 따라 불러들이는 기호와 이미지 자산을 다르게 설정할 수 있어야 한다.

◆ C-13-5. 게임 내용

- **C-13-5-1.** 각 언어권 및 문화권에 대해 게임 내용이 달라야 하는 경우에, 이를 별도의 소스 프로젝트와 빌드로 분리해서 만들면 안 된다.
  : 하나의 소스 코드와 빌드에서 모든 내용을 지원할 수 있도록 분류하는 체계를 소스 코드의 내용 구조 안에 갖춰야 한다.

- **C-13-5-2.** 단, 빌드 결과물의 패키지는 대상 언어와 문화권에 따른 전용 컨텐츠 리소스 만으로 구성하는 방식을 허용한다.
: 일부 마켓에 의해 공중 다운로드의 허가 용량에 제한이 있기도 하고, (대표적으로 Apple Appstore는 100MB, Google Play는 ~~50MB~~100MB) 모든 국가와 권역 별 컨텐츠 리소스를 하나의 패키지에 다 넣는 건 비현실적이다.

※ 예를 들자면, 한국어와 중국어, 일본어 버전으로 각각 출시를 해야 한다고 치자.
세 국가는 문화적으로도 조금씩 다르고, 언어도 다르기 때문에, 게임의 텍스트 내용은 물론이거니와, 첫 화면에 표시할 게임 제목의 텍스트와 그림부터 다를 수 있다.
그렇다고 해서 세 국가 버전의 전용 컨텐츠 리소스들을 모두 하나의 빌드 패키지에 집어넣는 건 좋은 방법이 아니다. 왜냐하면 패키지 용량이 너무 커지고, 그렇게 되면 일부 마켓에서는 Wi-fi로만 다운로드 해야 할 수 있다. 이는 사업적인 측면에서 받아들이기 어려운 방식이다.

그러니, 결국 세 국가 버전을 별도의 빌드 패키지로 만들어야 한다.
그런데 그렇다고 해서, 세 국가 버전 별로 소스 프로젝트를 따로 두는 결정은 더욱 좋지 않다.
국가 / 지역 별로 나눈 소스 프로젝트가 여러 벌이라면, '거의 같지만 약간씩 내용이 다른 부분이 있는' 소스 코드와 리소스들을 여러 벌 관리해야 한다. 이건 서비스 국가와 지역이 조금만 더 생겨도 관리 상의 파멸을 가져올 결정이다.

가장 좋은 방법은 **소스 프로젝트는 단 한군데이지만, 빌드 패키지는 각 국가 / 지역 별로 구분해서 생성**할 수 있는 소스 코드의 구조와 빌드 시스템을 갖추는 것이다.

C-14. 과금 / 결제

◆ C-14-1. 제한사항

- **C-14-1-1.** 서비스할 지역의 법률적인 제한 조건을 따른다.

- **C-14-1-2.** 서비스할 지역의 통화(Currency) 단위를 지원해야 한다.
: 이건 일반적으로 플랫폼 단계에서 지원할 것이다. (아마도...)

- **C-14-1-3.** 지원하는 플랫폼 및 장치에 대해 동일한 결제 서비스를 제공해야 한다.
: 이것도 플랫폼 단계에서 지원할 것이다.

- **C-14-1-4.** 환불과 관련된 정책은 해당 지역의 법률 제한 및 계약한 발행사 / 통신사의 정책과 연관하여 정한다.
: 그러나, 기본적으로는 실행한 결제에 대해서 환불하지 않는 것을 원칙으로 한다.

※ 대한민국에서 서비스할 경우, 이동통신사를 거쳐서 환불을 처리해야 하기 때문에, 사실상 환불 기능을 추가하기는 어렵다고 함. (별로 할 생각도 없지만...)

- **C-14-1-5.** 구입에 제한이 있는 컨텐츠가 존재하는지? 그리고 그것을 각 클라이언트가 식별하는 과정이 필요한지?
: 아마 게임 레벨 관련 컨텐츠는 그럴 것 같지만...

- **C-14-1-6.** 모든 형식의 금액 단위를 막론하고, 반드시 **10억 단위 밑으로 지정해야** 한다.

※ 금액을 표시하기 위한 정수값을 4바이트로 잡기로 했다.
: 4바이트의 최대 값은 -2,147,483,648 ~ -2,147,483,647이다. (효과적인 오류 처리를 위해, **부호 없는 정수 값으로의 사용은 고려하지 않기로** 한다.)

◆ C-14-2. 결제 처리 프로세스

- **C-14-2-1.** 클라이언트에는 게임 서버에게, 특정한 마켓에 과금을 해야 구매할 수 있는 아이템을 구매하겠다는 통지를 한다.

- **C-14-2-2.** 게임 서버는 클라이언트의 요청 내역을 확인하고, 결제 서버를 통해 요청한 클라이언트가 상품을 구매했는지 확인할 때 필요한 확인 키를 암호화해서 발급한다.
  : 암호 키가 어떤 형식일지는 마켓에 따라 다르다.

- **C-14-2-3.** 클라이언트는 서버로부터 받은 확인 키와 함께, 마켓에 상품의 구매를 요청한다.

- **C-14-2-4.** 구매가 성공하면 마켓은 영수증을 발급할 것이고, 영수증 데이터를 수신 받은 클라이언트는 그 영수증 데이터를 서버에게 전달한다.

- **C-14-2-5.** 서버는 영수증 데이터를 자신이 가지고 있는 확인 키 등을 비교하는 검사를 수행해서 영수증이 가짜가 아닌지 검증한다.
  : 가짜 영수증이면 조작 시도가 있었다는 뜻이고, 거래는 취소되며, 거래를 시도한 클라이언트를 관리상 적절한 방법으로 처리한다.

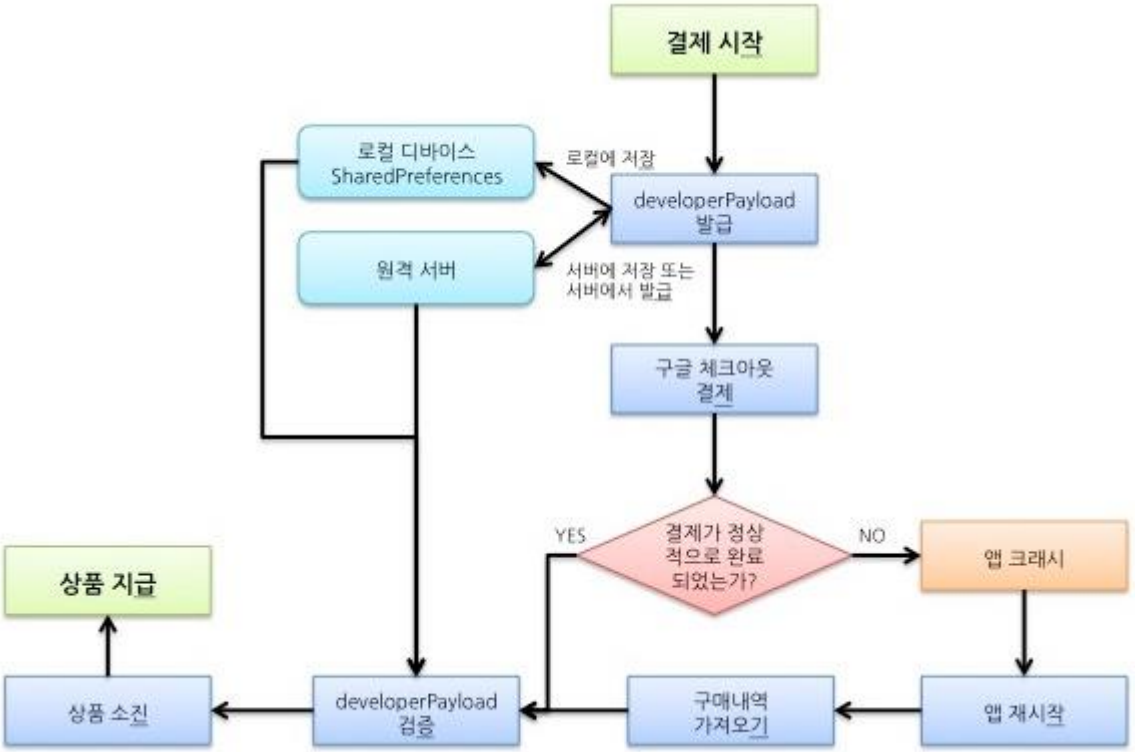
※ 클라이언트가 마켓을 통해 상품을 '구매'하더라도, 이를 실제로 적용시키는 작업은 서버에서 하기 때문에 이런 결제 취소가 가능하다.
단, 이는 환불과는 엄연히 다르다. 마켓에 대한 결제 자체가 성공했다면, 실제 상품 구매를 적용할 때까지 서버에 계속해서 결제 확인 요청을 할 수 있다는 의미일 뿐이다. 물론, 재차 결제 확인을 요청할 때도 가짜 영수증을 쓴다면 서버 측에서는 계속해서 기각할 것이다.
(그리고 가짜 영수증을 지속적으로 보내는 사용자 계정은 블랙리스트에 올라가게 되겠지...)

- **C-14-2-6.** 결제 서버가 구매 영수증이 진짜임을 확인했으면, 마켓 서버에 요청해서 이 결제 과정이 실제로 일어났는지 점검한다.
  : 마켓에 따라 이 과정이 없을 수도 있다.

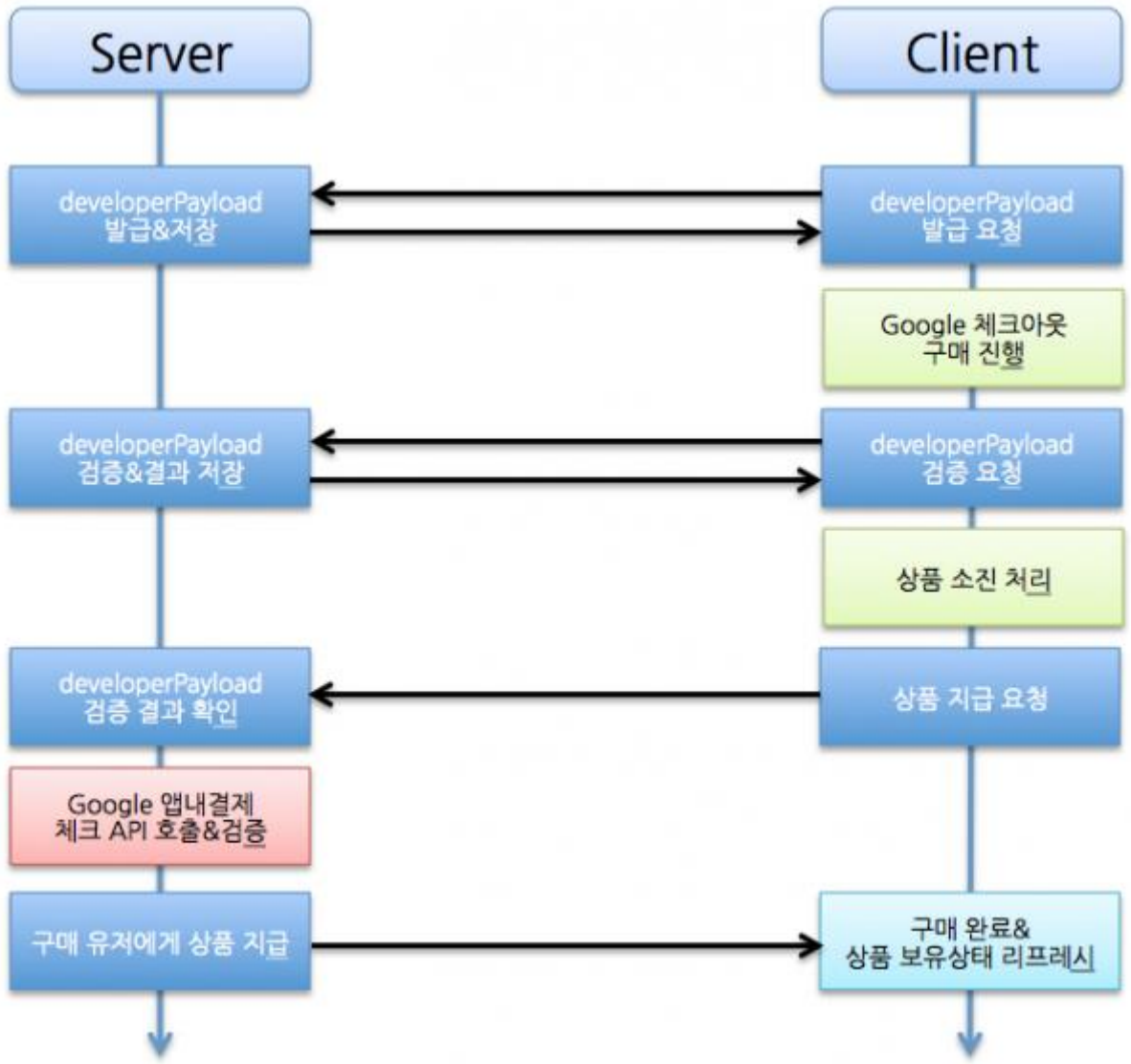
- **C-14-2-7.** 모든 검증 과정을 통과했다면, 구매한 상품의 재화 단위와 수량을 조사해서 사용자 계정의 해당 값에 추가한다.

- **C-14-2-8.** 결제 서버는 게임 서버를 통해 결제가 완료되었음을 통지하고, 게임 서버는 다시 클라이언트에게 결제 성공과 그 결과를 알린다.

- **C-14-2-9.** 클라이언트는 플레이어에게 구매한 상품의 재화가 늘어났음을 표시한다.


결제 시작
로컬 디바이스
SharedPreferences
로컬에 저장
developerPayload
발급
원격 서버
서버에 저장 또는
서버에서 발급
구글 체크아웃
결제
YES
결제가 정상
적으로 완료
되었는가?
NO
앱 크래시
상품 지급
상품 소진
developerPayload
검증
구매내역
가져오기
앱 재시작

< 권장하는 결제 프로세스 >



◆ C-14-3. 결제 실패에 대한 처리

- **C-14-3-1.** 마켓에서 구매를 하는 도중에 네트워크 연결이 끊기거나 애플리케이션이 어떤 이유로 종료된 경우

  :

- **C-14-3-2.** 마켓에서는 구매 처리를 하였는데, 클라이언트가 서버에게 구매 내역을 제출하기 전에 네트워크가 끊긴 경우

  :

◆ C-14-4. 결제 모듈 관리

- **C-14-4-1.** 반드시 하나의 응용 프로그램 프로젝트에서 모든 외부 결제 모듈들을 다룰 수 있게 작성해야 한다.
  : 결제 모듈과 같은 일부 라이브러리 때문에 소스 코드 프로젝트 자체를 분기해서 유지하는 구조가 되어서는 안 된다.

※ 특정 지역, 특정 플랫폼 마켓만을 지원하기 위해서 별도의 소스 코드 프로젝트를 만들기 시작하면, 나중에 관리할 때 큰 부담이 된다.

- **C-14-4-2.**

C-15. 보안 / 암호화

◆ C-15-1. 제한사항

- **C-15-1-1.**

◆ C-15-2. 사용자 데이터 보안

- **C-15-2-1.**

◆ C-15-3. 애플리케이션 및 게임 데이터 보안

- **C-15-3-1.** 일단 게임 실행 중 주요한 판정 데이터에 한하여 CRC를 적용하고, 데이터에 대한 난독화 솔루션을 적용하는 방향으로 계획한다.

※ 데이터 난독화는 암호화가 아니기 때문에, 역공학(Revers Engineering), 치트 행위들로부터 완전히 안전한 기법이 아니며, 단지 시간만 약간 끌어줄 뿐이다.
하지만, 일단 손쉬운 치트 시도의 대부분을 좌절시킨다는 점에서는 충분히 시간을 투자할만한 가치는 있다. (물론, 결국 뚫려버린다면 어쩔 수는 없고...)

- **C-15-3-2.** 디지털 권리 관리(Digital Rights Management, DRM) 처리
:

- **C-15-3-3.** 애플리케이션과 게임 데이터 보안에 대한 자체적인 대책이 세워질 때까지, 이 부분에 대해 최소한의 안전장치가 없는 마켓에는 출시하지 않는 것으로 한다.

◆ C-15-4. 과금 / 결제 보안

- **C-15-4-1.** 모든 결제 과정은 서버를 통한 승인 과정을 거치도록 한다.
:

- **C-15-4-2.** 보안 조치가 되지 않는 마켓에는, 자체적인 대책이 세워질 때까지 출시하지 않는 것으로 한다.

◆ C-15-5. 데이터 복제 / 변조 방지

- **C-15-5-1.** 아이템 데이터 복제 방지

◆ C-15-6. 루팅 방지

C-16. 오류 / 예외 처리

◆ C-16-1. 오류에 대한 분류

- **C-16-1-1.** 게임 프로그램에 의해 발생하는 모든 오류들은 고유한 오류 번호를 가져야 한다.

- **C-16-1-2.** 오류 번호는 **프로젝트 내의 자체 분류에 의해 매겨지며,** 대상 장치의 시스템 오류 번호와는 무관하다.

- **C-16-1-3.** 시스템 오류
: 게임 프로그램 자체 오류가 아닌, 대상 장치의 내부 오류.

- **C-16-1-4.** 네트워크 오류
: 게임 프로그램의 연결과 관련된 오류. 네트워크 플레이와 관련된 버그 성 상황들도 포함한다.

- **C-16-1-5.** 서버 오류
: 서버 내부의 논리적 실행에 대한 오류들이다.
서버 오류는 일반적으로 사용자에게 직접 노출하지 않는다. 이 오류들은. 완전히 서버 프로그램을 유지하는 데만 필요한 내용들이다.

- **C-16-1-6.** 클라이언트 오류
: 클라이언트의 논리적 실행에 대한 오류들이다.
일반적으로 이 부분은 게임 클라이언트 프로그램를 실행했을 때 발생할 수 있는 오동작에 대한 내용들을 모두 포함한다.

◆ C-16-2. 예외 처리

- **C-16-2-1.** 서버의 게임 서비스 관련 내용으로부터 파생하는 모든 예외들은 **서버의 서비스 프로세스를 종료하지 않도록 핸들링**하여야 한다.

※ 사람이 작성하는 프로그램이다 보니, 오류나 문제점이 없을 수는 없을 것이다. 하지만 오류가 발생할 때 서버 프로세스가 곧장 죽어버리는 경우는 큰 문제가 발생한다. 운영체제 단계에서의 문제로 인한 프로세스 종료까지 컨트롤 할 수는 없겠지만, 적어도 게임 서비스 관련 코드에서

발생한 예외 때문에 운영체제가 서버 프로세스를 죽여 버리는 일은 없도록 해야 한다.

- **C-16-2-2.** 서버의 게임 서비스 코드로부터 발생한 예외 사항들은 적절한 오류 코드로 분류한 뒤, 로그로 기록해야 한다.

◆ C-16-3. 오류에 대한 기록

- **C-16-3-1.** 서버에서 발생하는 모든 오류들은 **가능한 한 전부 로그를 남겨 기록**해야 한다.

- **C-16-3-2.** 클라이언트와 저작도구, 서비스 관리 도구에서 발생하는 오류들은 서버만큼 엄격하게 오류를 로깅하지 않아도 되지만, 가급적 로그를 남기는 것이 좋다.

- **C-16-3-3.** 클라이언트의 경우, 오류에 대해 지나치게 자세한 정보를 남기지 않도록 주의한다.
  : 사용자가 악용하여 게임 서비스를 공격하는 수단 중 하나가 될 수 있기 때문이다.

※ 해킹할 때 가장 좋은 재료들 중 하나가, 자신이 공격할 프로그램에서 오류가 발생했을 때, 어떤 오류 메시지를 내뱉는지를 보는 것이다. 개발하고 유지 보수하는 입장에서는 오류가 난 방식을 최대한 자세하고 체계적으로 분류할수록 좋지만, 이러한 이면이 있음에 주의해야 한다.

- **C-16-3-4.** 서버나 데이터베이스에서 발생한 **오류 메시지 내용을 그대로 클라이언트에게 전송하면 절대로 안 된다.**
  : 악의를 가진 사용자들이 서버나 DB를 공격할 수 있는 취약점 분석 근거로 활용할 수도 있기 때문이다.

※ 일반 사용자에게 배포하는 클라이언트는 서버로부터의 오류를 전송 받을 필요가 있다 하더라도, 가급적 두루뭉술하게 '가공한' 오류 메시지를 받도록 해야 한다.

◆ C-16-4. 다국어 환경을 지원하는 오류 메시지 처리

- **C-16-4-1.** 게임 프로그램에 의해 발생한 오류 / 예외들은, 고유의 오류 번호를 가지고 대상 언어의 메시지로 사용자에게 오류 내용을 알려줄 수 있어야 한다.

- **C-16-4-2.** 오류의 고유 번호는 어느 언어 환경을 사용하든 동일해야 한다. 그렇지만, 오류 메시지의 텍스트는 각 언어 환경에 따라 분리하여 작성하여야 한다.

※ 오류 번호를 인덱스로 하는, 언어별 오류 메시지 텍스트가 별도로 존재하면 된다.
그래서 현재 설정된 언어 코드를 알고 있다면, 오류 번호를 통해서 해당 언어의 오류 메시지

문자열을 가져올 수 있도록 할 수 있다.

- **C-16-4-3.** 오류 메시지는 매개변수를 통한 문자열 서식을 지정할 수 있어야 한다.
: 오류 메시지 텍스트의 모든 사항이 사전에 결정적인 정보들만 담고 있을 수는 없다. 때로는, 실행 시점에서만 알 수 있는 정보들을 포함해야 할 수 있다. 이를 효과적으로 표현하려면, 오류 메시지를 표시할 때, 실행 시점에만 알 수 있는 사항들을 매개변수로 받아서 표현하는 방법을 써야 한다.

- **C-16-4-4.** 오류 메시지의 문자열 서식에서, 매개변수들은 각 언어 별 어순에 따라 등장 위치를 조정할 수 있어야 한다.

※ C# 언어의 문자열 서식 기능은 기본적으로 이 기능을 지원하기 때문에 큰 문제가 없다.
그러나 C 언어의 표준 문자열 서식 기능은 무조건 고정된 순서를 가지고 있기 때문에, 어순에 따라 등장 위치를 조정할 수 없음을 유의해야 한다. 이런 경우에는 이 문제를 효과적으로 구현한 제품이나 라이브러리를 찾아볼 것을 권한다.
C++의 경우에도 C 방식 문자열 서식 기능은 같은 문제를 가지고 있다. 그러니, 표준 템플릿 라이브러리나 제 3의 제품을 통해서 이 문제를 해결할 것을 권한다.

◆ C-16-5. GUI 구성

- **C-16-5-1.** GUI 데이터가 불려지기도 전에 발생한 오류 / 예외 때문에 프로그램을 탈출해야 한다면, 그 UI는 시스템의 가장 기본적인 GUI로 구성하는 것을 허용한다.

※ 이를테면, Unity3D에서 제공하는 기본 GUI 함수들을 들 수 있다.
현재는 Unity 엔진의 내부 GUI를 직접 사용하지 않고, NGUI 플러그인을 사용하지만, 만일 NGUI 요소들을 채 불러오기도 전에 오류 등으로 인해 게임을 중단해야 한다면, Unity 내부에서 사용하는 GUI 요소로도 사용자의 확인 과정을 핸들링 할 수 있어야 한다. (사실 이렇게까지 될 경우는 없을 것으로 예상한다.)

- P. S. : Unity 4.6 이후부터 UGUI이라고 약칭하는, 개편된 Unity 엔진의 내장 GUI 시스템이 탑재되었다. 이제는 NGUI 플러그인을 사용하지 않는다고 할지라도, 성능과 사용성의 제약을 받으면서 '옛 GUI 시스템'을 쓸 필요는 없어졌다.

C-17. 무작위 요소 처리

◆ C-17-1. 무작위 값 적용 방식

- **C-17-1-1.** 모든 무작위 값들은 값이 등장할 '간격(Interval)'을 가질 수 있다.

※ 프로그래밍 언어의 표준 정수형 난수 발생기들은 암묵적으로 1의 등장 값 간격을 가지고 있다고 말할 수 있다.
하지만 경우에 따라, 1, 3, 5, 7 등의 홀수만 발생하게 하고 싶다거나, 10 단위의 값만 발생하게 만들고 싶을 수도 있다.

- **C-17-1-2.**

◆ C-17-2. 무작위 뽑기(가챠)

- **C-17-2-1.**

◆ C-17-3. 능력치 확정 방식

- **C-17-3-1.**

◆ C-17-4. 확률 보장에 관한 처리

- **C-17-4-1.** 독립 시행
: 어떤 사항에 대해 확률을 부여하면, 시행 횟수마다 독립적으로 확률을 적용해서 무작위 결과를 가져오는 방식이다.

※ 이건 소위 '복불복' 방식이다. 10%의 성공 확률이라면, 정말 '재수 없으면', 100번 시행해도 단 한 번도 성공하지 못할 수도 있다. 독립 시행 방식은 이러한 결과가 나오는 것도 허용하겠다는 정책이다.

- **C-17-4-2.** 결정 시행
: 어떤 사항에 대해 확률을 부여하고자 할 때, 시행 횟수에 대해 확실하게 확률을 보장해주는 방식이다.

※ 만약 어떤 행위에 대해 10%의 성공 확률이어야 한다면, 100회 시행을 했을 때 90회의 실패와 10회의 성공을 보장해야 한다.

- **C-17-4-3.** 플레이어 입장에서 투자 대비 결과를 확정적으로 보장해줘야 하는 게임 요소일 경우에 사용한다.

- **C-17-4-4.** 결정 시행의 구현 방식은 완벽하게 수학적으로 결정적이지 않아도 된다.
: 독립 시행 방식에 점차 보정을 줘서 '사실상' 결정적인 확률에 수렴하게 하는 방식을 사용해도 무방하다.

※ 만약 어떤 행위에 대해 10%의 성공 확률이어야 한다면, 100회 시행을 했을 때, 10회까지는 실패 : 성공 = 90 : 10의 비율로 시행하고, 그 이후부터는 실패 : 성공의 확률이 89 : 11, 88 : 12처럼 점차 올라가는 방식을 쓸 수도 있다.
아니면 그냥 단순하게 구현하는 대안으로써, 90회까지의 시행에서 실패했으면, 91회부터는 성공과 실패의 확률을 뒤집어서 수행하는 방식을 쓸 수도 있다. (성공이 90%, 실패가 10%)

C-18. 게임 플레이 복구 / 재현

◆ C-18-1. 정책 및 구조

- **C-18-1-1. 게임 플레이의 복구나 재현이 필요한 사항들을 기획 요소에서 배제**한다.

※ 게임 플레이를 정확하게 복구하고 재현하기 위해서는, 게임 자산들을 결정론적인 방식으로 판정하고 업데이트해야 한다.
이러한 기능을 사용하기 위한 기반 설비에 해당하는 구조들이 갖춰져 있어야 하고, 모든 판정에 관련한 내용들은 결정론적 방식을 구현하기 위한 구조적인 규칙을 엄격하게 지켜야 한다.
또한, 끊임없는 테스트를 통해 결정론적인 게임 프로세스가 실제로 동작하는지, 올바르게 게임 동작을 처음부터 복구할 수 있는지를 검증해야 한다.

이렇듯, 게임 플레이 복구 / 재현을 위한 기반 구조의 설계와 구현은 작은 일이 아니다.
또한, Unity 엔진을 사용하면서 얻고자 하는 장점(게임 컨텐츠 구현에만 집중할 수 있게 해주는 기반 시설들)을 상당 부분 포기해야 하는 점도 있다.

D. 사업 및 운영

D-1. 시장 요소

◆ D-1-1. 목표 사용자 층

- **D-1-1-1.** **RPG에 어느 정도 익숙한 플레이어**
  : 사용자들이 기본적으로 액션을 강조하는 방식의 CRPG(Computer Role Playing Game)들이 공통적으로 가지고 있는 직업, 레벨 구성, 플레이 패턴, 아이템, 스킬 유무, 조작 패턴 등에 대해 어느 정도 감각을 가지고 있다고 가정한다.

- **D-1-1-2.** 모바일 기기를 기반으로 한 게임 중에서 **강렬하고 호쾌한 액션과 타격감을 가진 게임을 원하는 사용자들**

- **D-1-1-3.** **실사 풍의 무겁고 진중한 느낌의 게임을 원하는 사용자들**

- **D-1-1-4.** 성인 남성
  : 상대적으로 **성인 남성들에게 매력적으로 보이고, 이들이 선호하는 컨텐츠 위주**로 제작한다. (폭력성, 선정성...?;;)

◆ D-1-2. 목표 시장 지역

- **D-1-2-1.** 특정한 지역만을 목표 시장으로 두지 않는다.

- **D-1-2-2.** 단, 가장 우선적으로 고려하는 시장은 대한민국(Republic Of Korea)과 중국(China)이다.

D-2. 수익 모델(내적 요소)

◆ D-2-1. 서비스 및 유료화 정책

- **D-2-1-1.** 기본적으로 최종 사용자 고객(클라이언트, 사용자, 플레이어, 게이머 모두 이를 말한다.)이 이 게임을 서비스하는 클라이언트의 응용 프로그램 그 자체를 구매하는 가격은 **무료**로 책정한다.

- **D-2-1-2.** 최종 사용자 고객은 이 게임 서비스를 이용함에 있어, 게임 내에서 유료로 제공하는 재화들을 구매할 수 있다.

- **D-2-1-3.** 재화를 구매함에 있어 1회 당 한계, 반복 횟수, 기간 등의 제한 사항들은, 이 게임을 서비스하는 마켓의 정책과 관련 지역의 법령이 제한하는 바를 따른다.

※ 예를 들면, 1 회당 10 만원 이상의 결제가 불가능하다든가, 1 달에 30 만원 이상의 결제가 불가능한지 등에 대한 법적이거나 계약적인 제한 사항들을 준수하는 범위 내에서 구현해야 한다.

- **D-2-1-4.** 모든 재화들은 반복적인 게임 플레이 중에 조금씩이라도 획득할 수 있다.
  : **게임 내 결제를 이용하지 않으면 절대로 획득하지 못하는 재화는 존재하지 않는다.**

※ 전혀 결제를 하지 않더라도, 게임 컨텐츠를 이용하는데 있어 제한을 두지 않게 하는 것이 목적이다.
결제를 전혀 하지 않았을 때, 획득할 수 없는 재화가 있고, 이 때문에 게임 플레이에 장벽이 생기는 현상은 없어야 한다. 또한, 지나치게 얻을 확률이 낮아서 사실상 획득이 불가능해 보이는 경우 역시 마찬가지다.

- **D-2-1-5.** 각 재화들은 **플레이어 캐릭터의 소지품 가방과 보관함에 보관할 수 있는 아이템의 형태**로 표현할 수 있다.
  : 또한, 그러한 아이템들은 같은 재화의 종류로 가치를 매기더라도, 가격에 따라 여러 종류의 아이템으로 나뉠 수 있다.

※ 이런 경우에는, 아이템을 '판매'하는 행위로써, 재화 보상을 지급받는 형태로 볼 수 있다.
재화 보상을 '직접' 제공해서 즉각적으로 보유한 재화 자산에 더하는 경우도 있을 수 있지만, 이렇게 재화 보상을 아이템 형태로 지급하는 경우에는, 직접적인 재화 보상과는 별도로, 보관함

에 보관하는 아이템 보상들 중 한 자리를 차지하는 형태로 표현할 수도 있는 셈이다.
이런 형태가 필요한 이유 중 하나는, 재화 보상을 무작위한 방식으로 주고 싶을 때 특히 유용하다.

- **D-2-1-6.** 여러 종류의 재화를 하나의 아이템으로 표현하는 것이 가능하다.
: 일반적인 경우는 아니지만, 경우에 따라 이런 기능이 필요할 수 있다. 이를 염두에 두고 설계해야 한다.

※ 이를 테면, 하나의 아이템을 판매했을 때, 사용자에게 돌아가는 재화가 게임 플레이 화폐 2000에 과금 화폐 10개일 수 있다. (2000골드, 10보석)
사실, 이런 건 순전히 표현의 문제일 뿐일 수 있다. 그냥 게임 플레이 화폐 2000 가치의 아이템 1개와 과금 화폐 10개 가치의 아이템 1개를 주는 방식으로 구현해도 관계 없다. 다만, 어떤 아이템의 가치 표현 방식을 좀 더 풍부하게 해준다는 의미로 생각해 볼만 하다.

- **D-2-1-7.** 모든 성장과 강화의 요소들은 **어떤 경우라도 '실패' 하지 않는다.**
: 사용자에게 실패의 경험을 주지 않는다.
상대적으로 더 낮은 가치의 보상을 주거나, 혹은 상대적으로 더 높은 가치의 보상을 주는 방향으로 구성할 뿐, **'실패'해서 플레이어가 손해를 봤다는 느낌이 들지 않게 한다.**

- **D-2-1-8.** 모든 성장과 강화의 요소들은 기본적으로 **수치가 하락하지 않는다.**
: 이 역시 사용자들에게 부정적인 경험을 하지 않게 하기 위한 사항이다.
상대적으로 더 조금이거나, 더 많은 향상의 차이가 존재할 뿐이다.

※ 물론, 이러한 내용이 아이템의 능력치를 비교할 때, 어떤 아이템이 더 좋은지 알려주는 데까지 간섭하지는 않는다.
더 낮은 능력을 가지는 아이템을 착용하고자 할 때는, 당연히 플레이어에게 어떤 능력이 더 하강하는지 알려주는 장치가 있어야 한다.
여기서 말하고자 하는 바는, 플레이어가 아이템을 강화했을 때, 뭔가 저주가 걸려서 아이템의 강화가 더 낮아진다거나, 실패해서 아이템이 공중분해가 된다거나 하는 방식을 두지 않는다는 의미일 뿐이다.

- **D-2-1-9.** 일반적으로 사용자가 **종류를 선택할 수 있는 종류의 재화들은, 무작위한 가운데 추첨해서 결과가 주어지는 재화보다 가격이 비싸다.**
: 선택권이 주어진다는 것 자체가, 사용자가 더 효율적으로 이용할 수 있다는 의미이기 때문에, 그만큼 가치가 더 높아야 한다.

- **D-2-1-10.** 같은 재화 종류 안에서는, **등급이 높은 재화는 등급이 낮은 재화보다 구입 비용, 판매 비용이 비싸다.**
: 경제학적 원리로써 당연한 말이다.

- **D-2-1-11.** 일반적으로 같은 성격의 재화들 중에서는, 사용 조건에 제약이 없는 재화일수록 더 가치가 높으며, 가격도 비싸야 한다.

※ 게임 아이템을 예로 들자면, 같은 정도의 강력함을 가지고 있는 아이템들인데, 한 쪽은 레벨 30 부터 사용할 수 있고, 한 쪽은 레벨 1 부터 자유롭게 사용할 수 있다면, 후자가 더 가치가 높다고 볼 수 있다. 사용 조건에 제약이 덜하기 때문에, 사용자가 더 가치 있게 사용할 수 있기 때문이다. (초보용 스테이지에 압도적으로 강력한 아이템으로 무장했을 때의 플레이 결과가 어떠할지 생각해보면 가치의 차이를 알 수 있다.)

- **D-2-1-12.** 재화의 가치를 구성할 때, **더 높은 가치의 계층들을 나누는 조건들이 조합될수록, 각 조건들의 가치들을 더한 가격 이상의 가격**이 매겨지도록 되어야 한다.
: 모두 따져서 같은 가치를 지니고 있더라도, 하나의 재화로 가능한 경우와 여러 종류의 재화로 가능한 경우가 있다면, 하나의 재화로 가능한 경우가 더 가치가 높다고 봐야 한다. 그만큼 가격도 더 비싸야 한다.

※ 쉽게 말해서, 모든 재화들을 파는 상품들은 결국 **'구매하는 이유'**가 있어야 구매를 할 동기가 생기기 때문에 이러한 방침을 둔다.
더 저렴하고 효율이 좋은 재화가 존재하면, 바보가 아닌 이상 누구라도 그 효율 좋은 재화만 구매할 것이다. 나머지 '효율이 안 좋은' 재화들은 버려질 수 밖에 없다.
그렇더라도 재화의 구매(=결제) 자체가 활발하고 게임 컨텐츠가 유지되는데 별 문제가 없다면 이는 작은 실패에 불과하겠지만, 이 때문에 게임 플레이 컨텐츠가 빠르게 소모되어, 결과적으로 사용자들이 장기적으로 이탈하게 된다면 큰 실패가 된다. 따라서 이런 부분을 주의 깊게 설계해야 한다.

- **D-2-1-13.** 플레이어가 획득할 수 있는 게임 내 자산들 중에서, 게임 내 결제 기능을 이용해야만 획득이 가능한 경우는 오직 **외형에 대한 변경**일 뿐이라야 한다.
: 게임 내 결제를 이용했는지 유무를 통해서, 전투 능력의 강력함이나 지속 시간에 영향을 미치는 요소를 차별적으로 적용해서는 안 된다.

※ 어떤 무기 아이템의 공격력이 월등히 강력한데, 게임 내 결제를 통해서만 획득할 수 있다면, 이 항목의 명세 사항을 위반한 것이다.
그렇지만, 스킬의 이펙트 모양을 바꾸는 경우(능력의 변화 없이), 게임 내 결제를 통해서만 획득할 수 있는 모델도 가능하다.

※ 게임 내 결제를 이용한 혜택은 게임을 플레이 하는 기회를 더 많이 제공하고, 반복 플레이 요소를 줄이기 위한 편의를 제공하는 데 초점을 둬야 한다.
캐릭터의 능력 그 자체에 대해 차별을 두는 방식으로 수익 모델 설계하는 일을 피한다.
또한, 가급적 게임 내 결제를 통해서만 자산을 획득할 수 있는 경우가 없는 게 좋다.

- **D-2-1-14.** 획득 가능한 자산들의 종류를 직접 선택하는 경우와, 무작위한 가운데 뽑아야 하는 경우, 직접 선택하는 경우에 지불해야 할 비용이 더 비싸다.
: 이렇게 해야 더 공평하다. 사용자에게 직접 선택권을 주는 경우, 사용자는 더 효율적으로 그 권리를 사용할 수 있기 때문이다.

◆ D-2-2. VIP 레벨

- **D-2-2-1.** 게임 내 결제를 통해 재화를 구매한 사용자에게, 게임 플레이에 더 많은 편의를 제공하기 위한 근거로 사용하는 레벨이다.

- **D-2-2-2.** 각 사용자 계정의 VIP 레벨에 근거해서, 그 사용자 계정으로 게임 플레이를 진행할 때 부여한 **편의와 혜택의 정도를 차등적으로 둔다.**
: VIP 레벨이 높은 사용자 계정은 VIP 레벨이 낮은 사용자 계정보다 게임 플레이의 편의 기능과 혜택이 더 많이 부여된다.

- **D-2-2-3.** **결제를 통해 구입한 과금 화폐의 개수**를 토대로 하여 VIP 레벨을 향상시킨다.

※ 과금 화폐 그 자체는 게임 플레이 중에도 조금씩 보상으로 획득할 수 있다.
또는 게임 플레이를 장려하기 위한 보상 혜택으로 획득할 수도 있다.
그러나, **결제 행위를 통하지 않은 모든 과금 화폐의 획득은 VIP 레벨의 향상에 어떤 영향도 미치지 않는다.**

- **D-2-2-4.** VIP 레벨은 **사용자의 계정에 종속**된다.
: VIP 레벨을 사용자의 각 캐릭터에 종속시킨다면...?

- **D-2-2-5.** 게임 내 결제를 전혀 이용하지 않은 사용자는 기본적으로 **VIP 0레벨**이다.

- **D-2-2-6.** VIP 레벨의 혜택을 설계할 때, 각 단계는 이전 단계의 모든 혜택 종류를 포함하고 있도록 설계해야 한다.
: 더 높은 VIP 레벨을 취득했을 때, 이전 VIP 레벨에서 가능했던 종류의 편의 기능이나 혜택을 더 이상 받을 수 없는 경우가 생기면 안 된다.

- **D-2-2-7.** VIP 레벨이 높아져도, 하위 VIP 레벨의 모든 혜택들의 수량이 누적되는 것은 아니다.
: **새로운 VIP 레벨의 혜택 수량으로 대체**될 뿐이다.

※ VIP 2 레벨의 혜택 중에서 '일일 던전 입장 제한 1 회 추가 제공'이 있고, VIP 3 레벨의 혜택 중에서 '일일 던전 입장 제한 2 회 추가 제공'이 있다고 치자.

VIP 3 레벨의 혜택은 VIP 2 레벨의 혜택을 포함하고 있지만, 혜택의 '종류'인 '일일 던전 입장 제한의 추가 제공'을 포함하고 있다는 의미일 뿐이다. 제공 횟수 자체는 VIP 3 레벨에 명시된 2 회 제공이지, VIP 2 레벨의 1 회 + VIP 3 레벨의 2 회를 더한 3 회로 계산하는 게 아니다.

- **D-2-2-8.** 각 플레이어의 VIP 레벨은 특별한 경우가 아니면 하락하지 않으며, 영구적으로 유지할 수 있다.

※ 사용자가 이미 획득한 VIP 레벨을 정지하거나 하강해야 하는 경우는 일반적인 게임 서비스 진행 중에 있어서는 안 된다.
서비스 운영 상의 징계 사유 정도가 아니라면 말이다.

※ 결국, VIP 레벨은 사용자가 이 게임에 얼마나 결제를 많이 했는지에 따라 결정된다고 보면 된다.

- **D-2-2-9.** VIP 레벨을 올려서 얻을 수 있는 혜택들의 종류는 다음과 같다.

종류	설명
스테이지 소탕권 지급	• 스테이지를 직접 플레이하지 않고, 완료 결과의 보상만을 획득할 수 있는 권한을 주는 티켓을 받는다. • VIP 레벨이 올라감에 따라, 지급 받는 개수가 더 늘어날 수 있다.
일일 출석 보상의 증가	• 일반적인 경우(VIP 1 레벨)의 출석 보상보다 더 많은 출석 보상을 받는다. • VIP 레벨이 올라감에 따라, 보상의 가치가 점점 더 증가할 수 있다.
스테이지 완료 시 게임 플레이 화폐(골드)의 획득량 증가	• 스테이지를 완료했을 때, 얻을 수 있는 게임 플레이 화폐가 일반적인 경우(VIP 1 레벨)보다 더 많이 받는다. • VIP 레벨이 올라감에 따라, 획득량은 더 증가할 수 있다.
~~희귀 등급 아이템 감정서 지급~~	~~• 전설 등급 아이템 감정서보다 더 낮은 VIP 레벨부터 지급받을 수 있고, 지급받는 개수도 더 많다.~~ ~~• VIP 레벨이 올라감에 따라, 지급 받는 개수가 더 늘어날 수 있다.~~
~~전설 등급 아이템 감정서 지급~~	~~• 희귀 등급 아이템 감정서보다 더 높은 VIP 레벨부터 지급받을 수 있고, 지급받는 개수도 더 적다.~~ ~~• VIP 레벨이 올라감에 따라, 지급 받는 개수가 더 늘어날 수 있다.~~
소탕권의 10 회 연속 사용 가능	• 한 번에 10 회의 소탕권을 쓸 수 있는 메뉴을 사용할 수 있다.

사용자 대전(P vs P)의 대기시간 초기화 사용 가능	・사용자 대전의 대기 시간을 초기화할 수 있는 메뉴를 사용할 수 있다. ・사용자 대전의 대기 시간을 초기화하는 비용은 이것과 별도로 지불해야 한다.
길드 대전(R vs R)의 대기시간 초기화 사용 가능	・길드 대전의 대기 시간을 초기화할 수 있는 메뉴를 사용할 수 있다. ・길드 대전의 대기 시간을 초기화하는 비용은 이것과 별도로 지불해야 한다.
초월 모드 스테이지 대기시간 초기화 사용 가능	・초월 모드 스테이지의 대기 시간을 초기화할 수 있는 메뉴를 사용할 수 있다. ・초월 모드 스테이지의 대기 시간을 초기화하는 비용은 이것과 별도로 지불해야 한다.
숨겨진 상점이 항상 등장하도록 변경	・숨겨진 상점이 무작위한 시간에 등장하거나 사라지지 않고, 항상 사용할 수 있도록 고정적으로 등장한다.
수련장에서 동시에 수련할 수 있는 스킬의 개수 추가	・수련장에서 여러 개의 스킬을 동시에 수련할 수 있게 된다. ・수련할 수 있는 스킬들의 종류는 플레이어가 선택할 수 있다. ・ VIP 레벨이 올라감에 따라, 동시에 수련할 수 있는 스킬의 개수가 늘어날 수 있다.
초월 모드 스테이지의 보상 증가	・초월 모드가 종료되었을 때 얻을 수 있는 보상이 양이 늘어난다. : 이에 대해 정확한 범위가 필요하다. ・ VIP 레벨이 올라감에 따라, 보상의 양이 더 많이 증가할 수 있다.

◆ D-2-3. 월 단위 정액제

- **D-2-3-1.** 결제 상품의 한 형태로써 판매한다.

※ 모바일 마켓에서의 결제 형태 상, 결제를 필요로 하는 상품 아이템을 구매하면, 그 상품 아이템을 일정 개수만큼 지급하는 형태로 표현해야 한다.
즉, 이 경우에는, '실제 시간으로 30 일간 지정된 혜택을 부여하는' 상품 아이템 1 개를 결제해서 구입하는 셈이다. 구입한 상품 아이템은 즉시 사용 처리되어, 사용한 플레이어의 계정이 30 일간 혜택을 받게 된다.
일종의, 사용자 계정에 대한 30 일 간의 부여효과라고 보면 된다.

- **D-2-3-2.** 1회 구매하면, 구매한 사용자에게 구매 시점으로부터 실제 시간으로 30일 동안 사전에 정의한 혜택을 부여한다.

- **D-2-3-3.** 월 단위 정액제의 효과가 종료되는 시점은, 구매일로부터 31일차가 되는 날의 서비스 지역 표준시로 오전 6:00시 이후이다.

- **D-2-3-4.** 월 단위 정액제 상품을 결제한 내역은, 결제한 사용자의 VIP 레벨 계산에 포함한다.

※ VIP 레벨은 사용자가 이 게임에 대해서 결제한 모든 내역에 대해 **예외 없이 합산**해서 적용해야 한다.

- **D-2-3-5.** 월 단위 정액제 상품을 이용하는 동안 받는 혜택은 다음과 같다.
: 이 혜택은, 상품을 결제한 사용자의 VIP 레벨에 따른 혜택과는 별도로 추가해서 주어진다.

종류	설명
매일 과금 화폐 제공	• 하루에 1 회 제공한다. • 횟수가 초기화되는 시점은 서비스 지역에서 사용하는 시간대의 0 시 기준이다.

- **D-2-3-6.** 월 단위 정액제 상품은 **연속적으로 결제할 수 없다.**
: 한 번에 몇 달치에 해당하는 정액제 상품을 미리 결제할 수 없다. 월 단위 정액제 상품을 이미 결제한 사용자는, 같은 상품에 대해 결제를 시도할 수 없다..

- **D-2-3-7.** 월 단위 정액제 상품을 결제한 사용자는, 그 즉시 상품의 사용 유효 기간이 얼마나 남았는지에 대해 알림을 받는다.

- **D-2-3-8.** 이미 월 단위 정액제를 결제한 사용자의 계정은, 월 단위 정액제 상품의 효과가 종료되기 3일 전, 서비스 지역 표준시로 오전 6:00 이후부터 새로운 월 단위 정액제 상품의 결제가 가능해진다.

※ 그러므로, 월 단위 정액제의 효과가 유효한 날 수를 계산할 때는 30 일을 초과한 상태도 가능하도록 염두에 두고 구현해야 한다. (남은 날 수는 최대 32 ~ 33 일까지 가능할 수 있기 때문이다.)

◆ D-2-4. 일일 사용(또는 이용) 가능 횟수

- **D-2-4-1.** 게임 서비스에 제공되는 컨텐츠 중 일부는, 하루에 이용하거나 사용할 수 있는 횟수가 제한되어 있다.

※ 이러한 정책은 상위 게임 컨텐츠의 소모 속도를 조절하기 위한 목적이 크다.

하루에 사용 / 이용 가능한 횟수가 제한되는 컨텐츠들은 특정한 종류의 재화만을 표적 방식으로 획득할 수 있거나, 보상의 종류나 질이 높은 경우가 많다.
이러한 컨텐츠들을 이용하는 데 아무런 제한도 없다면, 소위 '작업장'과 같은 방식의 플레이가 가능할 수 있다. 이런 방식의 플레이를 하는 사용자들은 순식간에 게임의 모든 컨텐츠들을 소모해버리고, 가능한 모든 순위들을 독식하며 다른 '일반적인' 게이머들과의 격차를 돌이킬 수 없을 정도로 벌려 놓을 것이다.
이런 현상이 지속되면, 신규 게이머나 상대적으로 가볍게 플레이 하는 게이머들이 그들을 어느 정도 따라잡을 수 있도록 추가적인 레벨 디자인, 새로운 컨텐츠 등을 지속적으로 공급해야 할 것이다.
게임 컨텐츠의 사용 횟수 제한이 이와 같은 '경제적인 격차'를 완전히 해소할 수는 없다고 본다. 비록 그렇더라도, 서비스를 운영하는 측에서는 완전한 '시장 자유인 상태'보다는 상대적으로 감당하기 쉬운 상태로 만들어 줄 수 있으리라 예상한다.

- **D-2-4-2.** 일일 사용(또는 이용) 가능한 횟수와 조건이 초기화되는 시점은, **게임을 서비스하는 지역의 표준시로 매일 오전 6:00**이다.

- **D-2-4-3.** 일일 사용(또는 이용) 가능한 횟수는 매일 초기화할 때, 그 전날의 미사용 횟수와 누적되지 않는다. 언제나 항상 지정된 횟수로 초기화한다.

- **D-2-4-4.** 사용자 계정의 VIP 레벨을 올리는 정도에 따라, 해당하는 컨텐츠들의 일일 사용 가능한 최대 횟수가 늘어날 수 있다.

- **D-2-4-5.** 컨텐츠의 일일 사용(또는 이용)의 횟수를 늘리는 구체적인 방식은 다음과 같다.

종류	설명
최대 사용 횟수 증가 (C - Capacity)	• 하루에 사용 / 이용할 수 있는 최대 한도의 값을 늘리는 방식이다. • 기본적으로 이용 가능한 횟수를 넘어섰을 때, 다른 부가적인 비용 조건을 줄 수도 있다.
사용 횟수 초기화 권한 (R - Reset)	• 현재까지 사용했던 일일 사용(또는 이용) 횟수를, 전혀 사용한 적이 없는 상태처럼 즉시 초기화한다. • 초기화를 수행하는 과정에서는 부가적인 비용 조건을 줄 수 있지만, 초기화된 이후의 사용 과정은 일반적인 경우와 똑 같은 조건으로 진행된다. : 일단 초기화된 이후에는 '자연적인' 이용 횟수 초기화 상태와 구분할 수 없다.

- **D-2-4-6.** VIP 레벨에 의해 일일 사용 가능한 최대 횟수가 늘어났더라도, **하루에 기본 제공되는 횟수에는 변함이 없다.**
: 즉, 일일 사용 가능한 횟수는, **하루에 무료로 사용 / 이용할 수 있는 횟수인** 셈이다.

※ 즉, VIP 레벨의 의해 일일 사용 가능한 최대 횟수를 늘리는 혜택은, 그 해당하는 게임 컨텐츠를 **이용할 수 있는 기회의 한도를 추가적으로 제공**해 줄 뿐이다.
컨텐츠의 추가 이용에 대한 비용까지 무료로 제공해주는 방식이 아니다.
VIP 레벨의 여부와 관계 없이, 하루에 이용 가능한 횟수를 전부 사용했다면, 어떤 방식으로도 다음 날이 되기 전에는 같은 컨텐츠를 다시 이용할 수 없게 된다.

- **D-2-4-7.** 하루에 무료로 사용하거나 이용할 수 있는 횟수를 모두 소모한 이후의 추가적인 사용에 대해서는, 그 **추가적인 사용 / 이용 1회마다 그에 대한 별도의 비용을 지불**해야 한다.
: 이러한 비용은, 그 컨텐츠를 이용하기 위해 기본적으로 지불하는 비용과 별도이다.

- **D-2-4-8.** 일일 사용(또는 이용) 횟수가 제한된 컨텐츠의 추가적인 사용에 대한 추가 비용은 과금 화폐를 단위로 요금을 정한다.

※ 위 두 항목을 같이 엮어서 생각해보면 아래와 같은 상황이 어떻게 처리되는지 알 수 있다.
하루에 3 회 입장이 제한된 던전이 있다. 그리고 이 던전은 1 회 입장할 때마다 입장 자원을 5 씩 소모한다고 가정하자.
그런데, 플레이어가 결제를 통해 VIP 레벨을 올렸고, 그 혜택에 의해 던전의 하루 입장 제한 횟수가 3 회에서 5 회로 증가했다. 그러면 이 상황에서, 던전의 일일 사용 횟수는 다음처럼 동작한다.
비록 VIP 레벨에 의해 하루 입장 제한 횟수가 5 회로 늘었지만, 매일 초기화되는 던전 입장 가능 횟수는 여전히 3 회씩이다.
나머지 VIP 레벨 혜택으로 추가된 2 회의 입장 가능 횟수가 있지만, 이 때부터는 1 회씩 입장할 때마다 과금 화폐를 40 을 지불하고 추가 입장 권한을 사야 한다. 물론, 입장할 때 입장 자원이 5 씩 소모되는 점은 똑같다.
그렇게 해서 추가적으로 던전 입장을 2 회 더 했으면, 하루에 사용 가능한 입장 횟수 5 회를 모두 채우게 되어, 다음 날이 되어 던전 입장 조건이 초기화될 때까지 어떤 수단으로도 던전에 또 다시 입장할 수 없다.

- **D-2-4-9.** 일일 사용(또는 이용) 가능 횟수를 가지는 컨텐츠의 종류와 그 세부적인 특성들은 다음과 같다.

종류	일일 사용 / 이용 횟수	추가 사용 / 이용의 방식	설명
게임 플레이 화폐 구입	1	C	• VIP 레벨에 따라 그 횟수가 늘어날 수 있다.
입장 자원 구입	1	C	• VIP 레벨에 따라 그 횟수가 늘어날 수 있다.

스토리 모드 스테이지(정예) 입장	3	R	• VIP 레벨에 따라 그 횟수가 늘어날 수 있다. • 선택한 스테이지의 이용 횟수만 초기화된다. : 플레이어가 선택을 하게 만든다.
~~일일 던전 입장~~	~~5~~	~~R~~	~~• VIP 레벨에 따라 그 횟수가 늘어날 수 있다. • 제한된 횟수 내에서 입장할 일일 던전을 골라서 입장하는 방식이다.~~
주간 던전 입장	3	R	• VIP 레벨에 따라 그 횟수가 늘어날 수 있다. • 입장 가능한 던전 지역이 2 개 이상이 된 경우, 제한된 횟수 내에서 골라서 입장하는 방식이다.
혼돈 던전 입장	3	C	• VIP 레벨에 따라 그 횟수가 늘어날 수 있다.
초월 던전 입장	3	C	• VIP 레벨에 따라 그 횟수가 늘어날 수 있다.
결투장 입장	5	C	• VIP 레벨에 따라 그 횟수가 늘어날 수 있다. • 입장할 때마다, 재사용 대기 시간이 있다.
길드 던전 입장	2	C	• VIP 레벨에 따라 그 횟수가 늘어날 수 있다.
길드 전쟁 입장	2	C	• VIP 레벨에 따라 그 횟수가 늘어날 수 있다. • 입장할 때마다, 재사용 대기 시간이 있다.
숨겨진 상점의 물품 목록 새로고침	5	C	• VIP 레벨에 따라 그 횟수가 늘어날 수 있다. • 새로고침을 시도할 때마다, 다음 번 새로고침하는 비용이 증가한다.
결투장 상점의 물품 목록 새로고침	5	C	• VIP 레벨에 따라 그 횟수가 늘어날 수 있다. • 새로고침을 시도할 때마다, 다음 번 새로고침하는 비용이 증가한다.
길드 상점의 물품 목록 새로고침	5	C	• VIP 레벨에 따라 그 횟수가 늘어날 수 있다. • 새로고침을 시도할 때마다, 다음 번 새로고침하는 비용이 증가한다.
무작위 뽑기 (무료)	5	없음	• 무료 사용인 경우에만 5 회이다. : 유료로 사용하면 이용 횟수의 제한이 없다. • 하루에 사용할 수 있는 무료 뽑기 횟수를 늘리는 혜택이나 보너스 컨텐츠는 없다. • 사용할 때마다 재사용 대기 시간이 있다.

◆ D-2-5. 게임 컨텐츠의 재사용 대기 시간

- **D-2-5-1.** 일부 게임 컨텐츠들은, 사용자의 접속 여부나, 현재 사용자 계정의 캐릭터가 플레이하고 있는 장면에 상관 없이 돌아가는 재사용 대기 시간을 가지는 경우가 있다.
: 클라이언트 응용 프로그램을 종료하더라도, 재사용 대기 시간은 여전히 돌아간다.

- **D-2-5-2.** 컨텐츠를 1회 이용할 때마다 재사용 대기 시작이 초기화된다.
 : 재사용 대기 시간이 지나기 전에는 다시 같은 컨텐츠를 사용하거나 이용할 수 없다.

- **D-2-5-3.** 이러한 컨텐츠들의 재사용 대기 시간은 서버에서 관리한다.
 : 클라이언트의 접속이 이루어지지 않은 상태에서도 시간을 관리해야 하기 때문에, 서버에서 관리해야 한다.

※ 게임 스테이지에서 플레이 중인 각 캐릭터들이 사용하는 스킬, 부여효과 등의 재사용 대기 시간은, 여기서 말하고 있는 컨텐츠의 재사용 대기 시간과 명백히 다른 범주에 속한다는 점에 주의해야 한다.
 게임 스테이지에서 플레이 중인 캐릭터들의 스킬과 부여효과 등은 그 스테이지를 빠져나오면 재사용 대기시간이든 뭐든, 그 스테이지의 게임 플레이 연산에 대한 모든 사항이 초기화된다.
 이 항목에서 설명하는 게임 컨텐츠의 재사용 대기 시간은 플레이어의 캐릭터가 로비에 있든, 게임 플레이 스테이지에 있든, 길드 전쟁에 들어가 있든, 심지어 응용 프로그램을 종료했든 상관없이 언제나 작동하는 재사용 대기 시간이다.

- **D-2-5-4.** 각 게임 컨텐츠의 재사용 대기 시간은, 해당 사용자 계정의 캐릭터마다 별도로 관리한다.
 : 사용자 계정 단위로 재사용 대기 시간을 관리해야 하는 컨텐츠가 이론적으로 불가능하지는 않다. 단지, 현재 설계 중인 게임 컨텐츠에는 그런 방식에 해당하는 기능이 없을 뿐이다.

- **D-2-5-5.** 일일 사용(또는 이용)의 횟수 제한과, 컨텐츠의 재사용 대기 시간은 서로 혼용해서 사용할 수 있다.
 : 이 경우, 컨텐츠를 1회 사용 / 이용할 때마다 재사용 대기 시간이 존재하고, 그 재사용 대기 시간이 끝나기 전에는 다시 같은 컨텐츠를 사용할 수 없다.
 또한, 일일 사용(또는 이용)의 제한 횟수에 도달해도 해당 컨텐츠를 더 사용할 수 없다.

- **D-2-5-6.** 재사용 대기 시간이 존재하는 컨텐츠와 그 세부적인 특성들은 다음과 같다.

종류	재사용 대기시간	설명
결투장 입장	600 초	• 하루에 입장 가능한 횟수가 존재한다.
길드 전쟁 입장	600 초	• 하루에 입장 가능한 횟수가 존재한다.
무작위 뽑기 (무료)	300 초	• 무료로 뽑는 경우에만 재사용 대기 시간이 존재한다. • 하루에 무료로 사용할 수 있는 횟수가 제한되어 있다.
숨겨진 상점의 판매 목록 변경	6 시간	• 숨겨진 상점이 항상 등장하는 상태일 때만 동작한다. : 그게 아닌 경우에는 1 시간만 유지하기 때문에 재사용 대기 시간이 있을 의미가 없다.

초월 모드 상점의 판매 목록 변경	6 시간	• 판매 목록을 새로고침 하려면 초월 모드 코인으로 비용을 지불한다.
결투장 상점의 판매 목록 변경	6 시간	• 판매 목록을 새로고침 하려면 결투장 코인으로 비용을 지불한다.
길드 상점의 판매 목록 변경	6 시간	• 판매 목록을 새로고침 하려면 길드 코인으로 비용을 지불한다.

- **D-2-5-7.** VIP 레벨의 혜택에 따라, 클라이언트에서 재사용 대기 시간을 초기화하도록 요청할 수 있는 권한을 얻을 수 있다.

※ 일일 사용(또는 이용) 횟수의 제한을 늘리거나 초기화하는 것과 마찬가지의 개념이다.
아예 불가능한 상태로부터, 가능한 상태가 된다는 의미이며, 재사용 대기 시간의 초기화 명령에 드는 비용은 별도로 지불해야 할 수 있다.

- **D-2-5-8.** 컨텐츠의 재사용 대기 시간을 초기화하는 횟수는 제한적일 수 있다.
: VIP 레벨의 혜택에 따라 이 제한 횟수가 증가할 수 있다.

- **D-2-5-9.** 컨텐츠의 재사용 대기 시간을 초기화할 때, 이 초기화를 위한 비용을 지불해야 하는 경우와 세부적인 특성들은 다음과 같다.

종류	초기화 비용	설명
결투장 입장	과금 화폐	• 재사용 대기 시간을 초기화 가능한 횟수가 제한되어 있다. • VIP 레벨에 따라 그 횟수가 늘어날 수 있다. • 재사용 대기 시간을 초기화할 때마다, 다음 번 재사용 대기 시간을 초기화하는 비용이 증가한다.
길드 전쟁 입장	과금 화폐	• 재사용 대기 시간을 초기화 가능한 횟수가 제한되어 있다. • VIP 레벨에 따라 그 횟수가 늘어날 수 있다. • 재사용 대기 시간을 초기화할 때마다, 다음 번 재사용 대기 시간을 초기화하는 비용이 증가한다.
무작위 뽑기 (무료)	과금 화폐	• 재사용 대기 시간을 초기화할 수 없다. : 무료로 뽑기할 수 있는 경우에만 재사용 대기 시간이 있다. • 재사용 대기 시간 없이 요금을 내고 즉각 무작위 뽑기를 할 수 있는 메뉴를 같이 제공한다.
숨겨진 상점의 판매 목록 변경	과금 화폐	• 숨겨진 상점이 항상 등장하는 상태일 때만 동작한다. : 그게 아닌 경우에는 1 시간만 유지하기 때문에 재사용 대기 시간이 있을 의미가 없다. • 재사용 대기 시간을 초기화 가능한 횟수가 제한되어 있다. • VIP 레벨에 따라 그 횟수가 늘어날 수 있다.

		• 재사용 대기 시간을 초기화할 때마다, 다음 번 재사용 대기 시간을 초기화하는 비용이 증가한다.
초월 모드 상점의 판매 목록 변경	초월 모드 화폐	• 재사용 대기 시간을 초기화 가능한 횟수가 제한되어 있다. • VIP 레벨에 따라 그 횟수가 늘어날 수 있다. • 재사용 대기 시간을 초기화할 때마다, 다음 번 재사용 대기 시간을 초기화하는 비용이 증가한다.
결투장 상점의 판매 목록 변경	결투장 코인	• 재사용 대기 시간을 초기화 가능한 횟수가 제한되어 있다. • VIP 레벨에 따라 그 횟수가 늘어날 수 있다. • 재사용 대기 시간을 초기화할 때마다, 다음 번 재사용 대기 시간을 초기화하는 비용이 증가한다.
길드 상점의 판매 목록 변경	길드 코인	• 재사용 대기 시간을 초기화 가능한 횟수가 제한되어 있다. • VIP 레벨에 따라 그 횟수가 늘어날 수 있다. • 재사용 대기 시간을 초기화할 때마다, 다음 번 재사용 대기 시간을 초기화하는 비용이 증가한다.

◆ D-2-6. 유료 컨텐츠의 무료 이용

- **D-2-6-1.** 게임 내 재화를 비용으로 지불해야 하는 컨텐츠들의 전체, 혹은 일부는 플레이어에게 제한된 횟수만큼 무료로 이용할 수 있는 권한을 준다.
  : 이는 일종의 '맛보기'와 같은 방식이다.

※ 비용을 필요로 하는 컨텐츠의 무료 제공을 통해, 플레이어들은 해당 컨텐츠의 필요성과 이득을 미리 체험할 수 있다. 이러한 체험은 플레이어들에게 '유료 컨텐츠'에 대한 욕구를 자극하게 하고, 결과적으로 좀 더 게임 내 결제를 적극적으로 하도록 유도할 수 있다.

- **D-2-6-2.** **유료 컨텐츠들을 무료로 이용하는 권한은 정기적 / 비정기적으로 지급**한다.
  : 지속적으로 유료화 컨텐츠의 기능과 이득을 사용자에게 경험하게 해주는 데 초점을 맞춘다.

※ 유료화 컨텐츠를 이용할 수 있는 기회를 너무 많이 주는 경우에는 사실상 무료화 컨텐츠가 되어 버리는 효과가 날 수도 있다는 점을 우려할 수도 있다.
하지만, 일부 그런 면이 있다고 할지라도, **적극적으로 플레이하는 게임 이용자들의 확보가 항상 더 우선되어야 한다.** 왜냐하면, 어떤 수익 모델이라도 결국 '규모의 경제'가 되지 않으면 충분한 수익을 달성하기가 어렵기 때문이다.

- **D-2-6-3.** 유료 컨텐츠를 무료로 이용할 때는, 그 횟수와 1회 사용시 재사용 대기 시간이 존재할 수 있다.
  : 유료로 같은 컨텐츠를 이용할 때는, 상대적으로 제한이 완화되거나, 아예 사용에 대한 제한이 없다.

※ 무료 사용 기회를 주더라도 결국 제한된 횟수, 제한된 재사용 대기 시간이 필요하기 때문에, 결제를 하지 않는 사용자들은 그만큼 게임 플레이 그 자체에 시간을 더 들여야 한다. 그리고, 그렇게 게임 플레이에 시간을 들이는 사용자가 많아질수록, 게임 내 결제를 이용할 확률은 높아진다. **(일단 게임을 플레이를 해야 결제를 하든지 말든지 할 것 아닌가?)**

- **D-2-6-4.** **게임 내에서 결제를 전혀 하지 않는 사용자들도, 유료 컨텐츠들을 이용할 수 있는 재화들을 조금씩 획득**할 수 있어야 한다.

※ 이러한 정책은, 일종의 잠재적인 고객을 확보하는 전략이다.
유료 컨텐츠를 이용하기 위한 재화들을 획득하는 수단이 오직 유료 결제 밖에 없다면, 대다수의 사용자들(약 90 ~ 95%?)은 해당 컨텐츠를 경험해 볼 의지 자체를 박탈당하게 되며, 아예 그냥 게임 플레이 자체에 흥미를 잃을 수 있다. ~~(아무리 노력해도 꿈도 희망도 없는 게임...)~~
요는, 게임 내 결제를 하지 않는 사용자들은 **돈 대신 시간**을 들여서 유료 컨텐츠를 즐기는 셈이다.
게임 내 결제를 적극적으로 하는 사용자들은, **시간 대신 돈**을 들여서 유료 컨텐츠를 더 즉각적으로, 더 제한 없이 즐긴다고 보면 된다.

- **D-2-6-5.** 유료화 컨텐츠를 무료로 이용하는 기회가 주어지는 경우, **무료 이용 기회의 횟수를 가장 우선적으로 소진**시킨다.

- **D-2-6-6.** 유료화 컨텐츠를 무료로 이용하는 경우와, 유료로 이용해야 하는 경우가 공존해야 할 때는, 이를 별도의 GUI로 두어서, 사용자가 선택할 수 있게 해야 한다.
: 특히, 무료로 이용하면 1회 이용할 때마다 재사용 대기 시간이 존재하는 컨텐츠들은 이런 방식으로 GUI를 설계하도록 고려해야 한다.

※ 재사용 대기 시간을 초기화하는 방식이 아닌 경우라면, 재사용 대기 시간 없이 즉각적으로 컨텐츠를 이용하고 싶은 사용자들을 위한 메뉴가 존재해야 한다는 뜻이다.
물론, 재사용 대기 시간과 같은 제한이 없어진 만큼, 그에 대한 비용을 지불해야 한다.

- **D-2-6-7.** 유료화 컨텐츠를 무료로 이용하는 경우와 그 세부적인 특성들은 다음과 같다.

종류	설명
과금 화폐 획득	• 정기적인 접속 보상으로 획득할 수 있다. • 일일 미션 달성의 보상으로 획득할 수 있다. • 업적을 달성한 보상으로 획득할 수 있다. • 서비스 운영 과정에서 비정기적으로 직접 지급받을 수도 있다. • 무작위 뽑기를 통해서, 과금 화폐를 지급하는 상품에 당첨될 수도 있다.

게임 플레이 화폐 획득	・게임 스테이지의 보상으로 획득할 수 있다. (가장 흔한 경우임) ・일일 미션 달성의 보상으로 획득할 수 있다. ・보유한 장비 아이템, 소모성 아이템, 재료, 보석 등의 판매로 획득할 수 있다. ・판매 가격이 매우 높은 아이템의 형태로 지급할 수 있다. ・무작위 뽑기를 통해서, 게임 플레이 화폐를 지급하는 상품에 당첨될 수도 있다.
입장 자원 획득	・정기적인 접속 보상으로 획득할 수 있다. ・일일 미션 달성의 보상으로 획득할 수 있다. ・서비스 운영 과정에서 비정기적으로 직접 지급받을 수도 있다. ・무작위 뽑기를 통해서, 과금 화폐를 지급하는 상품에 당첨될 수도 있다.
던전 소탕권 획득	・ VIP 레벨에 따라, 매일 지급될 수 있다. ・일일 미션 달성의 보상으로 획득할 수 있다. ・업적을 달성한 보상으로 획득할 수 있다. ・서비스 운영 과정에서 비정기적으로 지급받을 수도 있다. ・무작위 뽑기를 통해서, 과금 화폐를 지급하는 상품에 당첨될 수도 있다.
장비 아이템 뽑기권 획득	・고급 장비 뽑기권은 무료로 이용할 수 없다. ・일일 미션 달성의 보상으로 획득할 수 있다. ・업적을 달성한 보상으로 획득할 수 있다. ・서비스 운영 과정에서 비정기적으로 지급받을 수도 있다. ・무작위 뽑기를 통해서, 과금 화폐를 지급하는 상품에 당첨될 수도 있다.
스킬 룬 뽑기권 획득	・타입 별 스킬 룬 뽑기권은 무료로 이용할 수 없다. ・일일 미션 달성의 보상으로 획득할 수 있다. ・업적을 달성한 보상으로 획득할 수 있다. ・서비스 운영 과정에서 비정기적으로 지급받을 수도 있다. ・무작위 뽑기를 통해서, 과금 화폐를 지급하는 상품에 당첨될 수도 있다.
재료 아이템 획득	・게임 스테이지의 보상으로 획득할 수 있다. ・일일 미션 달성의 보상으로 획득할 수 있다. ・업적을 달성한 보상으로 획득할 수 있다. ・서비스 운영 과정에서 비정기적으로 지급받을 수도 있다. ・무작위 뽑기를 통해서, 과금 화폐를 지급하는 상품에 당첨될 수도 있다. ・일부 재료들은 지정된 스테이지 종류에서만 획득할 수 있다.
아이템 강화 보석 획득	・게임 스테이지의 보상으로 획득할 수 있다. ・일일 미션 달성의 보상으로 획득할 수 있다.

	• 업적을 달성한 보상으로 획득할 수 있다. • 서비스 운영 과정에서 비정기적으로 지급받을 수도 있다. • 무작위 뽑기를 통해서, 과금 화폐를 지급하는 상품에 당첨될 수도 있다. • 일부 상위 등급의 보석들은 무료로 획득할 수 없을 수 있다.
회복 수단 획득	• 게임 스테이지의 보상으로 획득할 수 있다. • 일일 미션 달성의 보상으로 획득할 수 있다. • 업적을 달성한 보상으로 획득할 수 있다. • 서비스 운영 과정에서 비정기적으로 지급받을 수도 있다. • 무작위 뽑기를 통해서, 과금 화폐를 지급하는 상품에 당첨될 수도 있다.
아이템 강화석 획득	• 게임 스테이지의 보상으로 획득할 수 있다. • 일일 미션 달성의 보상으로 획득할 수 있다. • 업적을 달성한 보상으로 획득할 수 있다. • 서비스 운영 과정에서 비정기적으로 지급받을 수도 있다. • 무작위 뽑기를 통해서, 과금 화폐를 지급하는 상품에 당첨될 수도 있다.
스킬 강화석 획득	• 게임 스테이지의 보상으로 획득할 수 있다. • 일일 미션 달성의 보상으로 획득할 수 있다. • 업적을 달성한 보상으로 획득할 수 있다. • 서비스 운영 과정에서 비정기적으로 지급받을 수도 있다. • 무작위 뽑기를 통해서, 과금 화폐를 지급하는 상품에 당첨될 수도 있다.
아이템 초월석 획득	• 게임 스테이지의 보상으로 획득할 수 있다. • 일일 미션 달성의 보상으로 획득할 수 있다. • 업적을 달성한 보상으로 획득할 수 있다. • 서비스 운영 과정에서 비정기적으로 지급받을 수도 있다. • 무작위 뽑기를 통해서, 과금 화폐를 지급하는 상품에 당첨될 수도 있다.
장비 아이템 마법 부여권 획득	• 일일 미션 달성의 보상으로 획득할 수 있다. • 업적을 달성한 보상으로 획득할 수 있다. • 서비스 운영 과정에서 비정기적으로 지급받을 수도 있다. • 무작위 뽑기를 통해서, 과금 화폐를 지급하는 상품에 당첨될 수도 있다.
~~장비 아이템 감정서 획득~~	~~• 일일 미션 달성의 보상으로 획득할 수 있다.~~ ~~• 업적을 달성한 보상으로 획득할 수 있다.~~ ~~• 서비스 운영 과정에서 비정기적으로 지급받을 수도 있다.~~ ~~• 무작위 뽑기를 통해서, 과금 화폐를 지급하는 상품에 당첨될 수도 있다.~~

무작위 무료 뽑기권 획득	• 매일 일정한 횟수를 획득할 수 있다. • 서비스 운영 과정에서 비정기적으로 지급받을 수도 있다.

- **D-2-6-8.** 이 재화를 비용으로 소모하는 컨텐츠와 그 세부적인 특성들은 다음과 같다.

종류	설명
게임 플레이 화폐 구매	• 게임 플레이 화폐는 게임 내부의 가장 많은 컨텐츠에서 비용을 지불하는 수단으로 사용한다. • 게임 플레이 화폐가 모자라는 경우, 과금 화폐가 충분히 있다면, 노력을 들일 필요 없이 대량의 게임 플레이 화폐를 구입할 수 있다.
입장 자원 구매	• 사용자 간 대전하는 기능을 제외한 모든 게임 플레이 스테이지들은 입장할 때마다 입장 자원이 소모된다. • 입장 자원이 추가적으로 필요한 경우, 과금 화폐가 충분히 있다면, 노력을 들일 필요 없이 대량의 입장 자원을 충전할 수 있다.
스테이지 보상 단계에서 한 번 더 뽑기	• 스테이지를 완료하면, 여러 개의 뽑기 상자들 중에서 무작위로 하나를 선택해서, 선택한 상자에서 등장하는 보상을 받는다. • 그러한 뽑기의 결과가 마음에 들지 않을 때, 나머지 상자들만 가지고 추가적으로 뽑기를 할 수 있다.
사망 시, 스테이지 내에서 즉시 부활하기	• 스테이지를 진행하다가 사망해서 빠져나가면, 그 스테이지의 경험치와 보상을 얻을 수 없다. • 비용을 내고 즉시 부활하면, 플레이어의 캐릭터는 완전히 회복된 상태로 부활하여, 기존에 플레이하던 스테이지를 계속 진행할 수 있다.
장비 아이템 뽑기	• 뽑기의 종류에 따라 비용이 다르다. • 선택이 무작위적일수록 비용이 저렴하고, 무작위 범위가 줄어들수록 수행 비용이 비싸진다.
스킬 룬 뽑기	• 뽑기의 종류에 따라 비용이 다르다. • 선택이 무작위적일수록 비용이 저렴하고, 무작위 범위가 줄어들수록 수행 비용이 비싸진다.

◆ D-2-7. 과금 화폐의 순환 흐름

- **D-2-7-1.** 이 재화의 획득 방식은 다음과 같은 의도로 이루어진다.

종류	설명
일상적인 획득	• 기본적으로 획득하려면 결제가 필요한 재화 단위이지만, 결제를 하지 않는다고 해서 아예 획득할 수 있는 기회가 없는 건 아니다. • 게임 스테이지를 완료했을 때의 보상으로는 등장하지 않는다.

	・단, 게임 플레이 활동 중에 비정기적으로 수집할 수 있는 기회를 준다 ・주기적인 접속 보상, 게임 내 활동 장려 정책, 순위 상승 보상, 비정기적 운영 이벤트에 의해 주로 획득할 수 있다. ・이 활동으로 획득할 수 있는 재화의 양은 매우 적다.
결제를 통한 구입	・즉각적으로 대량의 재화를 획득할 수 있다. ・수행할 때마다 VIP 점수와 레벨이 상승하기 때문에, 게임 플레이에 추가적인 혜택을 받을 수 있다.

- **D-2-7-2.** 이 재화를 비용으로 소모하는 컨텐츠와 그 세부적인 특성들은 다음과 같다.

종류	설명
게임 플레이 화폐 구매	・게임 플레이 화폐는 게임 내부의 가장 많은 컨텐츠에서 비용을 지불하는 수단으로 사용한다. ・게임 플레이 화폐가 모자라는 경우, 과금 화폐가 충분히 있다면, 노력을 들일 필요 없이 대량의 게임 플레이 화폐를 구입할 수 있다.
입장 자원 구매	・사용자 간 대전하는 기능을 제외한 모든 게임 플레이 스테이지들은 입장할 때마다 입장 자원이 소모된다. ・입장 자원이 추가적으로 필요한 경우, 과금 화폐가 충분히 있다면, 노력을 들일 필요 없이 대량의 입장 자원을 충전할 수 있다.
스테이지 보상 단계에서 한 번 더 뽑기	・스테이지를 완료하면, 여러 개의 뽑기 상자들 중에서 무작위로 하나를 선택해서, 선택한 상자에서 등장하는 보상을 받는다. ・그러한 뽑기의 결과가 마음에 들지 않을 때, 나머지 상자들만 가지고 추가적으로 뽑기를 할 수 있다.
사망 시, 스테이지 내에서 즉시 부활하기	・스테이지를 진행하다가 사망해서 빠져나가면, 그 스테이지의 경험치와 보상을 얻을 수 없다. ・비용을 내고 즉시 부활하면, 플레이어의 캐릭터는 완전히 회복된 상태로 부활하여, 기존에 플레이하던 스테이지를 계속 진행할 수 있다.
장비 아이템 뽑기	・뽑기의 종류에 따라 비용이 다르다. ・선택이 무작위적일수록 비용이 저렴하고, 무작위 범위가 줄어들수록 수행 비용이 비싸진다. ・좋은 장비 아이템을 빠른 시간 안에 얻을 수 있게 한다.
스킬 룬 뽑기	・뽑기의 종류에 따라 비용이 다르다. ・선택이 무작위적일수록 비용이 저렴하고, 무작위 범위가 줄어들수록 수행 비용이 비싸진다. ・좋은 스킬 룬을 빠른 시간 안에 얻을 수 있게 한다.
무작위 뽑기	・뽑기 비용에 대한 종류는 없고, 모두 같은 가격이다. ・뭐가 등장할지 모르는, 복권 같은 개념이다.

	・투입한 비용에 미치지 못하는 결과를 얻을 수도 있으나, 적은 비용을 들여서 큰 이득을 볼 수 있는 가능성도 존재한다.
장비 아이템 마법 부여	・바꿀 능력 옵션의 부위를 사용자가 선택할 수 있는 마법 부여는 가격을 더 비싸게 책정한다. ・획득한 아이템에서 전체적으로, 혹은 일부 마음에 들지 않는 능력 옵션들을 다른 원하는 능력 옵션으로 바꿀 수 있는 기회를 얻는다.
아이템 형상 변환	・형상을 변경하더라도, 아이템의 능력은 변하지 않는다. ・희귀한 형상을 얻을 수 있거나, 변환할 형상을 선택할 수 있는 경우에 더 비용을 비싸게 책정된다. ・아이템의 외형을 플레이어가 원하는 외형으로 바꿀 수 있는 기회를 준다. ・남들과는 다른 아이템 외형을 보유할 수 있다.
스킬 형상 변환	・형상을 변경하더라도, 아이템의 능력은 변하지 않는다. ・희귀한 형상을 얻을 수 있거나, 변환할 형상을 선택할 수 있는 경우에 더 비용을 비싸게 책정된다. ・아이템의 외형을 플레이어가 원하는 외형으로 바꿀 수 있는 기회를 준다.
숨겨진 상점에 등장하는 물품 구매	・숨겨진 상점에서 판매하는 물품의 가격 단위가 될 수 있다. ・일반적으로 구하기 어려운 재화들을 구할 수 있는 기회가 된다.
숨겨진 상점의 물품 목록 새로 고침	・새로 고침을 할 때마다 지불하는 비용이 증가한다. ・새로 고침을 할 수 있는 최대 횟수는 제한되어 있다. ・숨겨진 상점에서 원하는 물품들이 등장할 때까지, 제한적이나마 목록을 바꿀 수 있다. ・숨겨진 상점에서 판매하는 물품들을 여러 번 싹쓸이(...)할 수 있다.
스토리 모드(정예)의 일일 횟수 초기화	・일일 횟수 초기화는 선택한 대상의 스테이지에만 적용된다. : 스테이지 별로 초기화 가능 횟수가 따로 차감된다. ・같은 스테이지를 여러 번 초기화 할수록 비용이 증가한다. ・특정한 스테이지의 정예 난이도를 원래 허용하는 횟수보다 여러 번 플레이할 수 있다.
~~일일 던전의 추가 공략 횟수 구매~~	~~・일일 기본 제공되는 횟수를 모두 사용한 이후부터 비용을 지불하기 시작한다.~~ ~~・일일 기본 제공되는 횟수를 초과하는 횟수가 1 회씩 증가할 때마다 비용이 증가한다.~~ ~~・일일 던전을 더 많이 플레이할 수 있으므로, 이를 통해 획득하는 재료와 보석이 더 많아진다.~~
주간 던전의 추가 공략 횟수 구매	・일일 기본 제공되는 횟수를 모두 사용한 이후부터 비용을 지불하기 시작한다. ・일일 기본 제공되는 횟수를 초과하는 횟수가 1 회씩 증가할 때마다 비용이 증가한다.

	・주간 던전을 더 많이 플레이할 수 있으므로, 이를 통해 획득하는 회복제와 게임 플레이 화폐(골드 교환 아이템)이 더 많아진다.
혼돈 던전의 추가 공략 횟수 구매	・일일 기본 제공되는 횟수를 모두 사용한 이후부터 비용을 지불하기 시작한다. ・일일 기본 제공되는 횟수를 초과하는 횟수가 1 회씩 증가할 때마다 비용이 증가한다. ・혼돈 던전을 더 많이 플레이할 수 있으므로, 이를 통해 획득하는 고급 보상도 더 많이 획득할 수 있다.
초월 모드의 추가 공략 횟수 구매	・일일 기본 제공되는 횟수를 모두 사용한 이후부터 비용을 지불하기 시작한다. ・일일 기본 제공되는 횟수를 초과하는 횟수가 1 회씩 증가할 때마다 비용이 증가한다. ・초월 모드를 더 많이 플레이할 수 있으므로, 이를 통해 획득하는 고급 보상도 더 많이 획득할 수 있다.
결투장의 재사용 대기 시간 초기화	・재사용 대기 시간을 초기화할 수 있는 횟수가 제한되어 있다. ・ VIP 레벨에 따라, 재사용 대기 시간을 초기화할 수 있는 횟수가 증가할 수 있다. ・재사용 대기 시간을 초기화할 때마다, 다음 번 재사용 대기 시간을 초기화하는 비용이 증가한다. ・결투장을 이용한 이후, 재사용 대기 시간 없이 곧바로 다음 상대와의 결투를 진행하고자 할 때 사용한다. ・더 많은 결투장 점수와 결투장 코인을 획득할 수 있다.
길드 던전의 추가 공략 횟수 구매	・일일 기본 제공되는 횟수를 모두 사용한 이후부터 비용을 지불하기 시작한다. ・일일 기본 제공되는 횟수를 초과하는 횟수가 1 회씩 증가할 때마다 비용이 증가한다. ・길드 던전을 더 많이 플레이할 수 있으므로, 이를 통해 획득하는 고급 보상도 더 많이 획득할 수 있다.
길드 전쟁의 재사용 대기 시간 초기화	・재사용 대기 시간을 초기화할 수 있는 횟수가 제한되어 있다. ・ VIP 레벨에 따라, 재사용 대기 시간을 초기화할 수 있는 횟수가 증가할 수 있다. ・재사용 대기 시간을 초기화할 때마다, 다음 번 재사용 대기 시간을 초기화하는 비용이 증가한다. ・길드 전쟁을 이용한 이후, 재사용 대기 시간 없이 곧바로 다음 상대 길드와의 전쟁을 진행하고자 할 때 사용한다. ・더 많은 길드 점수와 길드 코인을 획득할 수 있다.

- **D-2-7-3.** 과금 화폐를 소모함으로써 얻는 주된 이득은 **게임 플레이에 투자하는 시간적인 노력의 감소**이다.

: 더 많은 플레이를 할 수 있는 기회, 더 많고 좋은 보상을 얻을 수 있는 기회를 주는 게임 요소들은 과금 화폐를 소모하도록 설계한다.

- **D-2-7-4.** 운영 이벤트로써 결제 상점의 상품에 구매 혜택을 주고자 할 때는, 가격을 낮추는 방식으로 혜택을 줄 수 없고, 원래 판매하는 충전 횟수에 추가적으로 더 충전을 해주는 방식으로 해야 한다.
: 즉, **구매 보너스를 줄 수는 있어도, 판매 가격을 할인해줄 수는 없다.**

※ 즉, 큐빅(과금 화폐) 1,000개를 구매하는데 한화(KWR)로 15,000원이 든다고 가정하자.
이 때, 사용자들의 결제를 장려하기 위해 혜택을 주는 이벤트를 한다고 하더라도, 판매 가격을 10% 할인해서 13,500에 판매하는 등의 방식을 사용할 수 없다.
그 대신, 같은 15,000원을 지불했을 때, 큐빅을 1,000개 대신 10%를 더 추가한 1,100개를 지급하는 식으로 이벤트 혜택을 설계해야 한다.

※ 이와 같은 제한을 두는 이유는, 사용자 계정이 보유하는 VIP 레벨은 결제 금액 그 자체와 연동되는 방식이기 때문이다.
만약, 결제하는 현실 화폐의 금액 자체가 변동이 될 수 있다면, VIP 레벨을 향상하는 시점은 결제액이 할인 되는 시점에 몹시 유리해진다. 또는, 같은 종류의 상품을 구매했음에도 할인율에 따라 VIP 레벨의 달성 정도가 달라져야 한다.
둘 다 사용자들에게 혼란과 불만을 가져다 줄 수 있는 요소들이다.
핵심은 결제 유도를 위한 이벤트 혜택의 종류와 내용에 상관 없이, VIP 레벨은 결제 금액에 정확히 비례해서, **기존 결제 사용자들과 신규 결제 사용자들이 같은 조건으로 획득**할 수 있어야 한다.
그렇기 때문에 결제 금액의 액수가 변동되는 혜택 이벤트의 사용을 금지하고, 다른 방향으로 구매를 장려하는 이벤트를 설계하도록 강제하는 것이다.

◆ D-2-8. 게임 플레이 화폐의 순환 흐름

- **D-2-8-1.** 이 재화의 획득 방식은 다음과 같은 의도로 이루어진다.

종류	설명
일상적인 획득	• 가장 많은 경우에 획득할 수 있는 재화이다. • 플레이어 캐릭터가 자신이 보유한 아이템을 판매할 때, 그 대금은 게임 플레이 화폐만으로 획득할 수 있다. • 획득할 수 있는 양은 플레이한 스테이지의 진행 상황이나 난이도에 따라 다양하다. • 용병 활동과 같은, 실제 시간을 소모하는 방식으로 지속적으로 획득할 수 있다.

	・게임 플레이만으로 많은 양을 획득하기 위해서는, 꾸준히 게임을 플레이하는 시간을 소요하도록 되어 있다.
과금 화폐를 통한 구입	・사용자가 결제한 비용을, 가장 자주 사용하는 게임 플레이 화폐로 간접적으로 교환하는 방식이다. ・별다른 노력을 하지 않고, 대량의 게임 플레이 화폐를 획득할 수 있다.

- **D-2-8-2.** 이 재화를 비용으로 소모하는 컨텐츠와 그 세부적인 특성들은 다음과 같다.

종류	설명
장비 아이템 강화	・각 장비 아이템들의 강화 점수를 올리는 데 드는 비용 ・장비 아이템들의 사용 가능한 레벨과 등급에 비례해서, 강화 레벨을 올리는데 필요한 강화 점수가 더 많이 필요하다. ・재료가 될 다른 아이템들이 필요하다. : 게임 플레이 화폐만 가지고 강화 점수를 올릴 수는 없다. ・가장 흔하게 게임 플레이 화폐를 소모하는 과정이다.
장비 아이템 초월	・장비 아이템의 강화 레벨이 최대에 도달했을 때, 아이템 초월석과 함께 게임 플레이 화폐를 지불해야 강화 레벨의 한계를 높일 수 있다. ・강화 레벨의 한계를 높일수록 더 많은 비용을 지불해야 한다. ・장비 아이템을 더 강화할 수 있는 기회를 부여함으로써, 플레이어 캐릭터가 더 강해지는 기회를 제공한다.
스킬 숙련도 강화	・스킬의 숙련 점수를 올리기 위해서는, 각 스킬마다 게임 플레이 화폐를 지불해서 숙련 점수를 올려야 한다. ・스킬의 숙련 레벨이 오를 때마다, 다음 숙련 레벨에 도달하기 위해 지불해야 하는 비용이 늘어난다. ・스킬 룬을 스킬 숙련도를 강화하는 재료로 같이 쓰는 경우에도, 게임 플레이 화폐를 지불해야 하는 점은 변하지 않는다. : 대신, 스킬 숙련 점수는 게임 플레이 화폐만 지불할 때보다 더 많이 오른다.
숨겨진 상점의 아이템 구매	・숨겨진 상점의 판매 물품 중에는 게임 플레이 화폐로 구매할 수 있는 물품도 있다. ・예상치 못한 재화나 아이템을 획득할 수 있는 기회가 된다. : 보통은 어디 가서 구매하기 어려운 아이템들을 팔기 때문이다.
장비 아이템의 제작	・장비 아이템의 등급을 올리는 제작에는 제작을 위한 재료도 필요하지만, 제작 그 자체에도 게임 플레이 화폐로 비용을 지불해야 한다. ・제작하는 장비 아이템의 등급이 높을수록 제작 비용이 비싸진다. ・장비 아이템의 등급을 높이기 위해서는, 그에 상응하는 대가를 지불해야 하는데, 가장 기초가 되는 비용이 게임 플레이 화폐다.

해당 스테이지에 적용할 부여효과 구매	・스테이지를 플레이하기 전에, 플레이어 캐릭터에게 각종 이로운 효과들을 주는 부여효과들을 구매할 수 있다. ・구매한 부여효과들은 스테이지를 1 회 플레이할 때만 유효하며, 스테이지를 빠져나오면 모두 초기화된다. : 다음 번 플레이에는 재구매해서 들어가야 한다. ・스테이지 공략에 어려움을 느끼는 플레이어나, 더 빠르게 공략하기 위한 플레이어들이 비용을 내고 선택할 수 있는 편의 기능이다.
회복제 구매	・체력 회복, 마력 회복 등을 할 수 있는 소모성 아이템을 구매한다. ・구매 비용은 캐릭터의 레벨에 비례해서 높아진다. : 회복제에는 종류가 별도로 없고, 체력 / 마력을 일관된 '비율'로 회복시키기 때문이다. ・스테이지 공략에 어려움을 느끼는 플레이어는, 회복제를 구매해서 전투 지속력을 늘릴 수 있다.
다른 플레이어의 용병 활동 습격	・다른 플레이어의 용병 활동을 습격할 때는, 습격 활동 자체에 대해 비용을 지불해야 한다. ・습격 횟수가 늘어날 때마다 기본적으로 지불해야 하는 비용이 비싸진다. ・습격 활동은 플레이어 캐릭터가 골드를 획득할 수 있는 수단 중 하나이지만, 지나치게 많이 할 경우, 기본적으로 소비하는 비용이 약탈한 이익보다 크지 않게 될 수 있다.
아이템 강화 보석의 합성	・하위 등급의 아이템 강화 보석을 합쳐서 상위 보석을 만들 수 있는데, 이 때 게임 플레이 화폐로 비용을 지불한다. ・합성할 보석의 등급이 높을수록 더 많은 비용을 지불해야 한다. ・게임 플레이 화폐를 자주, 혹은 대량으로 소모하는 컨텐츠이다. : 물론, 대신 플레이어는 장비 아이템을 더욱 강화시킬 수 있다.
장비 아이템에 장착된 보석을 해제	・장비 아이템의 빈 보석 홈에 보석을 장착하는 데는 비용이 들지 않지만, 장착한 보석을 제거하는 데는 비용이 든다. ・장착한 장비 아이템의 등급이 높을수록, 장비 아이템의 보물 레벨이 높을수록, 장착되어 있는 보석의 등급이 높을수록 제거하는 데 드는 비용이 상승한다. ・보석의 교체는 게임 플레이 화폐를 소모하는 컨텐츠의 하나이다.
스킬에 장착된 스킬 룬을 해제	・스킬의 룬 슬롯에 스킬 룬을 장착하는 데는 비용이 들지 않지만, 장착한 스킬 룬을 제거하는 데는 비용이 든다. ・장착한 스킬의 레벨이 높을수록, 장착되어 있는 스킬 룬의 등급이 높을수록 제거하는 데 드는 비용이 상승한다. ・스킬 룬의 교체는 게임 플레이 화폐를 소모하는 컨텐츠의 하나이다.

- **D-2-8-3.** 게임 플레이 도중에 가장 빈번하게, 가장 많이 획득할 수 있는 보상 단위이다.

- **D-2-8-4.** 가장 여러 곳에서, 가장 자주 소비하는 재화 단위이다.
: 거의 모든 컨텐츠 요소들은 게임 플레이 화폐를 적든, 많든 소모하면서 이용하도록 되어 있다.

※ 게임 플레이의 단기적인 주요 목적 중 하나는 바로 게임 플레이 화폐를 충분히 획득해서, 플레이어 캐릭터의 능력을 더 강하게 만드는 것이다.

◆ D-2-9. 입장 자원의 순환 흐름

- **D-2-9-1.** 이 재화의 획득 방식은 다음과 같은 의도로 이루어진다.

종류	설명
일상적인 획득	• 일정한 시간이 지날 때마다 1 씩 수량을 회복한다. • 자연 회복으로 얻을 수 있는 수량의 최대치가 정해져 있다. • 사용 가능한 입장 자원의 수량이 자연 회복으로 획득 가능한 최대치에 도달하거나 초과했다면, 그 최대치보다 사용 가능한 수량이 더 적어지기 전에는 더 이상 자연적으로 회복되지 않는다.
과금 화폐를 통한 구입	• 사용자의 결제 내역을 곧바로 게임 플레이 기회를 늘려주는 형태로 바꾸는 기능을 한다.

- **D-2-9-2.** 이 재화를 비용으로 소모하는 컨텐츠와 그 세부적인 특성들은 다음과 같다.

종류	설명
스토리 모드(일반)의 입장료	• 1 회 진입할 때마다 입장료로 지불해야 한다. • 스테이지 돌파에 실패하면, 소모한 입장료 중 일부를 반환받는다.
스토리 모드(정예)의 입장료	• 1 회 진입할 때마다 입장료로 지불해야 한다. • 스테이지 돌파에 실패하면, 소모한 입장료 중 일부를 반환받는다. • 일반 난이도일 때보다 입장료가 2 배 비싸다.
~~일일 던전의 입장료~~	~~• 1 회 진입할 때마다 입장료로 지불해야 한다.~~ ~~• 스테이지 돌파에 실패하면, 소모한 입장료 중 일부를 반환받는다.~~
주간 던전의 입장료	• 1 회 진입할 때마다 입장료로 지불해야 한다. • 스테이지 돌파에 실패하면, 소모한 입장료 중 일부를 반환받는다.
혼돈 던전의 입장료	• 1 회 진입할 때마다 입장료로 지불해야 한다. • 스테이지 돌파에 실패하면, 소모한 입장료 중 일부를 반환받는다.
초월 모드의 입장료	• 1 회 진입할 때마다 입장료로 지불해야 한다. • 소모한 입장료는 돌려받지 않는다. : 초월 모드는 미션 실패가 따로 없고, 최대한 도달한 지점만 존재하기 때문이다.

- **D-2-9-3.** ・정기적 / 비정기적으로 입장 자원이 주어지기는 하지만, 플레이 진행도와 캐릭터의 강함과는 관계가 없기 때문에, 무작정 캐릭터만 강하게 키운다고 해서 입장 자원을 많이 얻을 수 있는 건 아니다.

- **D-2-9-4.** 입장 자원 방식(이나 피로도 시스템)은 기본적으로, **게임 컨텐츠의 소모 속도를 컨트롤**하기 위해 존재한다.
: 그렇기 때문에 사용자의 현재 플레이 상태나 결과에 크게 상관 없이, 항상 일정한 양을 제공하는 방식의 정책을 쓴다.

◆ D-2-10. 소탕권의 순환 흐름

- **D-2-10-1.** 이 재화의 획득 방식은 다음과 같은 의도로 이루어진다.

종류	설명
일상적인 획득	・스테이지를 처음으로 최고 등급으로 완료했을 때, 그 때 한 번 보상으로 주어질 수 있다. : 일종의 업적 달성인 셈이다. ・정기적인 접속에 대한 보상으로 주어질 수 있다. ・서비스 운영에 의해 비정기적으로 소탕권이 지급될 수 있다.

- **D-2-10-2.** 이 재화를 비용으로 소모하는 컨텐츠와 그 세부적인 특성들은 다음과 같다.

종류	설명
스테이지 소탕	・스테이지를 플레이했을 때 주어질 수 있는 경험치는 획득할 수 없다. ・시나리오 모드의 던전에서만 사용이 가능하다.

- **D-2-10-3.** 경험치를 획득하지 못하는 대신, 게임 플레이 시간을 들이지 않고, 해당 스테이지의 보상만 빠르게 획득할 수 있는 수단이다.

◆ D-2-11. 아이템의 순환 흐름

- **D-2-11-1.** 이 재화의 획득 방식은 다음과 같은 의도로 이루어진다.

종류	설명
일상적인 획득	・모든 던전 스테이지에서 플레이 결과에 대한 보상으로 주어진다.

	・기본적으로 주어지는 보상 외에, 무작위로 고르는 이벤를 통해 추가적인 보상으로도 받을 수 있다. ・ ・무작위 뽑기에서 획득할 수 있다.
과금 화폐를 통한 구입	・아이템 뽑기에 대한 비용을 지불하고, 무작위한 능력의 아이템을 획득할 수 있다. ・무작위로 등장하는 희귀 상점에서 아이템을 구매할 수 있다.
초월 모드 상점을 통한 구입	・초월 모드 상점에서 판매하는 희귀한 등급과 옵션의 아이템을 구매할 수 있다.
결투장 상점을 통한 구입	・결투장 상점에서 판매하는 희귀한 등급과 옵션의 아이템을 구매할 수 있다.
길드 상점을 통한 구입	・길드 상점에서 판매하는 희귀한 등급과 옵션의 아이템을 구매할 수 있다.

- **D-2-11-2.** 이 재화를 비용으로 소모하는 컨텐츠와 그 세부적인 특성들은 다음과 같다.

₩

종류	설명
아이템 강화	・다른 아이템의 능력을 강화하기 위한 강화 점수를 올리는 재료로 사용한다.
아이템 제작	・아이템 등급을 올릴 때, 그 원본으로 사용한다.
상점에 판매	・상점에 판매해서 게임 플레이 화폐로 환전할 수도 있다.

◆ D-2-12. 스킬 룬의 순환 흐름

- **D-2-12-1.** 이 재화의 획득 방식은 다음과 같은 의도로 이루어진다.

종류	설명
일상적인 획득	・스토리 모드에서는 일반 난이도보다는 정예 난이도에서 주로 등장하도록 설계한다. ・스킬 룬의 등급이 높을수록 등장할 확률이 낮다.
과금 화폐를 통한 구입	・스킬 룬 뽑기에 대한 비용을 지불하고, 무작위한 능력과 타입의 스킬 룬을 획득할 수 있다. ・특정 타입에 대한 ・무작위로 등장하는 희귀 상점에서 스킬 룬을 구매할 수 있다.
초월 모드 상점을 통한 구입	・초월 모드 상점에서 판매하는 희귀한 등급과 옵션의 스킬 룬을 구입할 수 있다. ・이 경우에는 초월 모드 코인을 통해서만 구매할 수 있다.

결투장 상점을 통한 구입	・결투장 상점에서 판매하는 희귀한 등급과 옵션의 스킬 룬을 구입할 수 있다. ・이 경우에는 결투장 코인을 통해서만 구매할 수 있다.
길드 상점을 통한 구입	・길드 상점에서 판매하는 희귀한 등급과 옵션의 스킬 룬을 구입할 수 있다. ・이 경우에는 길드 코인을 통해서만 구매할 수 있다.

- **D-2-12-2.** 이 재화를 비용으로 소모하는 컨텐츠와 그 세부적인 특성들은 다음과 같다.

종류	설명
스킬의 룬 장착	・스킬 룬을 장착한 스킬에, 그 스킬 룬의 고유 능력을 추가적으로 부여한다. ・각 스킬은 합성할 수 있는 스킬 룬의 타입이 있다. 스킬에서 허용하는 타입이 일치해야 스킬 룬과 합성할 수 있다. ・어떤 능력 / 어떤 등급의 스킬 룬을 합성하는지에 따라, 스킬의 특성과 위력이 아예 달라질 수 있다. ・플레이어들에게 후반부까지 주요한 연구 대상이 될 컨텐츠 중 하나이다. : **어떤 스킬에 어떤 스킬 룬을 장착하는지, 그러한 스킬들을 어떻게 조합하는지에 따라 플레이 느낌이 완전히 달라질 수 있다.**
스킬 숙련도 강화	・스킬 룬을 스킬 숙련도를 강화하는 재료로 같이 쓰는 경우에도, 게임 플레이 화폐를 지불해야 하는 점은 변하지 않는다. : 대신, 스킬 숙련 점수는 게임 플레이 화폐만 지불할 때보다 더 많이 오른다. ・상대적으로 흔하게 얻을 수 있거나, 거의 쓸모가 없는 스킬 룬들은 판매하지 않더라도, 스킬 숙련도를 좀 더 빨리 올리는 데 소모할 수 있다.

◆ D-2-13. 아이템 강화석과 초월석의 순환 흐름

- **D-2-13-1.** 이 재화의 획득 방식은 다음과 같은 의도로 이루어진다.

종류	설명
일상적인 획득	・스토리 모드에서는 일반 난이도보다는 정예 난이도에서 주로 등장하도록 설계한다. ・아이템 초월석은 혼돈 던전에서만 등장한다. ・아이템 강화석은 게임 서비스 상의 정기적, 혹은 비정기적인 보상으로 주어질 수 있다.

	・아이템 초월석이 게임 운영 서비스 단계의 직접 보상으로 주어지는 경우는 매우 드물어야 한다.
초월 모드 상점을 통한 구입	・초월 모드 상점에서 아이템 강화석, 혹은 초월석을 판매하는 경우, 이를 구매할 수 있다. ・이 경우에는 초월 모드 코인을 통해서만 구매할 수 있다.
결투장 상점을 통한 구입	・결투장 상점에서 아이템 강화석, 혹은 초월석을 판매하는 경우, 이를 구매할 수 있다. ・이 경우에는 결투장 코인을 통해서만 구매할 수 있다.
길드 상점을 통한 구입	・길드 상점에서 아이템 강화석, 혹은 초월석을 판매하는 경우, 이를 구매할 수 있다. ・이 경우에는 길드 코인을 통해서만 구매할 수 있다.

- **D-2-13-2.** 이 재화를 비용으로 소모하는 컨텐츠와 그 세부적인 특성들은 다음과 같다.

종류	설명
장비 아이템 강화	・장비 아이템은 강화 점수가 존재하고, 강화 점수가 일정량에 도달하면 강화 레벨이 오른다. ・강화 레벨이 오른 장비 아이템은, 그 장비 아이템이 주 능력의 수치가 상승한다. ・ 모든 장비 아이템들은, 각자가 서로의 장비 아이템을 강화해주는 재료가 될 수 있다. 그러나, 아이템 강화석은 장비 아이템을 강화하는 재료의 용도로만 사용할 수 있는 특수한 아이템이다.
장비 아이템 강화 한계 레벨 초월	・아이템 초월석을 통해서만 장비 아이템의 강화 레벨을 더 높일 수 있다. ・아이템 초월석과 관련된 컨텐츠는 캐릭터 육성에 있어 상당히 후반부에 해당하는 컨텐츠이다. : 이미 강화 시스템은 주어져 있고, 초월석은 그 강화 시스템의 부가적인 컨텐츠이다.
상점 판매	・상점에 판매해서 게임 플레이 화폐로 환전할 수도 있다. ・강화 점수를 더 많이 올려줄 수 있거나, 초월 가능한 레벨이 더 높을수록 비싸게 팔 수 있다. ・진짜 골드가 없는데 급하거나, 심심해서 할 짓 없는 플레이어들이 이용할 것 같다.

◆ D-2-14. 스킬 강화석의 순환 흐름

- **D-2-14-1.** 이 재화의 획득 방식은 다음과 같은 의도로 이루어진다.

종류	설명
일상적인 획득	・스토리 모드에서는 일반 난이도보다는 정예 난이도에서 주로 등장하도록 설계한다. ・게임 서비스 상의 정기적, 혹은 비정기적인 보상으로 주어질 수 있다.
초월 모드 상점을 통한 구입	・초월 모드 상점에서 스킬 강화석을 판매할 경우, 이를 구입할 수 있다. ・이 경우에는 초월 모드 코인으로만 구매할 수 있다.
결투장 상점을 통한 구입	・결투장 상점에서 스킬 강화석을 판매할 경우, 이를 구입할 수 있다. ・이 경우에는 결투장 코인으로만 구매할 수 있다.
길드 상점을 통한 구입	・길드 상점에서 스킬 강화석을 판매할 경우, 이를 구입할 수 있다. ・이 경우에는 길드 코인으로만 구매할 수 있다.

- **D-2-14-2.** 이 재화를 비용으로 소모하는 컨텐츠와 그 세부적인 특성들은 다음과 같다.

종류	설명
스킬 숙련도 강화	・스킬의 숙련도는 게임 플레이 화폐를 투자해서 올리지만, 스킬 룬과 스킬 강화석을 재료로 해서 '같은 값에' 숙련도 점수를 더 많이 올릴 수 있다. ・스킬 룬도 스킬의 숙련도 강화를 위해 쓸 수 있지만, 스킬 강화석은 스킬의 숙련도 강화 외의 다른 용도로 사용할 수 없다.
상점 판매	・상점에 판매해서 게임 플레이 화폐로 환전할 수도 있다. ・스킬 숙련 점수를 더 많이 올려줄 수 있거나, 초월 가능한 레벨이 더 높을수록 비싸게 팔 수 있다. ・진짜 골드가 없는데 급하거나, 심심해서 할 짓 없는 플레이어들이 이용할 것 같다.

◆ D-2-15. 재료 아이템의 순환 흐름

- **D-2-15-1.** 이 재화의 획득 방식은 다음과 같은 의도로 이루어진다.

종류	설명
일상적인 획득	・고정적으로 획득하는 경로는 아이템 분해를 통해 획득하는 방식이다. ・주간 던전에서 플레이어가 선택할 수 있는 각 던전 지역마다 다른 재료 아이템을 보상으로 준다.

	• 스토리 모드나 다른 던전에서도 등장할 확률이 있으나, 고정적으로 등장하는 종류의 재화는 아니다.
초월 모드 상점을 통한 구입	• 초월 모드 상점에서 가끔 재료 아이템을 판매할 때가 있다. • 판매하는 재료의 종류와 개수, 가격은 다양하다. • 이 경우에는 초월 모드 코인으로만 구매할 수 있다.
결투장 상점을 통한 구입	• 결투장 상점에서 가끔 재료 아이템을 판매할 때가 있다. • 판매하는 재료의 종류와 개수, 가격은 다양하다. • 이 경우에는 결투장 코인으로만 구매할 수 있다.
길드 상점을 통한 구입	• 길드 상점에서 가끔 재료 아이템을 판매할 때가 있다. • 판매하는 재료의 종류와 개수, 가격은 다양하다. • 이 경우에는 길드 코인으로만 구매할 수 있다.

- **D-2-15-2.** 이 재화를 비용으로 소모하는 컨텐츠와 그 세부적인 특성들은 다음과 같다.

종류	설명
아이템 제작 재료	• 재료가 되는 아이템을 상위 등급으로 향상하는 데 재료로 소모한다. • 재료가 되는 아이템이 높은 등급일수록, 재료의 종류도 다양해야 하고, 소모하는 양도 더 많아진다. • 아이템 제작 과정에 난이도를 주기 위한 요소이다.
상점 판매	• 상점에 판매해서 게임 플레이 화폐로 환전할 수도 있다. • 희귀한 재료일수록 더 비싸게 팔 수 있다. • 진짜 골드가 없는데 급하거나, 심심해서 할 짓 없는 플레이어들이 이용할 것 같다.

- **D-2-15-3.** 아이템 제작과 아이템 뽑기 사이의 선택적 균형을 잡아주기 위한 장치이다.
: 아이템 제작은 등급을 '반드시' 올리므로, 이와 같은 보조적인 난이도를 주는 요소가 꼭 필요하다.

※ 아이템 제작에 재료가 들어가지 않는다고 가정했을 때, 가장 문제가 되는 부분이 아이템 뽑기이다.
왜냐하면, 아이템 뽑기는 어떤 종류의 뽑기를 하더라도, **등장할 아이템의 등급은 무작위**하며, 이를 플레이어가 조절할 수 없기 때문이다.
그러나, 아이템 제작은 뭐가 됐든, 확실하게 **재료가 되는 아이템의 등급이 한 단계 올라감을 보증**한다.
위 내용들을 종합해서 따져보면, 아이템 제작 과정이 그다지 번거롭지도 않고 난이도가 높지 않다면, 이는 완벽하게 아이템 뽑기의 상위 호환 기능에 가까우며, 이는 곧 아이템 뽑기를 할 이유 자체가 사라져 버린다는 말과 같다. 아이템 제작이 과금 화폐를 적극적으로 소모하게 만들지 않는 컨텐츠로 기획되었다는 점에서 더욱 그렇다.

그러므로 아이템의 제작은 하위 등급에서라면 몰라도, 상위 등급에서는 어느 정도까지는 매우 수행하기가 어렵고 까다로울 필요가 있다.

◆ D-2-16. 아이템 강화 보석의 순환 흐름

- **D-2-16-1.** 이 재화의 획득 방식은 다음과 같은 의도로 이루어진다.

종류	설명
일상적인 획득	・고정적으로 획득할 수 있는 컨텐츠는 주간 던전이다. ・게임 서비스 운영 상의 정기적 / 비정기적인 보상의 한 종류로 주어질 수 있다.
초월 모드 상점을 통한 구입	・초월 모드 상점에서 가끔 아이템 강화 보석을 판매할 때가 있다. ・판매하는 보석의 종류와 등급은 다양하다. ・이 경우에는 초월 모드 코인을 통해서만 구매할 수 있다.
결투장 상점을 통한 구입	・결투장 상점에서 가끔 아이템 강화 보석을 판매할 때가 있다. ・판매하는 보석의 종류와 등급은 다양하다. ・이 경우에는 결투장 코인을 통해서만 구매할 수 있다.
길드 상점을 통한 구입	・결투장 상점에서 가끔 아이템 강화 보석을 판매할 때가 있다. ・판매하는 보석의 종류와 등급은 다양하다. ・이 경우에는 길드 코인을 통해서만 구매할 수 있다.

- **D-2-16-2.** 이 재화를 비용으로 소모하는 컨텐츠와 그 세부적인 특성들은 다음과 같다.

종류	설명
장비 아이템의 보석 홈에 장착	・모든 보석들은, 각자 끼울 수 있는 장비 타입의 보석 홈에 끼우면, 고유하게 가지고 있는 능력을 장비 아이템에 더해준다. ・하위 등급 보석 그 자체도 더해주는 능력이 낮을 뿐, 아이템의 보석 홈에 끼우면 아이템을 강화시켜 준다.
아이템 제작 재료	・각 보석들은 아이템의 등급을 높이기 위한 제작에 재료로 들어간다. ・전설 등급 이상의 아이템을 제작할 때만 보석을 재료로 사용한다. : 하위 등급의 보석을 재료로 사용해야 하는 경우를 방지하기 위해서다.
상위 보석으로 합성	・하위 등급 보석들을 모아서 상위 등급 보석으로 합성하면, 같은 종류의 능력이 더 강해진다. ・모든 아이템의 보석 홈은 1 개로 한정되어 있으니, 한정된 보석 홈에 강한 보석을 끼울수록 캐릭터가 더 강해진다. ・보석의 합성에는 게임 플레이 화폐가 들어가며, 등급이 높을수록 소모해야 하는 게임 플레이 화폐도 많아진다.

	: 게임 플레이 화폐를 많이 소비하게 하여, 그 대가로 능력을 강화해주는 컨텐츠 중 하나다.
상점 판매	・상점에 판매해서 게임 플레이 화폐로 환전할 수도 있다. ・보석의 등급이 높을수록 비싸게 팔 수 있다. ・스킬 능력을 부여하는 보석이, 일반 보석보다 기본적으로 더 판매 가격이 비싸다. ・진짜 골드가 없는데 급하거나, 심심해서 할 짓 없는 플레이어들이 이용할 것 같다.

- **D-2-16-3.** 일부 등급의 보석은 하위 등급 보석들을 합성하는 방식 이외의 수단으로는 구할 수 없다.

※ 상위 등급 보석을 한 번에 얻을 수 있는 기능은 신중하게 도입해야 한다.
자칫하면 하뒤 등급 보석들의 획득 여부 자체가 쓸모 없어지게 만들기 때문이다.

◆ D-2-17. 형상 변환권(아이템 / 스킬)의 순환 흐름

- **D-2-17-1.** 이 재화의 획득 방식은 다음과 같은 의도로 이루어진다.

종류	설명
일상적인 획득	・형상 변환을 무료로 할 수 있는 권한은 일반적으로 등장하지 않는다. ・게임 서비스 운영 단계에서도, 형상 변환을 할 수 있는 권한을 주는 경우는 없다.
과금 화폐를 통한 구입	・형상 변환은 거의 예외 없이 과금 화폐를 지불해야 수행이 가능한 컨텐츠이다. ・플레이어의 선택권이 넓은 종류일수록, 형상 변환을 수행하는 비용이 비싸진다.

- **D-2-17-2.** 이 재화를 비용으로 소모하는 컨텐츠와 그 세부적인 특성들은 다음과 같다.

종류	설명
아이템 형상 변환	・형상을 변경한 아이템은, 아이콘의 텍스처와 파츠 모델, 그 모델의 텍스처, (존재하는 경우)이펙트가 모두 변경된다.
스킬 형상 변환	・형상을 변경한 스킬은, 스킬을 시전할 때와, 각 스킬의 단계별 이펙트들의 외형이 모두 변경된다.

- **D-2-17-3.** 형상 변환 기능은 캐릭터의 강함에 전혀 관련이 없는, 순전히 외모와 관련된 컨텐

츠이기 때문에, 무료로 제공하는 일이 사실상 없다.

◆ D-2-18. 아이템 마법 부여석의 순환 흐름

- **D-2-18-1.** 이 재화의 획득 방식은 다음과 같은 의도로 이루어진다.

종류	설명
일상적인 획득	• 아이템 마법 부여의 기회를 일반적인 스테이지 보상으로 획득하는 경우는 없다. • 정기적인 접속 보상, 업적 달성, 게임 서비스 운영 상의 정기적 / 비정기적인 장려 보상의 형태로는 주어질 수 있다.
과금 화폐를 통한 구입	• 마법 부여는 기본적으로 과금 화폐를 소비하기 위한 컨텐츠이다. • 사용자의 선택권이 넓은 마법 부여일수록 수행하는 가격이 비싸다.

- **D-2-18-2.** 이 재화를 비용으로 소모하는 컨텐츠와 그 세부적인 특성들은 다음과 같다.

종류	설명
아이템 마법 부여	• 이미 가지고 있는 아이템의 보조 속성들을 다른 속성들로 바꾸는 기회를 제공한다. • 사용자가 원하는 속성들의 목록이 등장할 때까지는 많은 시행을 거쳐야 한다. • 같은 아이템에 마법 부여를 여러 번 할수록, 다음 마법 부여에 필요한 비용이 증가한다. • 무작위로 전체 능력 옵션을 모두 바꾸는 마법 부여와, 플레이어가 능력 옵션 중 1 가지만 바꾸는 마법 부여는 섞어서 쓸 수 없다.

- **D-2-18-3.** 마법 부여는 기본적으로, 플레이어가 원하는 장비 아이템을 획득하는 데 필요한 시간을 단축하는 역할을 한다.

※ 마법 부여의 근본적인 내용 자체는, 스테이지를 플레이해서 아이템을 보상으로 획득하는 과정과 크게 다르지 않다.
그러나, 마법 부여를 하지 않는다면, 일단 획득하는 아이템의 등급, 장비 부위, 능력 옵션의 가능 범위까지 모든 사항들이 변화해서 등장할 수 있으므로, 결과적으로 '사용자가 원하는' 아이템의 능력에 다가가기 위해 걸리는 시간이 아주 길어질 수 밖에 없다.
하지만 이미 보유한 아이템에 행하는 마법 부여는 아이템의 등급, 장비 부위에 대한 부분을 고정한 상태이기 때문에, '일반적인 획득 과정'보다는 사용자가 원하는 방향으로 좁혀져 있는 범위 안에서 변화가 일어나는 셈이다.

- **D-2-18-4.** 마법 부여는 사용자가 원하는 방향으로 플레이어의 캐릭터를 성장하도록 돕는다.
: 플레이어들은 더 강력한 캐릭터를 만들기 위해, 장비 아이템에 원하는 능력 옵션들로 채워질 때까지 마법 부여를 적극적으로 할 것이다.

- **D-2-18-5.** 마법 부여를 통해 아이템의 속성을 교체할 때는, 교체하는 속성에 대한 선택권을 적게 부여한다.

※ 플레이어가 장비 아이템에 마법 부여를 할 때, 각 능력 옵션마다 선택권을 가지게 되면, 아주 효율적으로 필요한 능력 옵션만 골라서 마법 부여를 하는 방식으로 빠른 시간 안에, 자신의 캐릭터에게 가장 효율적인 아이템들을 만들어버리게 된다.
이렇게 되면, 사실 게임의 수익성은 둘째 문제고, 플레이어가 그 캐릭터를 성장하고 다른 아이템을 찾아야 할 이유가 사라져 버린다. 왜냐하면, 이미 가장 효율적인 옵션들로 마법부여를 해 놓은 상태이기 때문이다. 이러면 게임 자체에 대해 흥미가 없어질 수 밖에 없다.

그러니, 각 캐릭터에게 **'가장 이상적인' 능력 옵션으로만 채워진 아이템은 어떤 플레이 방식으로든 간에 등장하기가 극히 어려워야 한다.**
플레이어에게 옵션을 고를 수 있는 기회를 1 번으로 제한하는 것도, 그리고 그런 경우에 더 비싼 비용을 치러야 하는 것도 이런 이유 때문이다. 플레이어가 마법 부여로 인해 능력 옵션들이 등장하는 확률을 누적적으로 이용해서, 자신에게 이상적인 능력 옵션들의 조합으로 '진화'하게 만드는 부작용을 방지하기 위해서다.

◆ D-2-19. 초월 모드 코인의 순환 흐름

- **D-2-19-1.** 이 재화의 획득 방식은 다음과 같은 의도로 이루어진다.

종류	설명
초월 모드 진행	• 초월 모드를 진행하면 초월 모드 코인을 획득할 수 있다. • 초월 모드의 단계을 많이 진행할수록, 더 많은 초월 모드 코인을 획득할 수 있다.

- **D-2-19-2.** 이 재화를 비용으로 소모하는 컨텐츠와 그 세부적인 특성들은 다음과 같다.

종류	설명
초월 모드 상점	• 일반적으로 쉽게 구할 수 없는 재화들을 판매한다. • 어떤 재화들이 나올지는 그때그때 다르다. • 등장하는 판매 상품들의 목록은, 제한된 횟수만큼 다른 상품 목록으로 교체해서 볼 수 있다.

- **D-2-19-3.** 초월 모드에 적극적으로 참여할수록 더 많은 초월 모드 코인을 얻게 되고, 이로써 초월 모드에 참여하는 플레이어 캐릭터들은 더 빠른 시간 안에 더 강력해질 기회를 얻는다.

◆ D-2-20. 결투장 코인의 순환 흐름

- **D-2-20-1.** 이 재화의 획득 방식은 다음과 같은 의도로 이루어진다.

종류	설명
결투장 승리	・결투장에서 승리하면 결투장 코인을 획득할 수 있다. ・순위가 높은 상대를 쓰러뜨릴수록, 더 많은 결투장 코인을 획득할 수 있다.
결투장 패배	・패배하더라도, 결투장 코인을 획득할 수 있다. ・승리할 때보다는 매우 적은 양의 결투장 코인을 획득한다. ・지든, 이기든, 결투장에 적극적으로 참여하면 '어쨌든 돈은 벌 수 있다.'

- **D-2-20-2.** 이 재화를 비용으로 소모하는 컨텐츠와 그 세부적인 특성들은 다음과 같다.

종류	설명
결투장 상점	・일반적으로 쉽게 구할 수 없는 재화들을 판매한다. ・어떤 재화들이 나올지는 그때그때 다르다. ・등장하는 판매 상품들의 목록은, 제한된 횟수만큼 다른 상품 목록으로 교체해서 볼 수 있다.

- **D-2-20-3.** 자신의 결투장 순위가 높을수록, 기본적으로 획득할 수 있는 결투장 코인의 양이 늘어난다.
: 그러니, 적극적으로 결투장 순위를 높이는 플레이어가 더 많은 금전적 기회를 가진다.

- **D-2-20-4.** 결투장에 적극적으로 참여할수록 더 많은 결투장 코인을 얻게 되고, 이로써 결투장에 참여하는 플레이어 캐릭터들은 더 빠른 시간 안에 더 강력해질 기회를 얻는다.

◆ D-2-21. 길드 코인의 순환 흐름

- **D-2-21-1.** 이 재화의 획득 방식은 다음과 같은 의도로 이루어진다.

종류	설명

길드 던전 진행	・길드 내의 다른 캐릭터들과 함께 길드 던전을 진행하면, 길드 코인을 획득한다. ・길드 던전을 많이 진행할수록, 획득하는 더 많은 길드 코인을 얻을 수 있다. ・돌파한 길드 던전에서만 보상으로 길드 코인을 지급한다.
길드 전쟁 승리	・승리하면 길드 코인을 획득할 수 있다. ・순위가 높은 상대를 쓰러뜨릴수록, 더 많은 결투장 코인을 획득할 수 있다.
길드 전쟁 패배	・패배하더라도, 길드 코인을 획득할 수 있다. ・승리할 때보다는 매우 적은 양의 길드 코인을 획득한다. ・지든, 이기든, 길드 전쟁에 적극적으로 참여하면 '어쨌든 돈은 벌 수 있다.'

- **D-2-21-2.** 이 재화를 비용으로 소모하는 컨텐츠와 그 세부적인 특성들은 다음과 같다.

종류	설명
길드 상점	・일반적으로 쉽게 구할 수 없는 재화들을 판매한다. ・어떤 재화들이 나올지는 그때그때 다르다. ・등장하는 판매 상품들의 목록은, 제한된 횟수만큼 다른 상품 목록으로 교체해서 볼 수 있다.

- **D-2-21-3.** 길드 전쟁에서의 순위가 높을수록, 그 길드의 길드원 캐릭터들이 기본적으로 획득할 수 있는 길드 코인의 양이 늘어난다.

- **D-2-21-4.** 길드 활동에 적극적으로 참여할수록 더 많은 길드 코인을 얻게 되고, 이로써 길드원으로 활동하는 각 플레이어 캐릭터들은 더 빠른 시간 안에 더 강력해질 기회를 얻는다.

◆ D-2-22. 무작위 뽑기권의 순환 흐름

- **D-2-22-1.** 이 재화의 획득 방식은 다음과 같은 의도로 이루어진다.

종류	설명
일상적인 획득	・일반적으로 게임 플레이하는 과정에서 정규적인 보상으로 무작위 뽑기를 무료로 하는 권한을 얻을 수는 없다. ・매일 각 플레이어에게 무료로 무작위 뽑기를 할 수 있는 횟수가 주어진다.
과금 화폐를 통한 구입	・개수를 축적할 수 있는 재화의 단위로 판매하지 않는다.

	・그때그때 과금 화폐 단위의 비용을 지불하여 무작위 뽑기를 진행할 수 있다..

- **D-2-22-2.** 이 재화를 비용으로 소모하는 컨텐츠와 그 세부적인 특성들은 다음과 같다.

종류	설명
무작위 뽑기	・무료로 진행하는 경우에는 사용 횟수와 재사용 대기 시간이 존재한다. ・요금을 지불한 무작위 뽑기는 제한 없이 수행할 수 있다.

- **D-2-22-3.** 무작위 뽑기권은 당첨 보상으로 무엇이 등장할지 모르는 복권과 같은 역할을 한다.

D-3. 수익 모델(외적 요소)

◆ D-3-1. 광고 노출

- **D-3-1-1.** 광고는 공지사항처럼 웹 페이지 뷰를 제공하는 형식으로 구현한다.

- **D-3-1-2.** 광고 웹 페이지들은 한 번 본 이후에, 하루 동안 다시 보이지 않게 하는 옵션을 제공한다.

- **D-3-1-3.** 화면의 특정 영역에 지속적으로 광고를 노출하는 방식
  : 사용하지 않는다.

- **D-3-1-4.** 종료 팝업 윈도우의 배경으로 광고를 넣는 방법
  : 필요한 경우에 적용한다.

◆ D-3-2. 제 3자 제품과의 연동

- **D-3-2-1.** 타 게임을 다운로드 받으면 재화를 획득하는 기능
  : 이걸 어떤 용어로 부르는 게 있던데...;;

- **D-3-2-2.** 제 3자 제품과 연동했을 때 획득하는 재화의 단위는 과금 화폐이다.
  : 가장 처음 단계부터 필요로 하면서도, 결제를 필요로 하는 재화 단위이기 때문에, 사용자의 활동을 유인할 수 있는 최적의 수단이다.

D-4. 기종(Platform) 전략

◆ D-4-1. 제한 사항

- **D-4-1-1.** 시장에서 중간 등급 이상의 성능을 가진 기기들을 대상으로 한다.

- **D-4-1-2.** 게임 개발에 사용하는 엔진 라이브러리에서 요구하는 최소 하드웨어와 운영체제 버전에 미치지 못하는 기기는, 이 게임을 플레이 할 수 없다.

◆ D-4-2. Android

- **D-4-2-1.** 모든 사용자는 무료로 클라이언트 응용 프로그램을 다운로드 받아 설치할 수 있다.

- **D-4-2-2.** 응용 프로그램 내에서 특정한 컨텐츠 상품을 현실 화폐를 이용해 결제할 수 있다.

◆ D-4-3. iOS

- **D-4-3-1.** 모든 사용자는 무료로 클라이언트 응용 프로그램을 다운로드 받아 설치할 수 있다.

- **D-4-3-2.** 응용 프로그램 내에서 특정한 컨텐츠 상품을 현실 화폐를 이용해 결제할 수 있다.

◆ D-4-4. 기타

- **D-4-4-1.** 적어도 최초 출시에는 지원 계획이 없다.

D-5. 라이선스

◆ D-5-1. 프로그램 배포 라이선스

- **D-5-1-1.** 대한민국의 저작권 관련 법령이 정한 바를 준수하는 조건에서 배포한다.

- **D-5-1-2.** 스노우 패밀리㈜에서 직접 배포하거나, 혹은 스노우 패밀리㈜에게서 배포 허가를 받은 사업자만 프로그램을 배포할 수 있다.

- **D-5-1-3.** 스노우 패밀리㈜의 허가를 받지 않는 한, 어떠한 경우에도 게임 서비스의 소스 코드와 리소스 데이터들의 원본을 공중에 배포하거나 복제할 수 없다.

◆ D-5-2. 최종 사용자 라이선스

- **D-5-2-1.** 최종 사용자에 대한 라이선스 발행에 대한 조건은 발행사의 정책을 참고한다.
: 예를 들자면, 최종 사용자의 게임 플레이 이용은, **최종 사용자에게 부여된 계정 단위**로 이루어진다.

- **D-5-2-2.** 최종 사용자의 계정할당 방식
: 최종 사용자는, 게임을 실행할 경우, 최초 실행할 경우에 자동으로 계정 번호를 하나 할당 받는다. 이렇게 임시 할당된 계정 번호에 대한 정보는, 일단 게임을 실행한 기기의 지역 저장 장치에 저장된다.
그 후에, 최종 사용자의 선택으로 정식으로 가입하기로 한 경우, 사용자가 등록한 E-Mail당 계정을 하나씩 발급한다.

- **D-5-2-3.** 게임 플레이 컨텐츠 중 일부 아이템, 스킬 등의 게임 자산은 현금, 또는 현금으로 구입해야 하는 별도의 전용 게임 화폐를 통해 구매해야 한다.

- **D-5-2-4.** 현금화 관련 컨텐츠들은 기획 내용 및 과금 정책에 따라 변경될 수 있다.
프로젝트의 응용 프로그램은 이러한 변경 내역들을 올바르게 반영할 수 있어야 한다.

◆ D-5-3. 라이선스 인증 방식

- **D-5-3-1.** 최종 사용자는 게임 클라이언트에 최초 접속할 때, 사용자 등록 과정을 거쳐야 한다.

- **D-5-3-2.** 최종 사용자는 **게임 클라이언트에 접속할 때마다, 등록한 계정 정보를 통해 자기 자신의 라이선스를 인증**한다. 그리고 나서야, 프로그램의 다음 컨텐츠(게임 플레이)를 진행할 수 있다.
(싱글 플레이를 할 경우, 인증 절차를 거칠 것인지 여부를 결정해야 한다.)

- **D-5-3-3.** **최종 사용자의 사용 권한에 대한 인증 과정은 자동적으로 진행하게 할 수 있어야** 한다. 단, 그러한 자동화 과정은 반드시 사용자의 동의를 거쳐야 한다. 또한, 게임 클라이언트에서 인증 자동화 과정을 기본 옵션으로 둘 수 없다.

◆ D-5-4. 라이선스 해지

- **D-5-4-1.** 최종 사용자가 게임 클라이언트 설치가 허가된 계정을 삭제하거나, 혹은 계정 식별을 위해 필요한 발행사의 식별 ID나 코드 등을 해지한다면, **게임에 대한 라이선스 역시 해지한 것으로 간주**한다.

- **D-5-4-2.** 라이선스가 해지된 계정은 게임 데이터를 포함한 모든 데이터가 초기화된다.
: 만일 같은 계정을 다시 만들게 되면, 모든 것을 처음부터 새로 시작해야 한다.

D-6. 서비스 전략

◆ D-6-1. 배포 정책

- **D-6-1-1.** 앱 배포는 **각 플랫폼 별로, 출시 마켓마다 1종의 앱을 배포**한다.
: 플랫폼이 같은 경우에도 마켓이 다른 경우, 그 각각의 마켓에 앱 1종으로 배포한다.
하나의 마켓에 여러 종류의 앱으로 등록하지 않는다.

※ iOS 플랫폼의 경우, 마켓이 Appstore 하나로 통일이 되어 있는데 반하여, Android 플랫폼의 경우에는 Google이 직접 운영하는 GooglePlay를 비롯하여, 각 지역 통신사 별 마켓이 따로 있는 경우가 많다.
이런 경우에는, 같은 Android 플랫폼을 사용한다고 하더라도, 별개의 앱 종류로 구분하게 되므로, 사실상 앱 출시는 플랫폼 종류만 따라가는 게 아니라, 마켓 종류를 따라가는 게 맞다.

- **D-6-1-2.** 앱 내부 컨텐츠는 **전 세계적으로 통일되어 있는 1종의 컨텐츠로 배포**한다.
: 컨텐츠의 종류에 따라 같은 내용의 앱을 여러 종류로 분리하여 배포하지 않는다.

※ 그러니까, 한국어 버전과 영어 버전, 중국어 버전의 게임 앱을 따로 배포하지 않는다.
그냥 게임 앱은 마켓 별로 1종 뿐이며, 그 내부에서 언어를 옵션을 통해서 설정할 수 있을 뿐이다.

©Apple의 경우에는, 같은 컨텐츠에 대한 여러 개의 앱 배포를 애초에 허용하지도 않는다.

- **D-6-1-3.** 게임 내에서 **앱 내부의 기능을 통한 별도의 리소스 배포를 하지 않는다.**
: 최신 리소스 배포를 하기 위한 설비를 따로 구축하기 위해서는 별도의 서버를 유지해야 하기 때문이다.
현재의 비용 상태로는 그러한 서버를 유지할 수 없다.

※ 이 경우, 리소스 데이터가 교체되는 경우에도 앱 자체에 대한 패치가 이루어져야 한다.
뭔가 하나라도 바뀔 때마다 패치가 이루어진다는 점에서 사용자들에게 좀 더 불편할 수도 있지만, 제작하는 입장에서는 별도의 리소스 배포를 위한 서버를 유지하지 않아도 된다는 점에서 비용을 절감할 수 있다.

※ 마케팅 측면에서의 불리함 역시 존재하는데, 그 이유는 아래와 같다.
제작사에서 제공하는 앱 내부 기능을 이용한 리소스 배포가 없는 경우,(앱 내에서 패치 다운로

드가 없는 경우) 모든 게임 코드와 리소스 데이터들을 앱에 탑재해야 하는데, 그러면 필연적으로 앱 용량이 늘어나게 된다.

애플 앱스토어의 경우, 현재 iOS 7 기준으로 100MB의 용량까지 Wi-Fi를 이용하지 않고 다운로드 받을 수 있다. (Download On The Air) 그 이상의 용량인 경우, Wi-Fi를 통해서만 다운로드가 가능하다.
구글의 구글 플레이의 경우, ~~50MB~~100MB가 넘으면 초과하는 데이터에 대해 별도의 데이터 파일(*.obb)로 제작하여 배포하여야 하며, Wi-Fi를 통해 다운로드할 것을 권장하는 경고가 뜬다.

위 두 사례는 어떠한 마켓에 진출하더라도 벌어질 수 있는 현상이며, 앱을 다운로드하는 데 있어 통신 방법의 자유가 허용되는 경우보다 사용자 모집 인원, 앱 다운로드 순위 등에 있어 불리한 요건으로 작용할 수 있다.
특히, 이 부분은 대한민국 국내보다 해외 시장에서 더 민감한 요소일 수 있다.

◆ D-6-2. 마켓 플랫폼 정책

- **D-6-2-1.** 결제에 대한 보안을 구성할 수 없는 마켓은 출시 대상에서 제외한다.
: 구조적으로 클라이언트와 마켓 서버만의 검증만으로 구매를 처리할 수 밖에 없는 곳이라든가 …

- **D-6-2-2.** 운영체제 제조사들의 마켓

- **D-6-2-3.** 통신사 마켓

◆ D-6-3. 결제 정책

- **D-6-3-1.** 과금할 수 있는 기능은, 사용자가 과금할 경우, 해당 과금 아이템의 소유권이 완전히 사용자에게 한 번에 귀속되는 형태로만 설계한다.

※ 쉽게 예를 들자면, 기간제 대여와 같은 방식을 사용할 수 없다는 의미이다.
실제 시간을 통해 일종의 디지털 재산을 '대여'하고 기간이 지나면 '회수'하는 방식을 필수적으로 데이터베이스에 저장된 사용자 정보와 네트워크 연결 기능이 필요하다. 현재의 사업 방향으로는 이런 서비스는 불가능하기 때문에, 무조건 사용자가 뭔가를 과금하고 구매하면 사용자에게 과금 아이템을 완전히 넘겨주는 방식만 써야 한다.

- **D-6-3-2.** 한 번 구매한 과금 아이템은, 어떠한 경우에도 환불할 수 없다.

※ 환불 기능을 해주고 싶어도, 현재로서는 자동화된 환불 시스템을 갖출만한 기술이나 자본이 없다.
더구나 특정한 마켓(통신사 마켓 등)에 진입한 경우에는 각 마켓마다 환불 정책이나 기술들이 다 다르기 때문에 정치적으로도, 기술적으로도 매우 큰 규모의 기능이 될 수 밖에 없다. 심지어, 마켓의 운영 주체의 서비스 지원이 없이는 환불 자체가 불가능한 경우마저도 존재한다.

환불 기능을 지원하는 것은 사용자의 입장에서는 신뢰성을 높여줄 만한 요소이기는 하지만, 기능적인 면에서는 현재로써는 너무도 큰 도전이다.

- **D-6-3-3.** 결제 내용은 사용자의 계정과 연동한다.
: 게임을 플레이 하는 기기가 다르더라도, 동일한 사용자의 계정 정보를 이용한다면, 결제 내역과 그 결과물로 동일하게 다른 기기에서 사용할 수 있다.
반대로, 같은 기기에서 다른 사용자 계정으로 로그인 한다면, 결제 내역 및 결과물의 적용 대상도 바뀌기 때문에, 다른 계정의 결제 내역과 결과물을 사용할 수 없게 된다.

◆ D-6-4. 운영 정책

- **D-6-4-1.** 앱을 등록한 마켓의 앱 평가 기능에 대응한다.
: 유용한 사용자 의견 수집을 위해, 마켓에 등록할 때, 버그 등의 이슈를 보고해 줄 수 있는 대표적인 연락처(버그 보고용 e-mail 계정 주소, 전화번호 등)를 명기한다.

- **D-6-4-2.** 연락처를 남기는 정도는 서비스하는 마켓에서 앱을 등록할 때 요구하는 최소 요건에 따른다.
: 다만, 최소한 버그를 보고할 용도의 e-mail 계정 주소는 있어야 한다.

- **D-6-4-3.** 게임에 대한 블로그를 운영한다.
: 현실적으로 비용이 들지 않으면서 사용자들과 의사소통하고, 버그 보고 및 개선 사항에 대한 건의를 취합할 수 있는 유일한 수단이 아닐까 한다.

- **D-6-4-4.** 사회 관계망 서비스(Social Network Service) 운영
: 현재로써는, 고려하고 있지 않다.

※ twitter, facebook 등의 SNS를 통해, 게임 앱에 대한 게시물을 올리고 광고하는 운영을 말한다.
단순 개설 자체는 어렵지는 않으나, 큰 의미가 있을지는 불명확함. 출시 이후에 이 부분에 대해 다시 결정을 내리는 것이 좋을 것으로 보인다.

◆ D-6-5. 사회 관계망 서비스(Social Network Service)와 통합

- **D-6-5-1.** 현재로써는, 고려하고 있지 않다.

※ 사회 관계망 서비스의 정보를 활용하는 방식은, 그 활용 정도에 따라 많은 이슈를 발생시킬 수 있다.
이를테면, facebook의 연락처를 연동하여 친구를 초대할 수 있게 하거나, 캐릭터 정보를 공유할 수 있게 하는 이슈들은 첫 출시 목표로는 과도하다고 판단한다.

게임 출시 이후, 컨텐츠의 활성화 및 매출 성장 추세를 지켜보면서 SNS와 통합하는 부분을 다시 고려하는 것이 좋다고 판단한다.

◆ D-6-6. 고객 유도

- **D-6-6-1.** 출시 전 사전 마케팅

- **D-6-6-2.**

- **D-6-6-3.**

D-7. 고객관리

◆ D-7-1. 사용자 계정 정책

- **D-7-1-1.** 사용자 계정을 직접 구현하지 않고, 타 서비스의 계정과 연동하는 경우, 계정 정책에 관한 사항들은 계정을 연동하는 서비스의 정책을 따른다.
  : 즉, 이하 서술할 명세 사항들은 **계정 관리 시스템을 자체적으로 보유하면서 직접 구현해야 하는 경우에 한하여 적용**한다.

- **D-7-1-2.** 사용자가 원할 경우, 사용자의 계정을 삭제하는 메뉴를 제공해야 한다.

- **D-7-1-3.** 삭제한 사용자의 계정은 1년 간 별도로 보관한다.
  : 마음이 바뀌어서 복구해달라고 할 수도 있잖아...

- **D-7-1-4.** 사용자가 원하지 않더라도, 지역 법률에 의거해서 계정 정보와 식별 정보를 분리하거나 폐기해야 하는 경우에는, 법률이 정하는 바를 구현해야 한다.

※ 대한민국의 경우, 정보통신망이용촉진 및 정보보호 등에 관한 법률 제 29조(개인정보의 파기)에 의해, 최근 1년 간 이용 기록이 없는 휴면 계정과 개인 식별 정보에 대해 분리 및 삭제하도록 규정하고 있다.
다른 서비스 지역에서는 세부적인 내용이 다를 수 있으므로, 지역 법률에 맞게 구현한다.

◆ D-7-2. 사용자 계정 정보의 복구

- **D-7-2-1.**

◆ D-7-3. 푸시 알림 정책

- **D-7-3-1.**

D-8. 현지화(Localization)

◆ D-8-1. 출시 시점의 지원 언어

- **D-8-1-1.** 한국어
: 한국의 앱스토어 시장 규모가 작지만(한국에서의 Android 플랫폼의 점유율은 90% 이상이다.), 어쨌든, 1차적으로는 한국어로 출시한다.
(시장 규모가 작아도 일단 팔 수 있는 가능성은 있으니까...)

- **D-8-1-2.** 영어
: **사실상의 주력으로 노려야 할 시장은 영어권 국가들**이다. 상대적으로 한국보다 iOS의 점유율이 상당히 높은 편이고(하지만 세계적으로도 최근에는 Android 플랫폼에 서서히 밀려가는 추세이긴 하다,), 구매력도 훨씬 높은 경우가 많기 때문이다.

- **D-8-1-3.** iOS 플랫폼의 경우에는 같은 내용을 가진 앱을 언어군 별로 분리해서 앱스토어에 올리는 행위를 허가하지 않는다.
: 따라서, **앱 내부에서 언어군 별로 전환 처리하는 기능이 반드시 구현되어야 한다.**

<1 종의 앱에서 여러 언어로 전환하는 기능(by Galaxy On Fire 2)>

◆ D-8-2. 출시 이후 지원할 언어

- **D-8-2-1.** 중국어
: 세계 최대의 시장. (이것 밖에 더 말이 필요한감?)

※ 하지만, **카피캣이 워낙 난무하기 때문**에, 지나친 기대는 금물이다. (복잡하지 않은 구조의 게임인 경우, 거의 2 ~ 3 달 이내에 복제 게임 등장이 가능한 대륙의 기상...)

- **D-8-2-2.** 일본어
: 일본 사용자들의 평균 구매력은 대한민국 사용자들의 그것보다 훨씬 높다고 알려져 있다.

※ 일본 사용자들이 즐기는 게임에 대한 성향 자체가 '세계 주류'와 동떨어진 경우가 많기 때문에, 그 부분을 잘 연구해봐야 한다.

- **D-8-2-3.** 그 외 언어들은 시장 상황에 따라 판단하여 추가적으로 지원하도록 한다.

◆ D-8-3. 지역별 마케팅

- **D-8-3-1.** 특정한 서비스 지역에만 등장하는 컨텐츠
: 마케팅 전략에 따라, 특정한 문화권에서 더 선호하는 컨텐츠를 제작할 수 있다.

※ 예를 들면, 일본에서는 좀 더 귀엽고 어리게 보이는 캐릭터들이 인기를 끄는 경향이 있고, 서구권에서는 그들 나름대로 인기를 끄는 캐릭터들이 존재한다. (주로 ~맨 식으로 등장하는... 혹은 스포츠 스타라든가...)
이런 경우, 각 지역의 문화에 맞는 컨텐츠를 별도로 개발해서 적용하는데 따른 개발 비용과 아키텍처를 주의 깊게 고려해야 한다.

마케팅 이벤트의 또 다른 대표적인 경우는 명절 이벤트를 들 수 있다. 국가나 지역마다 명절의 기간, 행사 방식, 문화적인 느낌이 모두 제각각이기 때문에, 이러한 이벤트를 지역화에 맞춰서 진행하는 것은 좀 더 큰 비용을 유발한다.

- **D-8-3-2.** 출시 이후의 마케팅 계획
: 현실적으로 각 지역별로 특화하는 방식의 마케팅을 당장 기획하기는 어려울 것으로 판단한다.
지금 시점에서 판단하기보다는, 차후 게임의 수익 기대치 및 결과에 대해 어느 정도 신뢰할 수 있는 데이터가 쌓이면 그 때 새로 계획을 잡기로 한다.

D-9. 사용자 데이터 수집 정책

◆ D-9-1. 제한사항

- D-9-1-1.

◆ D-9-2. 수집 정보의 종류

- D-9-2-1.

◆ D-9-3. 수집 / 가공 방식

- D-9-3-1.

◆ D-9-4.

- D-9-4-1.

D-10. 기능수정 및 확장

◆ D-10-1. 패치

- D-10-1-1.

◆ D-10-2. 확장팩

- D-10-2-1.

◆ D-10-3. 후속작

- D-10-3-1.

◆ D-10-4. 사용자 모드

- D-10-4-1. 그런 거 없다.

D-11. 고객지원 정보

◆ D-11-1. 도움말

- **D-11-1-1.** 아이템의 종류나 등급이 많고, 합성하는 기능까지 있기 때문에, 아무래도 도움말이 필요한 사용자들이 있을 거라고 판단한다.

- **D-11-1-2.** 물론, 굳이 도움말을 제공하지 않는다고 하더라도, '알아서 할 사람들은 알아서 할 것이다.' 그렇지만, 뭐가 뭔지 잘 모르는 플레이어들을 위한 배려도 해줘야 하지 않겠는가?

- **D-11-1-3.** 기술적인 부분에서의 문제도 있는데, 모든 정보를 게임 내 GUI 형식으로 표시하려고 하다 보면, 대량의 실시간 폰트 렌더링의 사용이나 복잡한 단계의 GUI 그리기 등으로 상당한 성능상의 불이익을 감소해야 한다.

- **D-11-1-4.** 또한, 성능상 불이익 만으로 끝나면 좋지만, 대개는 플레이어들에게 지저분하고 알아보기 힘든, PC에서나 통할 법한 UX로 인식될 여지마저 있다.

※ 그렇지 않아도 이미 모바일 RPG 중에는 PC 게임 화면을 축소시킨 것에 가까운 GUI 형태를 지닌 게임들이 심심치 않게 돌아다닌다. 눈알이 빠질 정도로 조그만 글씨와 버튼을 가지고 씨름하게 만드는 게임을 몇 개 해 본 경험이 있을 것이다.
이 게임은 넓은 화면을 가진 타블렛 PC 전용 게임이 아니기 때문에 특히 이런 부분을 주의해야 한다. (사실 작은 타블렛 PC도 만만치 않게 수량이 많으므로, 타블렛 PC 대상 게임이라고 해서 안심할 수는 없다.)

- **D-11-1-5.** 도움말을 내장된 텍스처 파일을 보여주는 방식으로 할지, 아니면 외부에 웹 페이지를 링크하는 방식으로 할지 결정해야 한다.

※ 내장된 텍스처 파일은 오프라인 상태에서도 도움말을 보여줄 수 있다는 장점은 있지만, 내용을 수정할 때마다 실행파일 배포까지 새로 해야 하는 단점이 있다. 또한 텍스처 크기 자체도 상당히 클 것이므로, 앱 배포 용량이나 실행 메모리 용량에 신경이 쓰이지 않을 수 없다

.

웹 페이지 링크 방식은 기기가 인터넷에 연결되어 있어야만 접근할 수 있다는 단점이 가장 크다. (이 게임은 네트워크 연결이 필수가 아니다!) 또한, 어떻게 도움말 웹 페이지를 계속 유지할 것인지도 문제가 된다. (아무튼 고정 URL을 사용하려고 하면 돈이 필요하다.)
그 외에는 다 장점이다. 언어 별로 별도의 페이지를 제공해줄 수도 있고, 주소 링크가 바뀌지

않는 한, 도움말 내용을 바꾸더라도 실행파일을 새로 배포할 필요도 없다. 내용이 차지할 용량도 크게 신경 쓸 필요가 없다.

◆ D-11-2. 제작자 항목(Credit)

- **D-11-2-1.** 만들 수도 있고, 안 만들 수도 있고...

- **D-11-2-2.** 만약 만들게 된다면, 텍스트를 영어 기준으로 하는 것이 어떨까 싶다.
  : 제작자 소개까지 언어 별로 따로 만들기는 좀 귀찮지 않은가?

※ 좀 더 타협하는 방식으로는 한국어, 영어 버전 두 가지를 만들고, 한국 외에서 실행하는 경우에는 영어로 나오도록 만드는 방법이다.
구체적인 방법은 실행하는 장치의 언어 설정을 가져올 수 없거나, 그런 방법을 모를 경우에는 게임에서의 현재 언어 설정을 참고해서 쓰면 될 것이다.

◆ D-11-3. 연락처

- **D-11-3-1.** 제작자 상호와 버그 보고가 가능한 연락처를 제공해야 할 수 있다.

- **D-11-3-2.** 필요에 따라 이 게임과 관련된 공식 카페, Facebook 페이지, Twitter, 공식 커뮤니티 사이트 등에 대한 링크를 제공해야 할 수 있다.

D-12. 게임 서비스 관리 도구

◆ D-12-1. 관리 도구의 분류

- D-12-1-1.

◆ D-12-2. 운용 정보

- D-12-2-1.

◆ D-12-3. 구조와 제한 사항

- D-12-3-1. 관리 도구들은 웹 페이지 방식으로 제작한다.

◆ D-12-4.

- D-12-4-1.

E. 프로젝트 관리

E-1. 프로젝트 버전 관리

◆ E-1-1. 버전 관리 시스템(Version Control System)의 관리

- **E-1-1-1.** 프로젝트의 버전 관리 시스템은 중앙 저장소 방식으로, 서버 – 클라이언트 모델을 사용하는 **Subversion(SVN)**을 이용한다.

※ 하고 많은 VCS 중에서 굳이 Subversion(SVN)을 쓰는 이유는 아래와 같다.

사실 요즘의 추세는, Git 으로 대표되는 **분산형 버전 관리 시스템(Distributed Version Control System, DVCS)**으로 프로젝트를 관리하는 방식을 많이들 선호한다.

DVCS 에는 여러 가지 장점이 많지만, (그러니까 많이들 쓰겠지?) 그럼에도 불구하고, 이 프로젝트에서 적용하기에는 몇 가지 적합하지 않은 부분이 있다고 판단했기 때문에 사용하지 않기로 결정했다.

1. DVCS 개념의 복잡도로 인한 사용법 교육의 어려움

DVCS 는 말 그대로 저장소를 분산해서 관리하는 개념의 방식이라서, 각자의 컴퓨터마다 저장소가 별도로 존재한다. 그렇기 때문에 네트워크에 연결되어 있지 않은 상태에서도 Commit 이 가능하다는 훌륭한 장점이 있다.

그렇지만, 로컬 저장소와 원격 중앙 저장소가 분리되어 있기 때문에, 결국은 작업을 합치기 위한 '로컬 저장소끼리의 Commit' 과정이 또 있어야 한다. (이걸 Push 라고 한다.)

즉, **Commit 과정이 이중으로 이루어져야** 하며, 그 과정에서 로컬 저장소와 원격 저장소 간 병합(Merge)에 대한 이해도 필요하다. 이런 일련의 과정들은 SVN 보다 직관적이지 않으며, 개념적으로도 훨씬 복잡하고, 처음 배우는 사람이 얼른 배우기도 쉽지가 않다.

게임 프로젝트는 다른 일반 애플리케이션 프로젝트와는 다르게, 프로그래머가 아닌 개발 인력의 비중이 높다. 그러므로, 이렇게 사용 개념이 복잡한 VCS 는 **사용법에 대한 교육 문제**가 불거질 수 밖에 없다.

2. 스냅샷 단위의 저장소 방식으로 인한 저장소의 거대 용량화

DVCS 에서의 저장소는 SVN 과는 다르게 스냅샷 단위로 리비전을 관리한다. 그렇기 때문에 파일의 차이점만 압축한 방식으로 저장하지 않고, 원본 데이터와 디렉토리 구조가 그대로 저장소 파일로 이용된다.

이러한 방식은 체크아웃, 커밋, 업데이트 속도에 강점이 있고, 분기된 저장소끼리 병합할 때, 그 이력을 추적할 때도 장점이 있다. (실제로, DVCS 들은 SVN 과는 달리, 분기된 저장소끼리 서로 디렉토리 구조나 파일의 이름, 삭제 유무가 달라져도 정확하게 추적을 해준다.)

하지만 저장소에 대해 분기(Branch)나 태깅(Tagging)을 할 때마다, 원본 데이터들만큼의 저장소를 추가로 생성하는 단점이 있다. 이는 SVN 에서는 분기를 해도 단순히 참조 지점만 추가로 생성하는 것과는 대조적이다.

DVCS 의 이러한 저장소 방식은 저장소 자체의 크기가 커질 일이 별로 없는 일반적인 소프트웨어 프로젝트에서는 별 문제가 되지 않는다. 그러나, 게임 프로젝트에는 엄청난 개수의 미디어 파일들이 필요하며, 이러한 파일들은 소스 코드가 적힌 텍스트 파일과는 비교가 안될 정도로 용량이 크다.

결국, 프로젝트가 진행되어 가면서 DVCS 방식의 저장소에서 분기를 할 때마다, 전체 프로젝트의 저장소가 저장 매체에서 차지하는 용량은 급격하게 불어날 수 밖에 없다.

3. 적절한 Push 와 Merge 전략이 필요함

커밋 과정이 이중으로 이루어지는 점(로컬 저장소 커밋과 원격 중앙 저장소에 대한 푸시) 때문에 실제 사용자들이 몹시 혼동할 수도 있고, 원격 중앙 저장소에 대한 푸시에 소홀해질 가능성이 있다.

DVCS 의 개념에 익숙하지 않은 사용자들은 계속해서 로컬 저장소만 가지고 놀 가능성이 높아지고, 나중에 지적을 받든가 해서 한 번에 원격 중앙 저장소에 푸시 하려다가 프로젝트 충돌에 직면하기가 쉽다.

특히, 로컬 저장소에서만 장기간 작업을 하다가 컴퓨터가 맛이 가버리는 사태가 벌어져서 로컬 저장소를 복구하지 못한다면... 그냥 그 사람의 작업 내역들은 날아가버리는 셈이다.

DVCS 를 도입하려면 원격 중앙 저장소에 Push 를 했는지 확인하기 위한 명확한 규정이 있어야 한다.

4. 저장소의 일부 데이터만 체크아웃하기 어려움

DVCS 는 SVN 과 달리, 저장소의 일부 디렉토리만 체크아웃 한다는 개념이 없다. 무조건 저장소를 통째로 복제해야 한다. (그래서 명령도 체크아웃이 아니라 저장소를 복제 Clone 한다고 되어 있다.)

프로젝트의 용량이라고 해봐야 소스 코드가 대부분인 일반적인 프로젝트와 달리, 게임 프로젝트는 거대 용량의 미디어 파일들이 다수 포함되어 있으므로, 이러한 점은 부담스러울 수 밖에 없다.

이를 해소하기 위한 방법으로, Git 이나 Mercurial 에서는 Subrepository 같은 하위 저장소 개념을 제공하고는 있다. 그렇지만, 이 역시 사전에 디렉토리 구조를 완벽하게 고정하지 않으면 활용하기가 어렵고, 사용 개념도 더 복잡해져서, (이건 루트 저장소이고 저건 하위 저장소라는 걸 다른 사람들이 얼마나 손쉽게 알게 할 수 있을까?) 사용법에 관한 교육 문제를 더 심화시킨다.

- **E-1-1-2.** 분산 방식 버전 컨트롤 시스템(Distributed Version Control System. DVCS)을 사용하는 것은 가능하지만, Subversion으로 구성한 중앙 저장소로부터 분기를 얻어와서 작업하는 방식이어야 하며, 이를 프로젝트의 정식 중앙 저장소로 취급할 수 없다.
: 즉, 이는 개인 용도, 임시 저장소로 사용하는 용도가 적합하다.

※ 가장 널리 사용되는 DVCS 인 Git 의 경우에는 이러한 '출장 작업'에 용이한 기능을 제공한다.
Git 은 SVN 저장소로부터 저장소를 복제할 수 있고, 이렇게 복제한 Git 로컬 저장소에서 변경 내역을 커밋한 뒤, 그 이력을 다시 SVN 저장소에 그대로 반영해주는 기능이 있다.

- **E-1-1-3.** 중앙 저장소는 어느 개발자의 개인 사용 용도가 아닌, 공용으로 지정한 PC에 설치해야 한다.
: 특정 개발 환경에 의존하게 되는 위험성도 있고, 저장소가 설치된 PC는 24시간 내내 켜놔야 하기 때문에 그렇기도 하다.

- **E-1-1-4.** 프로젝트의 중앙 저장소에 대한 접근(저장소 내 파일들의 읽기, 쓰기)은 관리자가 허가한 사용자들에게만 허용해야 한다.
: 즉, 중앙 저장소에 접근하는 사용자들은 **고유의 계정과 비밀번호를 보유**해야 한다.

- **E-1-1-5.** 저장소는 **'프로젝트마다 1개만'** 존재해야 한다.

※ 저장소를 회사에서 단 한 개만 유지하는 방식도 심각하게 고민을 했었다.
저장소는 많이 유지할 필요가 없으며, 저장소의 Root 가 많을수록 사람이 수동으로 기억하고 관리해야 할 작업만 쓸데없이 많아질 뿐이다. 실제로, Apache 프로젝트의 경우, 수많은 하위 프로젝트들을 전부 단 한 개의 SVN 저장소에서 관리한다. (Apache 전체의 저장소 Root -> 각 프로젝트의 Root 디렉토리 -> 분기 디렉토리 -> 프로젝트 내용물의 구조로 되어 있다.)
저장소가 단 한 개인 경우에는 관리와 백업 절차가 아주 쉬워지고, 공통 모듈을 관리하기도 좋다.
그럼에도 불구하고 프로젝트 별로 저장소를 별도로 유지하기로 한 이유는, 새로운 프로젝트가 저장소에서 리비전 1 번부터 시작하게 하고 싶어서이다.(...)

- **E-1-1-6.** **같은 프로젝트에 대하여, 여러 개의 저장소를 생성하는 행위를 금지**한다.
: 프로젝트의 각 분야 별로 저장소를 따로 둬서는 안 된다.
저장소는 프로젝트 당 한 개만 있어야 하며, 각 분야 별 구분은 저장소의 하위 디렉토리로 구분해야 한다.

※ 주로 많이 발생하는 경우가, 클라이언트 프로젝트, 서버 프로젝트가 저장소를 따로 쓰겠다고 하거나, 그래픽 데이터용 저장소를 따로 두는 등의 상황이 벌어지곤 한다.

이렇게 되는 이유는 몇 가지 있는데, 우선 저장소를 올바르게 쓰기 위한 방법을 잘 모르고, (대개 이런 경우에는 저장소 내부에 분기를 위한 디렉토리 구조를 둬야 한다는 사실조차도 모르는 경우가 많다.) 그것 말고도 정치적인 이유(...)로 인해 저장소를 분리하기를 원하는 개발자들도 있다. 특히 프로그래머들 중에서 그런 사람들이 많은 편이다.

그들이 대는 정치적인 이유 중에는 대개 보안을 핑계 삼는 경우가 많다.
이유도 가지가지여서, 분야랑 상관없는 사람이 내 소스코드에 접근하지 못하게 해야 한다거나, 소스코드가 유출될 우려가 있다는 등의 근거를 대는 경우가 많다.
그러나 이는 올바른 근거라고 볼 수는 없고, 그냥 100% 핑계에 가깝다고 보면 된다. (...)

저장소를 자기만 접근할 수 있게 분리해야 한다고 하면서, 정작 그 저장소 자체는 다른 저장소들과 같은 컴퓨터의 디스크에 저장되어 있는 경우가 99% 이상이다. 게다가 저장소가 설치되어 있는 컴퓨터에는, 전혀 물리적인 보안 장치가 되어 있지도 않은 경우가 절대 다수다.
저장소를 정말로 그렇게나 보호하고 싶다면, **저장소가 설치되어 있는 컴퓨터에 대해 물리적인 보안을 취하는 게 우선**이다.
여기서 말하는 물리적인 보안이란 의미는, 저장소 서버 컴퓨터를 허가 받은 인원들만 출입할 수 있는 별도의 보관 장소에 둔다든가, (가장 확실한 방법이다.) 저장소 서버의 컴퓨터 케이스 자체에 잠금 장치, 도난 방지 장치 등의 설비를 갖추는 걸 말한다.
이런 것도 안 되어 있으면서 애먼 프로젝트의 저장소만 백날 분리해봐야, 저장소가 있는 스토리지 그 자체를 털어버리면 무용지물에 불과하다.(...)
그럴 거면 차라리 Github 이라든가, CodeProject, Bitbucket 등의 웹 기반으로 된 저장소 서비스를 이용하느니만 못하다. 그런 건 적어도 물리적으로 털리지는 않는다. (...)

소스코드 유출과 도용 문제에 대한 우려도 사실은 기우에 불과하다.
소스코드를 이용하지 못하는 사람은, 가져가봐야 어차피 텍스트 파일에 쓰여진 알 수 없는 영어 단어와 기호들의 나열에 불과할 뿐이다. 그리고 소프트웨어 프로젝트라는 건, 항상 끊임없이 유지보수가 이루어져야 하고, 그 프로젝트를 제작한 팀의 경험 자산이 곧 그 유지보수의 원천이다.
소스코드는 단지 그 결과물 중 하나일 뿐이며, 프로젝트 그 자체가 결코 아니다. 소스코드만 가지고서 그걸 분석하고 내재화해서 빌드 환경과 구성까지 완벽하게 갖춰 프로젝트를 베껴낼 것 같으면, 그 노력이나 아예 새 프로젝트를 새로 시작해서 제작하는 노력이나 거기서 거기다.
그리고, **게임 프로젝트에서는 오히려 소스코드 도용보다도 기획 컨셉 및 아트 데이터의 도용이 훨씬 빈번하고 심각**한 문제다.
해킹이나 취약점 분석에 유출된 소스가 이용될 수 있다고 주장하는 경우도 있다.
근거가 아주 없는 건 아니지만, 김칫국을 너무 거하게 들이켰다고 봐야 한다.
일단, 그런 거 걱정하기 전에 해킹을 할 정도로 가치가 높고 잘 팔리는 게임을 만드는 게 우선 아닌가? (아무도 안 하는 게임을 누가 해킹하려고 드나?)
그리고 프로그래밍에서의 보안은 취약점을 감추는 게 보안이 아니라, 취약점이 없게 만드는 게 보안이다. 설령 소스코드를 얻어다가 옆에 펼쳐놓고 분석을 하더라도, 해킹이나 치팅을 못하게 만드는 방법을 궁리하는 게 맞는 것이다.

결론적으로, 불필요한 저장소 분리 및 과도한 접근 권한 관리는 실질적으로 프로젝트에 이득을 주지도 않는다. 괜히 관리 노력만 증가시킬 뿐이며, 관리하기도 어렵고, 유지보수에도 별 도움이 안 된다.

그러니, 쓸데없는 데 개발 노력을 낭비하지 말고, **프로젝트에 참가한 인원들에게는 일하는 데 불편하지 않도록, 그 프로젝트의 저장소를 개방해줘야 한다.**

- **E-1-1-7.** 모든 프로젝트의 저장소는 반드시 저장소의 최상위 디렉토리 아래에 **Main, Publish, Research**의 3개 디렉토리를 가진다.

·○ **Main** = 프로젝트의 주 분기로써, 대부분 작업은 여기에서 이루어짐

·○ **Publish** = Main에서 작업하던 내용을 특정한 버전 시점으로 보존하기 위한 분기.
: 보통 Publish 분기의 내용은 Main으로 다시는 병합하지 않는다.

·○ **Research** = 개인적인 연구, 장기간의 아키텍처 개선, 핫픽스 등의 이유로 프로젝트를 분기(fork)할 때, Research 디렉토리의 하위에 분기한다.

※ SVN 에서 통상적으로 쓰는 관용어는 trunk, branch, tag 이다.
이는 각각 Main = trunk, Research = branch, Publish = tag 에 대응한다.

◆ E-1-2. 버전 관리 시스템 사용 규칙

- **E-1-2-1.** 사용자 등록을 위한 ID는 특수문자 없이 소문자로만 이루어진 영문 실명을 사용해야 한다.
: 홍길동 씨의 ID는 honggildong으로 사용해야 한다.

※ 실명 이름이 아닌, 가상의 이름으로 ID를 사용하는 방식은 오픈소스 프로젝트가 아닌 한, 회사 내부의 프로젝트에서는 권장하지 않는다.
왜냐하면, ID만 보고서 어떤 ID가 누구인지 일일이 외우고 있어야 하기 때문이다. 이슈 관리 시스템과 연동이 되어 있으면 추적이라도 할 수는 있지만, 만약 직원이 수백 명이라면 가상의 이름으로 만든 ID가 실제로 어느 이름의 직원인지 얼른 추적하는 건 불가능하다.
외부에 노출해야 하는 저장소인 경우라면, 당연히 사생활 보호 차원에서 가상의 이름으로 ID를 써야겠지만, 회사 내부에만 공개하는 저장소에서는 신속하게 작업자를 추적하는 게 우선이므로 실명 이름으로만 ID를 만드는 규칙을 강제하는 것이다.

단점은 동명이인인데 영문 이름까지 완전히 같은 경우이다(...).
그렇지만, 일단 이런 문제가 발생할 확률이 희귀하므로, 지나치게 걱정하지 않아도 될 문제라고 본다.
만일 진짜로 문제가 발생하는 경우, honggildong2, honggildong3 등으로 구분하는 정도로 충분할 것이다. 이름 뒤에 숫자 꼬리가 달리는 사람은 좀 기분 나쁘겠지만, 보통 그럴 때 하는 군대 유머가 있지 않은가? (억울하면 먼저 들어오든지...)

- **E-1-2-2.** VCS의 사용자 비밀번호는 영어 대 / 소문자와 숫자, 특수 기호를 조합해서 만들 수 있다.

- **E-1-2-3.** VCS 사용자 비밀번호의 길이에 대해 특별한 제약은 없다.
  : 그러나, 해킹이나 강제 대입 시도(Brutal Force Attack)를 방지하기 위해 최소한 6자 이상의 영어 대 / 소문자, 숫자, 특수 기호를 모두 조합해서 만들 것을 강력하게 권한다.

- **E-1-2-4.** 제작 및 수정을 할 수 있는 형식의 데이터들만 저장소에 포함해야 한다.
  : **저장소에 있는 데이터들만 가지고서 프로젝트를 똑같이 생성하고, 결과물(게임 실행 파일과 데이터들)도 완전히 똑같이 만들어낼 수 있는 상태가 되어야 한다.**

※ 소스 파일로부터 만들어낼 수 있는 빌드 결과물이나, 수정이 불가능한 변환 파일들(JPEG, PNG 등의 이미지 파일, 제작이 완료된 상태의 동영상 재생 포맷 등)은 저장소에 커밋하는 대상이 아니다.
커밋을 해야 하는 파일과 아닌 파일을 구분하는 방법 중 하나는, 그 파일을 편집 / 수정해서 내용을 갱신하거나 바꿀 수 있는 파일인지 여부이다.
예를 들어, JPG, PNG 등의 포맷은 그림을 보여줄 수만 있을 뿐, 편집이나 수정은 불가능하다.
따라서 반드시 이미지들은 PSD 등의 제작용 포맷으로 커밋해야 한다. 다른 파일들 역시 마찬가지다. 문서 포맷을 PDF 변환 파일만 커밋하고, 제작에 사용했던 DOCX, PPTX, XLSX, HWP 등의 오피스 소프트웨어 포맷을 커밋하지 않는다면, 저장소를 사용하는 게 아무런 의미가 없는 셈이다..

- **E-1-2-5.** **사용자에 따라 달라지는 환경 설정 파일들을 저장소에 포함하지 않는다.**
  : IDE 사용자 설정 파일, 워드 프로세서의 임시 저장 파일 등이 대표적으로 여기에 해당한다.

※ 사용자 설정 파일들을 커밋하게 되면, 체크아웃 받은 컴퓨터마다 내용이 변하고, 심지어는 프로그램을 실행할 때마다 설정 파일이 변경되기 때문에, 프로젝트는 항상 뭔가를 커밋을 해야만 하는 상태가 되어 버린다.
그렇다고 해서 그런 환경 설정 파일들이 프로젝트 진행에 필수적으로 필요한 파일인 것도 아니다. 이런 파일들은 VCS가 무시하도록 설정하거나, 아예 커밋 요소에 포함시키지 않게 해야 한다.

예를 들면, ©Microsoft Visual Studio의 *.user, *.suo 파일들은 Visual Studio의 컴퓨터 별 환경 설정 파일들이므로(이런 파일들은 대개 저장할 때마다 파일의 내용이 바뀐다.), 이런 파일들은 커밋하면 안 된다.
반면 *.sin, *.vcproj은 필수적인 파일들이고, 컴퓨터 별로 달라지지도 않는 파일들이기 때문에 커밋 대상이다.

워드 프로세서나 텍스트 파일들은 '~', '$', '#'등의 문자가 파일 이름의 맨 앞에 붙는 경우, 대개 자동 저장을 위한 임시 파일들이다. 이런 파일도 커밋하는 대상이 아니다.

Unity3D 엔진 프로젝트 파일의 경우를 살펴보자.
Assets 디렉토리의 내용은 필수적이기 때문에 포함해야 한다. 이름 그대로, 프로젝트의 에셋들을 담고 있기 때문이다.
ProjectSetting 디렉토리는 해당 Unity 프로젝트의 설정 정보를 담고 있기 때문에 꼭 커밋해야 한다.
하지만, Library, obj, Temp 디렉토리의 내용은 Unity 엔진이 프로젝트를 열 때마다, 혹은 빌드할 때마다 자동으로 생성하는 파일들이고, 그 파일 내용도 Unity 에디터를 실행하고 있는 컴퓨터마다 조금씩 달라지기 때문에(이진 체계 단위에서의 차이이기 때문에 실질적으로 파일의 내용면에서는 동일하다.) 저장소에 커밋하지 않는다.

◆ E-1-3. 커밋 규칙

- **E-1-3-1.** 저장소에 커밋을 하기 전에, **먼저 커밋할 내용에 대한 이슈를 이슈 관리 시스템에 등록**해야 한다.

※ 모든 커밋에는 커밋에 대한 이유가 있어야 하고, 이를 설명하는 내용과 관리하는 번호가 있어야 한다.
즉, 모든 커밋에는 몇 번 이슈에 대한 사항인지를 명시할 수 있어야 한다.

- **E-1-3-2.** 커밋할 때, **반드시 로그 메시지를 적어야 한다.**
: 빈 메시지로 커밋하면 안 된다.

- **E-1-3-3.** 커밋 로그 메시지에는 어떤 이유로 무슨 항목을 수정했는지(혹은 추가 / 삭제했는지)에 대한 내용이 들어 있어야 한다.
커밋 로그 메시지의 첫 머리에는 반드시 이슈 관리 시스템 상에서의 이슈 번호를 표시해야 한다.
: 이것은 246번 이슈라는 것을 나타내어야 한다.

- **E-1-3-4.** 이슈를 시작하거나 진행 중인 커밋 사항들은 아래와 같은 형식으로 커밋 메시지를 시작해야 한다.

issue #○○○ <--- 이슈 관리 시스템에서의 이슈 번호
수정한 내용(XX를 이렇게 저렇게 수정하였다.)

- **E-1-3-5.** 이슈를 완료하였을 때의 커밋 사항들은 아래와 같은 형식으로 커밋 메시지를 시작해야 한다.

fixed #○○○ <--- 이슈 관리 시스템에서의 이슈 번호
수정해서 해결한 내용(XX를 이렇게 저렇게 수정해서 해결하였다.)

- **E-1-3-6.** 프로젝트가 컴파일 되지 않거나 빌드 되지 않는 상태로 커밋해서는 안 된다.
: 이는 프로그래머를 대상으로 한 규정이라고 보면 된다.
실행 시간 중의 오류는 테스트 단계에서 미처 잡아내지 못할 가능성이 있지만, 컴파일 / 빌드 오류는 최소한의 확인도 안 한 것이므로 용서해서는 안 된다.

※ 이 규칙을 위반한 프로그래머는 다음 프로그래머 위반자가 나올 때까지 프로젝트의 공식 빌드와 배포를 전담해야 한다.

◆ E-1-4. 분기(Branch) / 태깅(Tagging) 규칙

- **E-1-4-1.** 프로젝트의 주 버전(Major Version) 또는 부 버전(Minor Version)이 변경되는 경우에는 해당 시점에서 Publish 디렉토리에 Main 디렉토리의 내용을 태깅해야 한다.

- **E-1-4-2.** Publish 디렉토리에 버전 태깅을 하는 경우, Main 분기의 내용이 하나도 빠짐없이 태깅 되어야 한다.
: 즉, Main 디렉토리 전체를 태깅해야지, Main 분기 안의 일부 디렉토리만 태깅하는 짓을 하면 안 된다.

※ 특히, 클라이언트 담당하는 프로그래머들이 버전 태깅하라고 하면 클라이언트 프로젝트의 소스코드들만 달랑 태깅하는 사고를 치곤 한다.
태깅은 원칙적으로 **해당 버전을 제작했던 당시의 환경을 그대로 보존**하고자 하는 목적을 가지고 하는 행위이다. 즉, 프로그램 소스코드와 빌드 스크립트 뿐 아니라, 그 당시의 기획서, 명세서, 아트 데이터 등의 모든 제작 자산들을 보존해야 한다.
특정 버전의 태그 저장소를 체크아웃해서, 그 버전의 모든 게임 자산들과 실행파일, 실행환경을 만들어내지 못한다면, 제대로 된 버전 태깅이 아니다.

- **E-1-4-3.** Publish 디렉토리에 태깅할 때의 디렉토리 이름 규칙은 다음과 같다.
: Publish 디렉토리 아래에 'Version_X_XX'처럼 버전 번호를 이름으로 가지는 디렉토리를 만들고, 그 디렉토리에 해당 시점의 모든 Main 내용이 들어가게 한다.

- **E-1-4-4.** Research 디렉토리에 분기할 때는, Research 디렉토리의 하위에 직접 분기하지 말고, 하위에 디렉토리를 만들어서 분기해야 한다.
: 왜냐하면, 분기는 여러 개가 생길 수도 있기 때문에, 그 각각을 구분해서 체크아웃할 수 있는 방식으로 분기를 만들어줘야 한다.

◆ E-1-5. 병합 규칙

- **E-1-5-1.** 저장소로부터 업데이트를 받았을 때, 한 개 이상의 파일이 충돌이 났다면, 충돌을 반드시 해결하고 커밋 해야 한다.
  : 충돌이 일어난 상태를 해결하지 않고 커밋 하면 안 된다. (요즘엔 SVN 시스템에서 가능하지도 않겠지만...)

※ 이 규칙을 위반한 프로그래머는 다음 프로그래머 위반자가 나올 때까지 프로젝트의 공식 빌드와 배포를 전담해야 한다.

- **E-1-5-2.** 저장소 충돌을 해결할 일차적인 책임은, 충돌 현상을 발견한 당사자에게 있다.
  : 저장소 충돌을 발견한 사람은 이를 해결해야만 커밋이 가능하므로, 결과적으로 먼저 커밋한 쪽이 승리자다.(...)

◆ E-1-6. 개발 분기 관리

- **E-1-6-1.** 개발 분기는 다음과 같은 분류로 나눈다.

  개발자 테스트
  비공개 테스트
  공개 테스트
  검수 대기
  공개 서비스

- **E-1-6-2.** 서버는 각 개발 분기에서 허용하는 최신 버전 번호를 인식하고, 이에 따라 자동으로 어떤 개발 분기의 서버나 데이터베이스에 연결해야 할지를 스스로 판단하는 기능을 구현해야 한다.

※ 특히, 마켓 검수에 들어가는 애플리케이션은 통과 '즉시', 곧바로 서비스에 들어가기 때문에 이러한 자동 전환 과정은 필수적이다. 검수용 애플리케이션와 출시용 애플리케이션을 따로 적용할 수 없을 가능성이 높다.

- **E-1-6-3.** 다만, 개발 과정에서는 수동으로 전환할 수 있는 인터페이스를 제공해줄 수 있다.
  : 이건 어디까지나 개발 초기 단계에서의 편리를 위한 사항이고, 궁극적으로는 개발 분기별로 버전을 설정해서 활용해야 한다.

E-2. 코딩 규칙

◆ E-2-1. 준수 사항

- **E-2-1-1.** **코딩 규칙은 전사적으로 모든 소스 코드와 일반 텍스트 파일에 강제적으로 적용**해야 한다.
: 회사 차원에서 정한 코딩 규칙은 특별한 이유가 없는 한 반드시 준수해야 하며, 여기에 어떤 예외도 두지 않는다.
아주 가끔은 어쩌다가 어길 수도 있겠지만, 그건 정말로 '아주 가끔'이어야만 한다.

※ 어차피 개발자마다 코딩 스타일이 미묘하게라도 제각각 다르고, 이에 대해 정해진 정답은 따로 없다. (Python 처럼 언어 차원에서 문법적으로 정해두고 시작하는 경우는 예외로 하자.)
하지만 특히 **코딩 스타일이 극단적으로 다른 경우의 개발자들은, 서로의 편집 스타일에 적응하지 못해서 코드 분석에 노력 낭비가 들어간다.** 그 때문에 일부러 자신이 선호하는 스타일로 다른 사람이 짜 놓은 코드를 편집하기도 하고, 텍스트 에디터의 매크로나 IDE 의 설정에 따라 그러한 코드가 자동으로 재조정되기도 한다.
이러한 작업이 반복되다 보면, 텍스트 병합 도구에서도 '탭 – 공백' 차이와 같은 의미 없는 비교 정보가 넘쳐나게 되고, 심지어 개발자 간에 감정까지 상하게 되기도 한다.
그러니, 모든 사람을 만족시킬 수는 없겠지만, 코딩 스타일 견해 때문에 개발 노력을 낭비하지 않도록, 회사 차원에서 특정한 기준을 정하고, 모든 개발자들이 강제로 이를 따르게 하는 것이다.

- **E-2-1-2.** 이를 위해 사용하는 텍스트 편집기(Editor)나 통합 개발 환경(IDE)의 설정을 일괄적으로 특정한 설정으로 맞춰서 쓰도록 강제할 수 있다.

※ 편집기 설정을 일괄적으로 통일해야, 자동으로 회사 표준이 아닌 방식으로 수정되는 경우를 막을 수 있다.
이 규칙이 없을 때, 눈치채기 어려우면서 특히 심하게 나타나는 게 '탭 - 공백' 설정이다. 누구는 탭으로 되어 있고 누구는 공백으로 되어 있으면, 눈으로는 잘 안 보이지만 텍스트 병합 도구에서는 의미도 없는 차이점 정보들로 가득 보이게 된다.

- **E-2-1-3.** **개인적인 판단으로 회사 코딩 표준과 다른 양식을 사용하지 않는다.**
: 특히, 자신에게 익숙하고 편안하다는 이유로 수시로 회사 코딩 표준을 어기면 제재 대상이다.

- **E-2-1-4.** 코딩 표준에 이의가 있다면, 정식으로 요청해서 **개발팀 합의 하에 코딩 표준을 수정**

해야 한다.

※ 그 어느 양식에 관한 표준이라도 마찬가지 문제를 안고 있듯, 소스 코딩 표준도 개발자 모두를 만족시킬 수는 없다.
관성에 따라 잘못 사용하고 있는 표준도 있고, 처음에는 괜찮았는데, 나중에는 구성원들이 모두 불편해 하기 때문에 바꿔야 하는 표준도 있을 수 있다 ~~(아니면, 멋있는 방식이어서 바꾸는 게 매력적인 경우도 있을 수 있고...)~~
. 오해할까 봐 덧붙이자면, 코딩 표준은 절대 수정할 수 없는 금석문이 아니다. **합리적인 이유에 의해 언제든 수정할 수 있는 사항**들이다.
즉, **자의적으로 판단해서 혼자만 쓰는 코딩 규칙을 만들지 말라**는 의미이다.

- **E-2-1-5.** 코딩 표준을 수정한 경우, 기존 프로젝트의 소스 코드들을 새 표준에 맞춰서 모두 수정한 이후에 새로운 작업을 시작해야 한다.

※ 이러한 강제를 두는 이유는, 우선적으로 회사의 코딩 표준이 모든 소스 코드와 텍스트 파일에 대해 공히 적용된다는 사실을 확실히 하기 위해서다. ~~(소스 코드 A는 버전 1.0 표준이고 B는 버전 1.2 표준이면 이상하잖아)~~
두 번째 이유는, 회사의 코딩 규칙을 처음부터 잘 설계해야 하며, 이를 손쉽게 바꾸려는 시도를 방지하기 위함이다. 특히, 개인적인 취향 때문에 코딩 표준을 이리저리 바꾸고 싶어하는 시도를 차단하고자 함이다. ~~(바꾸자고 한 사람이 코드 다 수정하기... 이런 거?)~~

◆ E-2-2. 일반 코딩 규칙

- **E-2-2-1.** 정확한 샘플 코딩 규칙은 Document₩Policy에 있는 'CodingConvention'이라는 이름이 붙은 소스 파일들을 참조한다.
: 세세한 내용은 말로 설명하는 것보다 샘플 소스 코드 예제를 보는 게 더 이해가 빠르다.

- **E-2-2-2.** 시스템 헝가리언 표기법(System Hungarian Notation)은 사용하지 않는다.
: float 변수 접두어로 f 붙이거나, int 변수 접두어로 n 따위를 붙이는 방식을 말한다. 이런 코딩 방식은 사용하지 않는다.

※ 위 방식은 Charles Petzold 등에 의해 널리 전파되기는 하였으나, 원래 헝가리언 표기법을 창안한 Charles Simonyi 의 원래 의도와는 전혀 맞지 않게 사용되었다고 한다. (이에 대한 자세한 사항을 알고 싶다면, Joel Spolski 의 [More Joel On Software]를 참고하기 바란다.)
참고로, ©Microsoft 에서도 더 이상 내부에서든 외부에서든 개발자들에게 헝가리언 표기 방식을 권장하지 않는다.

※ 이 규칙을 위배하는 경우가 하나 있는데, 바로 인터페이스(interface) 객체 부분이다.

인터페이스 객체의 이름에는 '(프로젝트 접두어) + I(아이)'를 붙이는 식으로 사용하고 있다.
Robert Martine 이 쓴 책인 [Clean Code]에서는 인터페이스 객체 이름의 접두어로 I 를 붙이는 관례를 이상하고 쓸데없는 관례라고 평하고 있긴 하다.
그렇지만, 비슷한 기능의 대리 객체 버전과 인스턴스 객체 버전이 필요할 때는 같은 이름을 기반으로 하는 클래스와 인터페이스를 다 만들어야 하는 경우도 종종 생긴다.
그래서 고민 끝에, 인터페이스만 예외적으로 I 접두어를 쓰기로 했다.

- **E-2-2-3.** 중괄호 스타일은 다음과 같은 방식을 사용하도록 통일한다.

```
if(Condition)
{
    DoSomething
    ....
}
```

※ if(condition) {
DoSomething
...
}

위와 같은 K&R 스타일을 고수하는 개발자들도 여전히 아주 많기는 하다.
, 하지만, 개인적인 생각으로는 지금 시대에는 좀 구식인 스타일이라고 생각한다. ~~(모니터가 80 컬럼이던 시대에는 유용했겠지...)~~

- **E-2-2-4.** 탭(Tab)은 모두 공백(Space)으로 바꿔서 쓴다.

- **E-2-2-5.** 모든 탭 간격은 공백 4칸(4 – Space)이다.

※ 간혹 XML 이나 JavaScript 등이 들여쓰기가 많다고 해서 탭을 임의로 2 칸으로 설정하거나, emacs 편집기의 기본 설정이라고 해서 탭을 8 칸으로 설정한 채 그대로 쓰면 안 된다.
편집기 설정이 그렇게 되어 있었다면, 편집기의 설정 내용을 바꿔줘야 한다.
코딩 규칙에서 별도의 예외사항을 명시하지 않고 모든 탭 간격이 공백 4 칸이라고 했다면, 텍스트 파일의 형식이나 언어에 관계없이 모두 탭을 공백 4 칸으로 적용해야 한다.

- **E-2-2-6.** 모든 소스 코드와 텍스트 파일은 **'서명 없는 UTF-8 인코딩(No Byte Order Mark(BOM) UTF-8, Code Page-65001)'** 방식으로 저장해야 한다.

- **E-2-2-7.** 단, **Unity에서 사용하는 소스 코드 파일**들은 **'서명 있는 UTF-8 인코딩(Code Page-65001)'로 저장**해야 한다.

: 이는 MacOSX 버전의 Unity에서, BOM 없는 UTF-8 소스 코드의 텍스트를 제대로 해석하지 못하는 경우가 발생하기 때문이다.

※ BOM 은 Byte Order Mark 라고 해서, 텍스트 파일의 가장 첫 번째 이진 문자로 오는 특수한 기호이다. 텍스트 파일의 인코딩 규칙을 알리는 역할을 한다.
다만, UTF-8 형식의 경우에는, 기존 Ansi 기반 문서와도 호환이 되기 때문에(영어 기준일 뿐이다. 영어 아닌 문자들이 있으면 당연히 호환 안 됨), BOM 이 굳이 필요 없다는 진영과, BOM 을 넣는 것을 권장하는 진영으로 갈려 있다. 전자는 Unix-like 진영(유닉스, 리눅스 등)에서 밀고 있고, 후자는 Microsoft 가 밀고 있다.
순수하게 추세로만 보자면, BOM 없는 UTF-8 이 대세라고 볼 수 있다. 그래서 일반적으로는 BOM 없는 UTF-8 텍스트 파일을 기본으로 하되, 문제가 있는 예외적인 경우에는 다른 인코딩을 적용하기로 한다.

- **E-2-2-8.** 모든 변수들은 낙타 표기법(Camel Notation)을 사용한다.
: 이름의 가장 첫 글자가 소문자로 시작하고, 띄어쓰기 대신 다음 단어의 첫 글자를 대문자로 표기한다.
이는 클래스 / 구조체의 멤버 변수(필드)부터 시작해서, 지역 변수와 정적 변수에도 공통적으로 적용한다.

- **E-2-2-9.** 변수를 제외한 이름공간(namespace) 명칭과 클래스 / 구조체 / 인터페이스의 이름, 함수 이름, 열거형 이름, 상수 이름은 이름의 가장 첫 글자가 대문자로 시작하는 낙타 표기법을 사용한다.

- **E-2-2-10.** 전처리기의 이름은 대문자로만 표기하며, 단어와 단어 사이에는 언더 바(_)를 붙인다.

- **E-2-2-11.** 모든 이름들은 **최소한 3글자 이상**이어야 한다.
: I, j, k 등 단문자 변수 이름을 사용하지 말 것.

※ 완전히 수학적인 함수를 작성하는 경우나, 일반화된 알고리즘에는 매개변수를 한 글자로 쓰는 경우도 종종 있으므로, 완전 강제 사항까지는 아니라고 할 수 있다.
하지만 한 글자 변수는 가급적 피하는 게 좋은데, 첫 번째로는 이름의 의미를 알리기가 어렵고, 두 번째로는 마우스로 한 번에 찍어서 선택하기가 힘들다.(...)

- **E-2-2-12.** 변수의 이름이 20글자를 넘어가지 않는 한, 가급적이면 **단어를 축약한 표기를 쓰지 않는다.**
: Sprite -> spr로 표기하거나 Transform -> tf로 표기하는 행위 등을 말한다.

※ 축약어를 남발하면, 처음 코드를 읽는 사람이 코드의 내용을 얼른 파악하기가 힘들어진다.

변수 이름의 길이에 대해 달리 제약이 없다면, 변수에 사용할 단어는 되도록 축약형을 쓰지 않고 원래 단어의 글자 그대로 사용한다.
개발자들 간에 암묵적으로 쓰는 축약 표현들은 상황이 좀 낫지만, **모든 개발자가 그런 축약 표현들을 미리 알고 있으리라고 가정하면 안 된다.**
축약 표현을 사용하기 전에, 꼭 그렇게 해야할 필요가 있을지 한 번 더 신중하게 생각해볼 것.

- **E-2-2-13.** 선언한 이름의 길이가 길어서 축약어를 사용하는 게 불가피하다면, 이 축약어의 의미를 파악할 수 있는 주석을 명시해줘야 한다.
: 이러한 축약어 주석은 ReadMe.txt 파일이나, 해당 코드의 가장 위에 별도의 용어집 주석을 달아서 쉽게 찾아볼 수 있게 해야 한다.

- **E-2-2-14.** 모든 소스 코드 파일의 이름에는 프로젝트를 가리키는 약자를 접두어(Pre-fix)로 두는 방식으로 이름을 짓는다.
: Evil Of Darkness이므로 소스 코드 파일들은 "EodSource.c", "EodSource.cs", "EodSource.java" 식으로 짓는다.

◆ E-2-3. 프로젝트 전용 코딩 규칙

- **E-2-3-1.** 이하 내용들은 프로젝트 S에서만 전용으로 적용하는 코딩 규칙이다.

- **E-2-3-2.** 모든 소스 코드들은, 각자 해당 언어에 맞는 샘플 소스 코드의 형식을 응용해서 사용한다.
: C# 소스 코드의 경우 PsSample.cs 소스 코드가 프로젝트에 포함되어 있으며, 이를 복제해서 수정해서 쓰면 된다.

- **E-2-3-3.** 소스 파일을 새로 추가해야 할 경우, 직접 생성하기보다는 프로젝트에 동봉되어 있는 샘플 소스 파일을 복제한 뒤, 수정해서 사용하는 방식을 권장한다.

※ 샘플 소스 파일은 프로젝트 소스 코드가 일반적으로 준수해야 하는 규격 사항과 텍스트 인코딩 규칙이 정확히 정의되어 있다.
직접 소스 파일을 생성해서 사용한다면, 아무래도 사람이 하는 일이니만큼, 이러한 규칙들을 깨먹기 쉬워진다.
그러니, 가급적 샘플 소스 파일을 복사해서 사용하고, 만일 없다면, 샘플 소스 파일을 하나 만들어서 저장소에 등록해주면 모두가 편리하게 코딩 규칙을 지킬 수 있다.

- **E-2-3-4.** 모든 소스 파일들은 파일 이름 앞에 Eod 접두어를 붙여야 한다.
: Evil Of Darkness의 소스 파일임을 나타낸다.

- **E-2-3-5.** UnityEngine.MonoBehaviour를 상속받는 클래스를 정의하는 경우, 클래스의 이름과 파일의 이름이 반드시 일치해야 한다.
: 이는 Unity 엔진의 요구사항이기도 하다.

※ 이 규약을 어기는 경우, MonoBehaviour 상속 클래스를 UnityEngine.GameObject에 컴포넌트로 붙였을 때, 유니티 에디터의 인스펙터에서 그 내용이 정확히 표시되지 않는다.
이렇게 된 경우, public 선언한 매개 변수들을 조작할 수 없음은 물론이고, 붙인 클래스가 여러 개인 경우, 각각의 이름조차도 구별을 할 수 없게 된다.
하여튼, 이 부분은 게임 엔진에서의 제약 사항이니만큼, 사용자가 따라줘야 한다.

- **E-2-3-6.** UnityEngine.MonoBehaviour를 상속받거나, 전역 이름 공간에 프로젝트 관련 클래스를 정의하는 경우, 그 클래스의 이름 앞에는 프로젝트 약자를 접두어로 두는 방식으로 이름을 지어야 한다.

```
class PsCustomClass : UnityEngine.MonoBehaviour
{
    DoSomething
    ....
}
```

- **E-2-3-7.** UnityEngine.MonoBehaviour를 상속하지 않는 클래스를 정의할 경우, 이 클래스들은 프로젝트의 이름을 딴 이름 공간(Namespace)에 속해야 한다.

```
namespace ProjectName
{
    class CustomClass
    {
    }
}
```

- **E-2-3-8.** 이름 공간에 정의하거나, 다른 클래스에 속한 하위 클래스를 정의할 때는, 그 클래스의 이름에 프로젝트 접두어를 붙이지 않아도 된다.
: 어차피 이름 공간, 혹은 상위 클래스 이름을 타고 들어가야 하기 때문에, 불필요한 정보 중복이다.

```
namespace ProjectName
{
    class CustomClass
    {
```

```
            class SubClass
            {
            }
        }
    }

    class PsCustomClass : UnityEngine.MonoBehaviour
    {
        class SubClass
        {
        }
    }
```

※ 이 모든 사항들은, 혹시 있을지 모르는, 소스 코드 파일 이름이나 객체 이름 등이 겹치는 상황을 방지하고, 어느 부분이 프로젝트에서 정의한 코드이고, 어느 부분이 외부 제품 코드, 혹은 표준 코드인지 구분할 수 있게 하기 위한 방편이다.
합의한 규칙에 맞게 일관성 있는 스타일로 코드를 작성해야, 코드를 읽는 사람들이 코딩 스타일 때문에 읽거나 의미를 파악하는데 어려움을 주는 부작용을 최소화할 수 있다. 그리고, 깔끔하다. (사실 이게 제일 큼)

E-3. 서비스 버전 관리

◆ E-3-1. 응용 프로그램 버전 정책

- **E-3-1-1.** 버전은 '(주 버전) . (부 버전) . (빌드 번호)'의 형식으로 나타낸다.

- **E-3-1-2.** 버전으로 사용할 체계는 가급적 한 가지 기준만 가지고 사용한다.

※ 실행 파일, 대상 플랫폼, 프로그래밍 언어에 관계없이, 버전 방식은 한 가지만 사용하는 것이 관리하기 쉽다. 기술적으로 그게 어려운 경우에는 어쩔 수 없겠지만...
(이 명세 문서조차도 버전 번호를 부여하는 방식은 응용프로그램에 탑재하는 것과 같은 체계를 사용한다...)

- **E-3-1-3.** 주 버전 번호(Major Version)
: 주 버전 번호는 게임의 주요 기능이 현저하게 바뀌거나 개선된 경우에 판올림한다.
특히, **데이터베이스의 스키마 혹은 테이블의 내용이 바뀌는 경우에는, 반드시 주 버전 번호를 갱신해야 한다.**

※ 데이터베이스의 필드 개수, 자료형 등이 바뀌게 되면, 일반적으로 더 이상 이전 버전과 통신할 수 없음을 의미하기 때문이다.

- **E-3-1-4.** 부 버전(Minor Version) 번호
: 주 버전 번호가 바뀌면 다시 0부터 시작한다.

- **E-3-1-5.** 빌드 번호(Build Number)
: 빌드 번호는 주 버전, 부 버전의 번호 교체와 관계없이, 항상 증가하기만 한다.
내부 빌드의 경우 홀수 번호를, **외부 출시 빌드인 경우 짝수 번호**를 사용해야 한다.

※ 빌드 번호의 부여 방식은, 업무 부하가 최소화되는 합리적인 선에서 결정한다.
: 현재는 암묵적으로 개발자들이 인정한 시점에서 부여하고 있으며, 해당 빌드 번호의 최종 결과물이 확정된 시점과, 다음 빌드 번호로 넘어가는 시점 사이에, 소스코드 관리 시스템의 로그에 그 기록을 남겨서 구분을 짓는다.

- **E-3-1-6.** 내부 빌드는 테스트 등으로 사용하며, 연속적으로 여러 번의 내부 빌드를 사용할 경우에는 빌드 번호를 2자리씩 건너 뛴다. 외부 빌드만 연속적으로 출시하는 경우에도 마찬가지

다.

※ 내부적으로 버전을 소통할 경우에는 축약해서 빌드 번호만을 가지고 소통할 수 있다는 장점이 있다. ('X쩜 ~ Y쩜 ~ Z 버전에서...'보다는 '빌드 번호 AAA에서...'가 말하기 훨씬 편하고 혼동도 덜 된다.)

또한, 내부 출시 빌드 번호와 외부 출시 빌드 번호가 구분된다는 점은, 해당 버전에서 문제가 발생했을 때, 이 이슈가 얼마나 심각한 것인지를 판단하는 데 있어 도움이 된다. (당연히 외부에 출시한 버전의 문제점이 최우선이다.)

- **E-3-1-7.** 최초 상용화 출시 버전의 기준은 **1.0.$(그 당시의 최신 빌드 번호)**이다.
그 이전 개발 단계에서의 빌드는 주 버전(Major Version)의 번호가 0이어야 한다.

※ 프로젝트의 소스 코드에는, 반드시 프로젝트의 버전을 정의하기 위한 소스 파일이 최소한 한 개는 포함되어야 한다.
또한, Unity3D의 Bundle ID 속성에도 버전 정보가 항상 버전 정의 소스 파일의 것과 일치하도록 유지하여야 한다.

- **E-3-1-8.** 최종 사용자가 실행 파일의 버전을 확인할 수 있는 수단을 제공해야 한다.
: 일반적으로는, 진입 장면의 배경화면 구석에 버전 번호를 표시하는 형식을 사용한다.

◆ E-3-2. 데이터 버전 정책

- **E-3-2-1.**

◆ E-3-3. 서버 버전 정책

- **E-3-3-1.**

◆ E-3-4. 데이터베이스 버전 정책

- **E-3-4-1.** 데이터베이스의 스키마, 혹은 테이블에 대한 내용 변경이 있다면, 해당 서비스의 모든 애플리케이션은 버전을 변경해야(올려야) 한다.
: 응용 프로그램이 아니고, 데이터베이스와 직접 관련이 없는 데이터들은 상황에 따라 고려할 수 있다.

- **E-3-4-2.** 데이터베이스의 스키마, 테이블, 저장 프로시저, 트리거 등, **해당 서비스의 데이터베이스를 구축하기 위한 모든 사항들은 SQL 구문, 혹은 명령행으로 가능해야 하고, 이를 별도의 파일로 보관해야 한다.**

※ 프로젝트의 버전 관리를 한다고 하는 수 많은 프로젝트들이, 응용 프로그램의 소스 코드만 저장소에 등록해놓고 있고, 정작 그 못지 않게 중요한 데이터베이스를 생성하는 방식을 보관하지 않고 있다.
그런 상황이라면, 데이터베이스를 개발하던 컴퓨터가 망가지는 등의 사고가 난다면 복구하기가 힘들거나, 심지어 거의 불가능해서 관련 기능을 처음부터 재구축하는 수 밖에 없을 것이다.

- **E-3-4-3.** 특정한 서비스 버전마다 각각 그 버전에서 통하는 데이터베이스를 불러올 수 있어야 한다.

※ 개발을 하거나, 서비스를 하다 보면, 예전 버전의 서비스를 돌려야 하는 경우가 생길 수 있다.
이 때 데이터베이스는 무조건 최신 포맷만 지원할 수 있다면, 구 버전의 서버 / 클라이언트 / 저작 도구 등의 응용 프로그램들은 사실상 무용지물이 될 수 밖에 없다.
따라서, 다른 응용 프로그램들을 버전마다 보존하는 것처럼, 데이터베이스의 포맷 역시 서비스 버전마다 보존해주는 작업이 중요하다.

- **E-3-4-4.** (이미 상당히 안정화된 이후에,) 데이터베이스의 스키마, 테이블, 저장 프로시저 등의 내부 구조와 내용을 변경한 경우에는, **구 버전의 DB가 새로운 버전의 DB로 이전할 수 있도록, DB를 변경했던 SQL 문과 명령문에 대한 내용들을 별도로 기록하고 보관**해야 한다.

※ 프로젝트의 버전 관리를 한다고 하는 수 많은 프로젝트들이, 응용 프로그램의 소스 코드만 저장소에 등록해놓고 있고, 정작 그 못지 않게 중요한 데이터베이스를 생성하는 방식을 보관하지 않고 있다.
그런 상황이라면, 데이터베이스를 개발하던 컴퓨터가 망가지는 등의 사고가 난다면 복구하기가 힘들거나, 심지어 거의 불가능해서 관련 기능을 처음부터 재구축하는 수 밖에 없을 것이다.

※ 한 번이라도 서비스한 이후에는, 데이터베이스의 구조가 변경될 경우에, 기존 데이터베이스를 날리고 새로 데이터베이스를 올리는 선택할 하기가 어렵다. (아마, 아예 그런 선택을 할 수도 없을 가능성이 높다..)
그래서 어느 정도 데이터베이스의 포맷이 안정화된 이후에는 데이터베이스를 일일이 이전(Migration)하는 선택을 하게 된다. 이러한 이전 과정에서 사용했던 SQL 구문과 명령문들을 버전과 실행 순서대로 보존해두면 향후 데이터베이스 관리에 큰 도움을 줄 수 있다.
데이터베이스가 어떤 버전이었는지 알 수만 있다면, 보존해둔 버전 / 실행 순서의 SQL 구문과 명령문을 순차적으로 수행해주는 것으로 구 버전 DB의 데이터들을 최신 구조의 DB로 무사히 옮겨올 수 있다.

◆ E-3-5. 마켓 버전 정책

- E-3-5-1.

E-4. 빌드 / 배포

◆ E-4-1. 빌드 환경

- E-4-1-1.

◆ E-4-2. 빌드 분류

- E-4-2-1.

◆ E-4-3. 빌드 프로세스

- E-4-3-1.

◆ E-4-4. 빌드 관리

- E-4-4-1.

◆ E-4-5. 빌드 공유 / 배포

- E-4-5-1.

E-5. 이슈 관리

◆ E-5-1. 이슈 관리 시스템

- **E-5-1-1.** 전사적으로 한 개만 사용한다.

- **E-5-1-2.** 레드마인(Redmine)을 사용한다.

- **E-5-1-3.** 프로젝트 소스 코드의 버전 관리 시스템과 연동해서 사용한다.

- **E-5-1-4.** 프로젝트 소스 코드를 버전 관리 시스템에 등록할 때는, 이슈 관리 시스템에 등록되어 있는 이슈의 식별 번호를 반드시 집어넣고, 그 내용을 설명하는 로그를 작성해야 한다.
  : 가능하다면, 이 과정이 지켜지지 않을 경우에는 버전 관리 시스템에 회부(Commit)가 불가능하도록 설정하면 좋다.

◆ E-5-2. 오류 / 버그 수집

- **E-5-2-1.** 일단 E-Mail을 기반으로 수집한다.

- **E-5-2-2.** 마켓 API나 대상 기기 OS의 기능을 이용해 버그를 수집할 수 있는지 여부 검토해봐야 함.

- **E-5-2-3.** 자동 충돌 보고를 활용할 수 있을지 확인해봐야 한다.
  : 이는 기기의 운영체제와 마켓 플랫폼의 정책에 따라 다를 수 있다.

◆ E-5-3. 사용 규칙

- **E-5-3-1.**

E-6. 프로젝트 보안

◆ E-6-1. 위험요소 분석

- **E-6-1-1.** 중요한 공용 컴퓨터들에 대해 물리적인 보안 장치가 없다.
  : 물리적 잠금 장치가 있는 방, 허가된 사람만 출입할 수 있는 보안 장치, 도난 방지 장치, 잠금 장치 등으로 보호하는 것이 좋다.

- **E-6-1-2.** 소스 코드 관리 시스템이 설치된 컴퓨터가 외부 인터넷 망과 직접 연결되어 있다.
  : 외부에서 해킹이나 악성코드를 통한 침투가 가능할 수 있다.
  내부 망에만 접속할 수 있게 한다면 상대적으로 안전하다.

- **E-6-1-3.** 중요한 공용 컴퓨터들에 대해 가용성 유지 장치가 충분하지 않다.
  : 디스크가 RAID 미러링이 되어 있지 않기 때문에, 디스크가 고장 날 경우, 저장소는 즉시 사용이 불가능해진다.
  공용 컴퓨터에 보조 전원 공급 장치가 없어서, 주 전원 공급 장치가 파손된 경우에 저장소 시스템이 즉각 정지할 것이다.

- **E-6-1-4.** 중요한 공용 컴퓨터들에 대한 백업 절차가 안전하지 않다.
  : 별도의 백업 컴퓨터와 디스크에 저장소와 이슈 관리 시스템, 자료 등을 백업하고는 있지만, 백업 컴퓨터가 제 3의 장소에 있지 않기 때문에, 화재 등의 천재지변에 취약하다.

◆ E-6-2. 물리적 보안

- **E-6-2-1.** 이에 대해서는 특별한 보안 조치 없음

◆ E-6-3. 소프트웨어 데이터 보안

- **E-6-3-1.** 빌드 유출 방지 대책
  : 빌드 지문?

- **E-6-3-2.** 각 빌드 파일의 식별 방식

:

E-7. 프로젝트 복구

◆ E-7-1. 백업 정책

- **E-7-1-1.** 저장소 백업 데이터가 존재하는 곳은 가급적 2군데 이상인 것이 좋고, 최소한 1곳은 개발팀 사무실 외부의 안전한 장소가 좋다.

- **E-7-1-2.** 백업 구성 절차는 사람이 구성하더라도, 정기적인 백업 수행 자체는 사람의 개입이 없이 자동으로 수행할 수 있게 구성해야 한다.

- **E-7-1-3.** 매일 최소한 1회 이상, 소스 코드 저장소 전체에 대한 백업을 수행해야 한다.

- **E-7-1-4.** 주 1회 이상, 이슈 관리 시스템 전체에 대한 백업을 수행해야 한다.

◆ E-7-2. 백업 수행 방식

- **E-7-2-1.** 백업 절차는 명령행(Command Line) 스크립트를 실행해서, 한 번의 명령행 스크립트 실행으로 백업 절차 전체를 완료할 수 있어야 한다.
  : 이렇게 하지 않으면 백업 절차를 자동화할 수 없다.

※ 백업 컴퓨터의 운영체제가 Windows인 경우에는 *.bat, 또는 *.com 파일을 이용한다.
MacOS X나 Linux 등의 Unix-like 운영체제인 경우에는 *.sh와 같은 쉘(Shell) 스크립트 파일을 이용한다.

- **E-7-2-2.** 개발팀이 한가한 시간에 자동으로 백업을 한다.
  : 예를 들자면 새벽 2 ~ 3시 경에 백업을 시작한다든가...

- **E-7-2-3.** 백업 컴퓨터의 운영체제 스케줄링 기능을 이용해서, 사람이 개입하지 않고도 자동으로 백업을 수행할 수 있게 한다.

※ 백업 컴퓨터의 운영체제가 Windows인 경우에는 작업 스케줄러(Task Scheduler)를 사용하고, 리눅스나 MacOS X 등 Unix-like 운영체제라면 cron 도구를 사용한다.

◆ E-7-3. 준수해야 할 사항들

- **E-7-3-1.** 백업 용 소스 코드 저장소는 오직 백업을 하는 용도로만 사용해야 하며, 개발용 저장소의 대용으로 쓰거나, 혹은 개발용 저장소와 혼용해서 써서는 안 된다.
  : 특히, 혼용해서 쓰지 않도록 주의해야 한다.

- **E-7-3-2.** 정기적으로 백업 저장소가 제대로 기능하는지 점검해야 한다.
  : 이미 사용 불가능할 정도로 망가져 있거나, 백업이 되고 있지 않은 상태인데도 이를 인식하지 못하고 있다가, 정작 복구가 필요할 때에 낭패를 겪을 수가 있다.

E-8. 프로젝트 위험 관리

◆ E-8-1. 위험 관리 계획

- E-8-1-1.

◆ E-8-2. 공유 정책

- E-8-2-1.

◆ E-8-3.

- E-8-3-1.

F. 품질보증 활동

F-1. 테스트

◆ F-1-1. 테스트 프로세스

- F-1-1-1.

◆ F-1-2. 단위 테스트

- F-1-2-1.

◆ F-1-3. 테스트의 자동화

- F-1-3-1.

F-2. 품질보증

◆ F-2-1. 일반 계획

- F-2-1-1.

◆ F-2-2. 수행방식

- F-2-2-1.

◆ F-2-3. 지역화 관련

- F-2-3-1.

F-3. 출시 / 배포

◆ F-3-1. 출시 / 배포 방식

- **F-3-1-1.** 출시 및 배포는 지정된 애플리케이션 온라인 스토어에서, 네트워크를 통한 내려 받기 형식으로만 가능하다.
  : 실행 파일에 대한 직접 복사 등의 방식은 허용하지 않는다.

- **F-3-1-2.** 출시 및 배포용 애플리케이션은, 이 프로젝트에서 지원하는 플랫폼의 기기 또는 플랫폼 제작사가 지원하는 방식을 통해서만 접근할 수 있다.

※ 우리는 어둠의 경로로 실행 파일을 배포하지 않습니다.(...)

- **F-3-1-3.** 각 플랫폼을 통해 내려 받을 수 있는 배포 파일은, **해당 플랫폼 및 기기에서만 실행할 수 있는 전용 형식**일 수 있다.

※ Android에서 iPhone 용 애플리케이션을 내려 받을 수 없다. 그 반대도 역시 마찬가지다.

- **F-3-1-4.** 실행파일 다운로드는 **3G, Wi-Fi, LTE(Long Term Evolution)**를 모두 이용할 수 있다.
  (만약, 다른 방식이 있을 경우 추가 바람.)

- **F-3-1-5.** 실행파일 외의 추가적인 리소스 데이터의 다운로드는 **Wi-Fi 또는 LTE에 연결된 상태에서** 다운로드 할 수 있다. (*유사한 다른 방식이 가능한 경우, 추가 바람.)
  : 만일, Wi-Fi나 LTE 등의 고속 네트워크에 연결되어 있지 않고, 대상 플랫폼에서 자동으로 사용자에게 이를 공지하는 기능을 제공하지 않을 경우, 사용자에게 이를 알리는 장치를 구현해야 한다.

◆ F-3-2. 배포할 크기 / 용량 제한

- **F-3-2-1.** 실행 파일 및 리소스 데이터의 용량 총합이 ~~50MB~~100MB 이내인 경우
  : 실행 파일 및 리소스 데이터를 하나의 패키지로 묶어서 한꺼번에 배포하는 것이 가능하다.

※ 이 경우는 Unity3D에서도 Resources 클래스를 이용할 수 있으므로 가장 간단한 방식이다.

가급적 여기에 맞춘다면, 현재(2012. 5)로서 가장 검증된 방식으로 리소스를 불러올 수 있다.
Resource 클래스에는 비동기적으로 리소스 자원을 불러오는 함수도 지원한다.

- **F-3-2-2.** 실행 파일 및 리소스 데이터의 용량 총합이 ~~50MB~~100MB를 초과하는 경우
: 리소스 데이터를 포함한 결과가 ~~50MB~~100MB 이내라면 가장 이상적이지만, 그렇지 못하게 될 경우에는 **실행 파일 및 최소의 필수 리소스 집합이 ~~50MB~~100MB 이내가 되도록 맞춘다.**
나머지 리소스 데이터는 별도의 방식을 이용해 게임 프로그램 내부에서 추가적으로 받아오게 해야 한다. (마치 온라인 게임을 실행시키기 전, 리소스 데이터의 업데이트를 받는 것과 같다.)

※ 이 방식은 난이도가 높으며, Unity3D에서 사용가능한지 여부가 개발팀에서 아직 명확하게 테스트되지 않았다. (2012. 5 기준)
Unity3D에서는 WWW 클래스를 이용해서 로딩해야 할 것으로 보이고, 이를 위해 추가적으로 리소스를 로딩하는 구조를 새롭게 설계해야 할 수가 있다.

- **F-3-2-3.** 리소스 데이터에 대한 배포 크기 제한
: 실행 파일 외, **추가적인 리소스 데이터들에 대한 특별한 총 크기 제한은 없다.**
다만, 대상 장치 중에서, 가장 저장 용량이 작은 장치의 최대 용량 이하로 제한한다고 할 수 있다.

- **F-3-2-4.** ~~50MB~~100MB를 초과하는 추가적인 데이터에 대해 다운로드할 경우, **데이터 송 / 수신에 따라 요금이 발생할 수도 있는 경우**, 이 사실에 대해 사용자에게 미리 알리고, 진행 여부를 선택할 수 있게 해야 한다.

※ 이 기능은 운영체제 및 응용 프로그램 마켓의 실행 단계에서 이미 지원하는 기능일 수 있다. 만일 그렇다면, 이 기능은 명세에는 있어도 직접 구현할 필요는 없다.

◆ F-3-3. 업데이트 방식

- **F-3-3-1.** 업데이트 방식의 사양은 배포 방식의 사양과 같다.