# **"던전 마스터" 기술 아키텍처 및 구현 방안 제안서**

## **Part I: foundational Architecture - "던전 마스터"의 초석**

본 파트에서는 프로젝트의 확장성, 유지보수성, 그리고 개발팀의 워크플로우에 가장 지대한 영향을 미칠 핵심 아키텍처 원칙들을 수립합니다. "던전 마스터"와 같이 다수의 시스템이 유기적으로 상호작용하는 게임에는 고도로 모듈화된 데이터 주도 접근 방식이 가장 이상적이며, 이를 중심으로 기술적 기반을 제안합니다.

### **Section 1: 데이터 중심의 세계: ScriptableObject를 활용한 확장성 및 협업 효율 증대**

#### **핵심 원칙: 데이터 컨테이너로서의 ScriptableObject**

"던전 마스터"의 구현에 있어 가장 근본적인 아키텍처 원칙은, 인스턴스에 종속되지 않는 거의 모든 게임 데이터를 ScriptableObject(이하 SO)에 저장하는 것입니다. 이는 기획서(GDD)에서 명확히 구분한 영구 데이터인 '베이스 카드'와 인스턴스 내 임시 데이터인 '인게임 유닛'의 개념과 완벽하게 부합합니다.1 SO는 씬(Scene)의 컴포넌트가 아닌, 프로젝트 내에 애셋(Asset) 형태로 존재하는 데이터 컨테이너를 생성하는 유니티의 네이티브 솔루션입니다.2

이러한 SO 중심 접근 방식은 다음과 같은 명백한 이점을 제공합니다:

* **기획자 자율성 확보:** 게임 기획자는 코드 작성이나 변경 없이 유니티 에디터 내에서 직접 게임 콘텐츠(신규 몬스터, 장비, 스킬 등)를 생성, 수정, 밸런싱할 수 있습니다. 이는 기획팀의 역량을 강화하고 프로그래머의 병목 현상을 줄여 팀 협업 효율을 극대화하는 핵심 요소입니다.5
* **디커플링(Decoupling) 및 모듈성:** 코드에 값을 하드코딩하거나 MonoBehaviour 스크립트 간에 직접적인 의존성을 생성하는 대신 SO 애셋을 참조함으로써, 각 컴포넌트를 쉽게 재사용하고 재구성할 수 있는 모듈식 시스템이 구축됩니다.6 이는 수많은 시스템이 상호작용하는 본 프로젝트의 복잡성을 관리하는 데 필수적입니다.
* **메모리 효율성:** SO는 플라이웨이트 패턴(Flyweight Pattern)을 따릅니다. 즉, 하나의 SO 애셋은 메모리에 한 번만 로드되며 다수의 게임 오브젝트가 이를 참조하는 방식으로 동작합니다. 만약 모든 몬스터 인스턴스가 각자의 기본 능력치 데이터를 복사하여 소유한다면 심각한 메모리 중복이 발생하지만, SO를 사용함으로써 이러한 문제를 원천적으로 방지할 수 있습니다. 이는 특히 리소스가 제한적인 모바일 플랫폼에서 매우 중요한 최적화 기법입니다.3

#### **구현 전략**

구체적인 구현을 위해, GDD에 명시된 주요 데이터 유형별로 ScriptableObject를 상속받는 C# 기반 클래스를 정의합니다. 예를 들어 UnitData(몬스터 및 마왕용), EquipmentData, SkillData, DungeonRoomData, ArtifactData, EnemyFactionData, RandomEventData 등이 이에 해당합니다.

UnitData SO를 예로 들면, 이 애셋은 GDD의 몬스터 스탯 테이블에 정의된 모든 '베이스 카드'의 고정 스탯(예: 카드 등급, 아머 타입, 속성 타입, 치명타율 등)을 포함하게 됩니다.1 더불어, '성장 가능 스탯'의 기본값(예:

BaseAttack, BaseHealth)과 스킬의 기본 정의 데이터도 이 SO에 저장됩니다. [CreateAssetMenu] 어트리뷰트를 활용하면 기획자가 프로젝트 뷰에서 마우스 우클릭만으로 새로운 몬스터나 장비 애셋을 손쉽게 생성할 수 있습니다.2

#### **데이터 관리 및 라이브 업데이트**

라이브 서비스를 목표로 하는 모바일 게임의 성공을 위해, 이러한 SO들을 관리하고 로드하는 방식으로 \*\*어드레서블 애셋 시스템(Addressable Asset System)\*\*의 도입을 강력히 권장합니다. 어드레서블 시스템을 사용하면 게임 밸런스를 조정하거나, 신규 몬스터를 추가하거나, 이벤트를 진행할 때마다 앱 스토어에 새로운 클라이언트 빌드를 제출할 필요 없이, 업데이트된 SO가 포함된 애셋 번들을 원격으로 전달하여 콘텐츠를 갱신할 수 있습니다.9 게임의

DataManager와 같은 중앙 관리 시스템은 SO에 대한 직접 참조를 갖는 대신, 게임 시작 시 또는 필요 시점에 어드레서블 주소를 통해 비동기적으로 SO를 로드하는 방식을 취하게 됩니다.

#### **SO-as-Blueprint 패러다임: 불변의 청사진과 가변의 인스턴스**

SO를 단순한 데이터 저장소로 보는 것을 넘어, \*\*런타임 객체를 위한 '불변의 청사진(Immutable Blueprint)'\*\*으로 취급하는 것이 이 아키텍처의 핵심입니다. GDD가 '베이스 카드'와 '인게임 유닛'을 명확히 구분한 것은 이 패러다임의 중요성을 이미 시사하고 있습니다.1

이 패러다임의 논리적 흐름은 다음과 같습니다:

1. GDD에 따르면 '베이스 카드'는 영구적이며 로비에서 강화되고, '인게임 유닛'은 일시적이며 던전 내에서 성장합니다.
2. SO는 프로젝트 애셋이므로 플레이 세션 간에 데이터가 유지됩니다. 런타임에 SO를 직접 수정하는 것은 매우 위험하며, 에디터를 재시작하기 전까지 모든 참조에 영향을 미칩니다.2 따라서 SO는 '영구 데이터' 역할에 완벽하게 부합합니다.
3. 반면, 일반 C# 클래스나 MonoBehaviour 인스턴스는 씬이 언로드되거나 명시적으로 파괴될 때 메모리에서 해제됩니다. 이는 '일시적 데이터' 역할에 적합합니다.
4. 따라서 최적의 아키텍처는 다음과 같습니다: 던전 플레이가 시작되면, InGameUnitManager는 플레이어의 저장된 데이터(특정 UnitData SO를 참조하는 정보)를 읽습니다. 배치할 몬스터에 대해, 제네릭 InGameUnit 프리팹을 인스턴스화합니다. 그리고 해당 인스턴스의 Initialize(UnitData blueprint) 메서드를 호출합니다.
5. 이 Initialize 메서드는 SO 청사진으로부터 기본 능력치를 복사하여 InGameUnit 인스턴스의 지역 변수에 저장합니다. 이 시점부터 인게임에서 발생하는 모든 변화(피해, 버프, 임시 레벨업 등)는 오직 이 **인스턴스**에만 영향을 미칩니다. 원본 SO 청사진은 다음 플레이를 위해 전혀 변경되지 않은 상태로 유지됩니다. 이 구조는 GDD의 설계를 완벽하게 모델링하며, 매우 안전하고 견고한 아키텍처 패턴입니다.4

### **Section 2: 디커플링의 힘: ScriptableObject 기반 이벤트 아키텍처 구현**

#### **핵심 원칙: 시스템 간의 느슨한 결합**

CombatSystem이 UISystem의 특정 메서드를 직접 호출하는 것과 같은 "스파게티 코드" 아키텍처를 피하기 위해, \*\*이벤트 기반 아키텍처(Event-Driven Architecture, EDA)\*\*를 도입합니다.11 이는 "발행자(Broadcaster)"와 "구독자(Listener)"가 중간 매개체인 "채널"을 통해 통신하는 시스템을 구축하여, 각 시스템이 서로에 대해 알 필요가 없는 느슨한 결합(Loose Coupling) 상태를 유지하도록 합니다.14

#### **구현: 이벤트 채널로서의 ScriptableObject**

Section 1에서 제안한 SO 아키텍처를 확장하여, "이벤트 채널(Event Channel)" 역할을 하는 SO 애셋을 생성합니다. 이는 유니티 환경에 특화된 강력한 패턴으로, 이벤트를 기획자가 직접 확인하고 관리할 수 있게 만들어 줍니다.15

EventChannel SO는 C#의 Action이나 UnityEvent 델리게이트를 포함하는 단순한 애셋입니다. 한 스크립트가 이 채널을 통해 이벤트를 "발행"하면, 다른 수많은 스크립트들이 서로의 존재를 모른 채 이 이벤트를 "구독"하여 반응할 수 있습니다.6

#### **"던전 마스터"에서의 실제 적용 예시**

OnEnemyKilled 이벤트를 예로 들어보겠습니다. CombatSystem에서 적의 체력이 0이 되었을 때, 이 시스템은 ScoreSystem에 점수를 추가하라고 직접 명령하거나 RewardSystem에 아이템을 드랍하라고 지시하지 않습니다. 대신, EnemyData SO를 페이로드(payload)로 담아 전역 OnEnemyKilled 이벤트를 발행합니다.

이 이벤트에 대한 구독자들은 다음과 같이 독립적으로 작동합니다:

* ScoreSystem은 OnEnemyKilled 이벤트를 구독하고 있다가, EnemyData를 받아 적의 속성에 기반하여 점수를 계산합니다.1
* RewardSystem 역시 동일한 이벤트를 구독하여, 전달받은 EnemyData를 기반으로 아이템 드랍 여부와 종류를 결정합니다.
* AchievementSystem은 이 이벤트를 통해 "특정 타입 몬스터 X마리 처치"와 같은 과제의 진행도를 갱신합니다.1

이러한 아키텍처 덕분에, 만약 "멋진 순간을 녹화하는 바이럴 리플레이" 기능 18을 새로 추가하더라도, 이 신규 시스템이

OnEnemyKilled 이벤트를 구독하기만 하면 됩니다. 기존의 CombatSystem 코드는 단 한 줄도 수정할 필요가 없습니다. 이것이 바로 확장성과 유지보수성의 정수입니다.11

#### **이벤트 시스템: 게임의 중추 신경계**

이벤트 시스템은 단순히 시스템을 분리하는 도구가 아니라, 게임 전체의 **중추 신경계**와 같습니다. 이는 정보의 흐름과 모든 게임 메커니즘 간의 인과 관계를 정의합니다. 핵심 이벤트를 설계하는 것은 사실상 게임의 전체 논리 흐름을 설계하는 것과 같습니다.

GDD는 복잡한 이벤트 연쇄를 묘사합니다: 플레이어의 '침략' 활동1은 '위협도'를 높이고1, 이는 다음 '웨이브'의 적을 강화하며1, 이는 '전투' 결과에 영향을 미칩니다. 전투 결과는 '점수'와 '보상'을 결정하고1, 이는 '영구 성장'을 가능하게 하며1, 이는 다시 플레이어의 다음 행동 선택에 영향을 줍니다.

이러한 흐름을 직접적인 함수 호출로 모델링하려 한다면, InvasionButton\_OnClick() 메서드가 ThreatManager, EnemySpawner, CombatManager 등을 모두 직접 참조해야 하는, 얽히고설킨 의존성 지옥에 빠지게 될 것입니다. 이는 수정과 디버깅을 극도로 어렵게 만듭니다.6

대신, 이벤트 연쇄로 이 흐름을 모델링합니다:

1. InvasionButton\_OnClick() → PlayerAction\_Invasion 이벤트 발행
2. ThreatManager가 구독 → PlayerAction\_Invasion 수신 → 위협도 갱신 → ThreatLevel\_Updated 이벤트 발행
3. EnemySpawner가 구독 → ThreatLevel\_Updated 수신 → 다음 웨이브 스폰 테이블 수정
4. ...이와 같은 방식으로 계속됩니다.

이 과정은 중요한 사실을 드러냅니다: 아키텍처의 견고함은 이벤트 채널이 얼마나 잘 정의되었는지에 정비례합니다. 따라서 본격적인 기능 구현에 앞서, 가장 먼저 수행해야 할 작업은 게임의 핵심 EventChannel SO들을 정의하는 것입니다. 이 이벤트 목록은 게임의 동적 행동에 대한 기술 명세서가 되어, 전체 시스템을 예측 가능하고 테스트하기 쉽게 만들어 줍니다.4

| 이벤트 채널 애셋명 (Event Channel Asset Name) | 페이로드 데이터 타입 (Payload Data Type) | 주요 발행자 (Primary Broadcaster) | 주요 구독자 (Primary Listener) | 핵심 역할 (Core Responsibility) |
| --- | --- | --- | --- | --- |
| OnEnemyKilledEvent | UnitData | CombatSystem | ScoreSystem, RewardSystem, AchievementSystem | 적 유닛이 처치되었음을 알리고, 후속 처리를 위해 해당 유닛의 원본 데이터를 전달합니다. |
| OnThreatLevelChangedEvent | int newThreatLevel | ThreatManager | EnemySpawner, UI\_ThreatMeter | '위협도' 수치가 변경되었음을 모든 관련 시스템에 통지하여 적 강화 및 UI 갱신을 유발합니다. |
| OnBaseCardUpgradedEvent | BaseCardData | UI\_UpgradePanel | PlayerDataManager, AudioSystem | 플레이어가 '베이스 카드'를 성공적으로 강화했음을 알리고, 데이터 저장 및 효과음 출력을 트리거합니다. |
| OnPlayerDamagedEvent | float damageAmount, Unit attacker | CombatSystem | ArtifactSystem, UI\_PlayerHealthBar | 플레이어 유닛이 피해를 입었을 때, 피해량과 공격자 정보를 전달하여 특정 유물 효과나 UI 반응을 처리합니다. |

### **Section 3: 프로젝트 구조와 애자일 워크플로우**

#### **핵심 원칙: 일관성과 효율성**

깨끗하고 일관된 프로젝트 구조와 개발 프로세스는 팀의 효율성과 프로젝트의 장기적인 건전성을 위해 필수적입니다. 본 섹션에서는 구체적인 폴더 구조와 애자일(Agile) 개발 방법론을 제안합니다.

#### **Unity 프로젝트 구성**

* **폴더 구조:** 애셋을 쉽게 찾을 수 있도록 엄격한 폴더 구조를 강제해야 합니다. 특히, 에셋 스토어 등에서 가져온 서드파티(third-party) 애셋과 프로젝트 자체 애셋을 분리하여 업데이트 시 발생할 수 있는 충돌을 방지하는 것이 매우 중요합니다.20
  + **제안 구조:**
    - /\_Project/: 모든 게임 관련 애셋을 위한 루트 폴더
      * /Scripts/: C# 코드. 네임스페이스/기능별로 구성 (예: Core, Combat, UI)
      * /ScriptableObjects/: 모든 SO 애셋. 타입별 하위 폴더 구성 (예: Units, Equipment, Events)
      * /Prefabs/: 재사용 가능한 게임 오브젝트
      * /Art/, /Audio/: 모든 시각 및 사운드 애셋
    - /ThirdParty/: 모든 임포트된 서드파티 패키지
* **명명 규칙:** 일관성을 유지하기 위해 파일, 애셋, 클래스에 대한 명확한 명명 규칙을 수립하고 문서화해야 합니다 (예: UnitData\_Goblin, Event\_OnEnemyKilled, Stat\_AttackSpeed).20

#### **애자일 개발 프로세스**

* **방법론:** \*\*스크럼(Scrum)\*\*과 같은 애자일 프레임워크를 채택합니다.21 이는 작업을 시간 제한이 있는 "스프린트(Sprint)"(예: 2주) 단위로 나누고, 우선순위가 지정된 "제품 백로그(Product Backlog)"를 유지하며, 정기적인 회의(일일 스탠드업, 스프린트 리뷰, 회고)를 진행하는 방식입니다. 이러한 반복적 접근법은 요구사항과 '재미' 요소가 계속해서 진화하는 게임 개발에 이상적입니다.23
* **우선순위 설정:** 백로그의 우선순위를 정하기 위해 **MoSCoW**(Must-Have, Should-Have, Could-Have, Won't-Have) 기법을 사용합니다.24 "던전 마스터"의 경우, "Must-Have"는 핵심 루프인 [전투 → 보상 → 베이스 카드 강화]입니다.1 '비동기 PvP'와 같은 기능은 1 후반 단계의 "Could-Have"가 될 것입니다.

#### **지속적 통합/지속적 배포 (CI/CD)**

프로젝트 초기부터 CI/CD 파이프라인을 구현해야 합니다. 이는 새로운 코드가 버전 관리 시스템에 커밋될 때마다 자동으로 게임을 빌드하고 테스트하는 프로세스를 의미합니다.

* **도구:** GameCI for GitLab/GitHub과 같은 솔루션을 사용합니다.25 이들은 유니티용으로 사전 구성된 도커(Docker) 이미지를 제공하여 설정 과정을 크게 단순화합니다.
* **이점:** CI/CD는 통합 오류를 조기에 발견하고, 게임이 항상 빌드 가능한 상태임을 보장하며, 테스트용 빌드 생성 과정을 자동화하여 개발 시간을 크게 절약해 줍니다.24

#### **아키텍처가 주도하는 애자일**

본 보고서에서 제안하는 SO 및 이벤트 기반 아키텍처는 애자일 워크플로우를 직접적으로 지원하고 강화합니다. 모듈성은 단순히 기술적 이점을 넘어, 프로젝트 관리의 강력한 무기가 됩니다.

애자일은 큰 문제를 작고, 독립적이며, 테스트 가능한 사용자 스토리로 나누는 것을 요구합니다.22 단일체(monolithic) 구조의 강하게 결합된 아키텍처에서는 이것이 어렵습니다. 한 시스템의 변경이 예측 불가능한 파급 효과를 일으켜, 작업량을 추정하거나 독립적으로 테스트하기 어렵게 만듭니다.

하지만 제안된 아키텍처는 코드베이스에 자연스러운 경계선을 만듭니다. CombatSystem과 UISystem이 이벤트 채널에 의해 분리되어 있기 때문에, 한 개발자는 "전투" 관련 스토리를, 다른 개발자는 "UI" 관련 스토리를 병합 충돌의 낮은 위험 속에서 동시에 작업할 수 있습니다.5

더 나아가, SO의 데이터 주도적 특성 덕분에 기획자와 프로그래머가 병렬로 작업할 수 있습니다. 프로그래머가 StatSystem을 구현하는 동안, 기획자는 수십 개의 UnitData SO를 생성하며 첫 스프린트의 몬스터 라인업을 정의할 수 있습니다.5 따라서 모듈성과 디커플링을 우선시하는 아키텍처 결정은 미래의 유지보수성을 위한 기술적 선택일 뿐만 아니라, 일상적인 애자일 개발 프로세스의 효율성과 병렬성을 직접적으로 향상시키는 전략적 선택입니다. 즉, 아키텍처가 워크플로우를 지원하는 구조가 됩니다.

## **Part II: System-by-System Implementation Deep Dive**

본 파트에서는 앞서 수립한 고수준 아키텍처를 GDD에 정의된 핵심 시스템들의 구체적인 구현 계획으로 전환합니다.

### **Section 4: 통합 스탯 시스템: 모든 수정자를 위한 유연한 기반**

#### **문제 정의**

GDD는 다수의 변수(스킬계수, 보호율, 속성상성배율 등)를 포함하는 복잡한 데미지 공식을 정의하고 있습니다.1 또한, 능력치 변경의 출처가 매우 다양합니다: 베이스 카드 강화 1, 마왕 영구 장비 1, 인스턴스 임시 장비 1, 버프/디버프, 유물 1, 마왕 권능 1 등. 단순히

float attackPower;와 같은 변수로는 이 복잡성을 감당할 수 없습니다.

#### **해결책: 스탯 수정자 패턴 (Stat Modifier Pattern)**

이 문제에 대한 해결책으로, RPG 게임에서 복잡한 능력치 계산을 처리하기 위해 잘 정립된 솔루션인 "스탯 수정자 패턴"에 기반한 견고하고 재사용 가능한 스탯 시스템을 구현합니다.26

#### **구현 세부사항**

* **CharacterStat 클래스:** 공격력, 방어력 등 개별 능력치를 나타내는 클래스입니다. 이 클래스는 능력치의 BaseValue(기본값)와 StatModifier 객체들의 리스트를 가집니다. 외부에 공개되는 Value 프로퍼티는 접근 시점에 모든 수정자를 계산하여 최종 수정된 값을 즉시 반환합니다.
* **StatModifier 클래스/구조체:** Value(값), Type(타입을 나타내는 열거형), 그리고 선택적으로 Source(이 수정자를 적용한 아이템이나 버프 등 출처)를 포함하는 간단한 데이터 객체입니다.26
* **ModifierType 열거형(Enum):** 이 열거형은 계산 로직의 핵심입니다. 수정자가 적용되는 방식을 정의하며, 주로 Flat(고정값 합산), PercentAdd(합연산 퍼센트), PercentMult(곱연산 퍼센트)의 세 가지 타입을 가집니다.26
* **계산 로직과 순서의 중요성:** CharacterStat.Value의 getter는 예측 가능한 결과를 보장하기 위해 엄격하고 정의된 순서에 따라 계산을 수행해야 합니다. 이는 게임 밸런싱에 있어 **절대적으로 중요합니다**.26 표준적이고 검증된 계산 순서는 다음과 같습니다:
  1. finalValue = BaseValue로 시작합니다.
  2. 모든 Flat 타입 수정자들을 합산하여 finalValue에 더합니다.
  3. 모든 PercentAdd 타입 수정자들을 합산합니다. 2단계에서 계산된 값에 (1 + sumOfPercentAdd)를 곱합니다.
  4. 각각의 PercentMult 타입 수정자에 대해, 현재 finalValue에 (1 + percentMult)를 순차적으로 곱합니다.

#### **조건부 수정자 (Conditional Modifiers)**

GDD는 조건부 효과를 암시합니다 (예: '화염 제단' 방이 화염 속성 몬스터에게 더 큰 버프를 제공 1). 이는

StatModifier 개념을 확장하여 처리할 수 있습니다. ConditionalStatModifier 클래스를 만들어, 수정자가 적용되기 전에 체크할 조건(예: Target.AttributeType == Fire)을 포함시키는 방법이 있습니다. 하지만 더 깔끔한 대안은, 버프를 적용하는 시스템(예: DungeonRoom 로직)이 직접 조건을 확인하고, 조건이 충족될 때만 해당 유닛의 CharacterStat 리스트에 일반 StatModifier를 추가하는 방식입니다.31

#### **버프와 디버프: 수정자 컨테이너**

"버프"나 "디버프"는 별개의 특별한 시스템이 아닙니다. 이것들은 본질적으로 StatModifier들의 리스트와 지속 시간(duration)을 담고 있는 임시 객체일 뿐입니다.33 유닛에게 버프가 적용되면, 그 안에 담긴

StatModifier들이 대상 유닛의 해당 CharacterStat 리스트에 추가됩니다. 버프의 지속 시간이 만료되면, Source 속성을 이용해 어떤 버프가 적용한 수정자인지 식별하여 해당 수정자들을 리스트에서 제거합니다.26 이 방식은 버프/디버프 시스템을 통합 스탯 시스템에 우아하게 통합시킵니다.

| 계산 순서 | 수정자 타입 (Modifier Type) | 계산 공식 (Formula) | GDD 내 예시 출처 (Example Source in GDD) |
| --- | --- | --- | --- |
| 1 | 기본값 (Base Value) | $final = base$ | 몬스터 베이스 카드의 기본 스탯 |
| 2 | 고정값 합산 (Flat Additive) | $final = final + \sum(flat\\_modifiers)$ | 마왕 영구 장비의 고정 공격력 +10 |
| 3 | 합연산 퍼센트 (Additive Percent) | $final = final \times (1 + \sum(percent\\_add\\_modifiers))$ | 마왕 권능의 '모든 유닛 공격력 +5%' |
| 4 | 곱연산 퍼센트 (Multiplicative Percent) | $final = final \times \prod(1 + percent\\_mult\\_modifiers)$ | 유물 '화염 심장'의 '화염 속성 몬스터 공격력 +30%' |

### **Section 5: 핵심 데이터 흐름: '베이스 카드'에서 전장의 유닛으로**

#### **핵심 원칙: 청사진과 인스턴스의 생명주기**

이 섹션은 Section 1에서 제시한 "SO-as-Blueprint" 패러다임의 실제 적용 과정을 상세히 설명합니다. 영구적인 메타 데이터가 어떻게 전장의 일시적이고 상태를 가진 객체로 변환되는지의 생명주기를 정의합니다.

#### **데이터 흐름도**

1. **메타 게임 (로비):** 플레이어는 UI를 통해 '주화'를 소모하여 '베이스 카드'를 강화합니다.1 이 행동은 플레이어의 영구 저장 데이터(예: JSON 파일)를 수정합니다. 이 데이터는  
   베이스 카드의 ID와 현재 강화 레벨(예: GrowthRateLevel, SkillLevel)을 저장합니다. 이 과정에서 원본 Base Card SO 애셋 자체는 **절대 수정되지 않습니다.**
2. **인스턴스 시작 (던전 진입):** GameManager는 플레이어의 저장 데이터를 로드합니다. UnitFactory 또는 Spawner는 인게임 유닛 생성을 준비합니다.
3. **인스턴스화:** 특정 몬스터를 생성하기 위해 UnitFactory는 다음을 수행합니다:
   * 어드레서블을 통해 해당 몬스터의 UnitData SO를 로드합니다.9
   * 제네릭 InGameUnit 프리팹을 인스턴스화합니다.
   * 공격력, 방어력 등을 위한 CharacterStat 클래스의 새 인스턴스들을 생성합니다.
4. **초기화:** InGameUnit의 Initialize 메서드가 호출됩니다.
   * 이 메서드는 UnitData SO와 해당 카드에 대한 플레이어의 저장된 강화 레벨을 인자로 받습니다.
   * SO의 기본 수치와 저장된 강화 레벨을 조합하여 각 능력치의 실제 BaseValue를 계산합니다. (예: BaseAttack = SO의 기본 공격력 + SO의 성장률 \* 저장 데이터의 성장률 레벨)
   * 이렇게 계산된 BaseValue가 CharacterStat 인스턴스에 설정됩니다.
5. **인게임 생명주기:** 이 시점부터 모든 변화(받는 피해, 유물로부터의 버프 1, 임시 장비 1 등)는  
   InGameUnit 인스턴스에 있는 CharacterStat 리스트에 수정자로 적용됩니다.
6. **인스턴스 종료:** 던전 플레이가 끝나면 모든 InGameUnit 인스턴스는 파괴됩니다. 플레이어의 영구 저장 데이터와 원본 UnitData SO는 전혀 영향을 받지 않은 채로 남습니다.

#### **데이터 주도 설계가 진정한 로그라이크의 다양성을 만든다**

불변의 청사진(SO)과 가변의 인스턴스(클래스 객체)를 이처럼 엄격하게 분리하는 구조는, '적 세력' 1이나 '유물' 1과 같은 다른 로그라이크 시스템들이 깔끔하게 작동하고 예측 불가능한 재미(Emergent Gameplay)를 만들어내는 기술적 기반이 됩니다.

GDD는 해당 플레이에만 적용되는 광범위한 변화를 요구합니다. 예를 들어, '적 세력' 시스템은 이번 판의 모든 '성기사단' 적에게 "+20% 방어력" 버프를 부여할 수 있습니다. 어떤 '유물'은 모든 '언데드' 아군에게 '생명력 흡수' 능력을 부여할 수 있습니다.

만약 SO를 직접 수정하는 방식이었다면, 이는 재앙과도 같았을 것입니다. 버프를 적용한 뒤, 플레이가 끝날 때 완벽하게 제거하는 것을 보장해야 하는 복잡한 관리가 필요합니다.

하지만 청사진/인스턴스 모델에서는 구현이 매우 우아해집니다. EnemyFactionSystem은 '성기사단' 유닛이 **스폰될 때마다** 임시 StatModifier(+20% 방어력)를 적용하기만 하면 됩니다. ArtifactSystem은 모든 아군 유닛에게 조건부 로직(타입이 언데드일 경우 생명력 흡수 효과 추가)을 적용할 수 있습니다.

이는 동일한 '고블린' UnitData SO가 해당 판의 컨텍스트(활성화된 적 세력, 플레이어가 획득한 유물, 무작위 이벤트 등)에 따라 매우 다른 성능의 인게임 유닛으로 나타날 수 있음을 의미합니다. 이것이 바로 로그라이크 장르의 핵심인 전략적 다양성과 리플레이 가치를 만들어내는 기술적 토대입니다.35 아키텍처가 장르의 핵심 기둥을 직접적으로 뒷받침하는 것입니다.

### **Section 6: 전투 루프 구현**

이 섹션은 GDD의 데미지 공식 1을 구체적인 함수로 변환하는 방법을 다룹니다. 공식에 사용되는 각 변수(예:

공격력, 방어력, 피해감소율)가 공격자와 방어자의 각 CharacterStat 객체의 .Value 프로퍼티에 접근하여 어떻게 검색되는지를 보여줍니다. 특히, OnAttack, OnDamageDealt, OnHeal, OnDodge와 같은 전투 관련 이벤트에 대해 이벤트 시스템을 적극적으로 활용하는 것을 강조합니다. 예를 들어, 피해를 입을 때마다 스킬 쿨타임이 감소하는 유물 '피의 계약서' 1는, 플레이어 유닛에 대한

OnDamageDealt 이벤트를 구독하는 것만으로 간단하게 구현할 수 있습니다. 이 방식은 전투 로직과 유물 시스템 간의 직접적인 의존성을 제거하여 유연성을 극대화합니다.

### **Section 7: 동적 로그라이크 시스템 관리**

이 섹션에서는 '위협도' 1, '랜덤 이벤트' 1, '적 세력' 1 시스템의 구현을 상세히 설명합니다.

* **위협도 시스템:** 전역 FloatVariable SO를 사용하여 위협도 수치를 관리합니다.4 플레이어의 특정 행동(예: '침략')은 이벤트를 발생시키고,  
  ThreatManager가 이 이벤트를 수신하여 위협도 SO의 값을 갱신합니다. EnemySpawner는 이 SO의 값을 읽어 새로 스폰되는 적들에게 스탯 수정자를 적용합니다. 이때 적용되는 강화 공식(예: 위협도 1당 스탯 0.5% 증가) 자체도 별도의 BalanceData SO에 저장하여 기획자가 쉽게 수치를 조정할 수 있도록 합니다.
* **랜덤 이벤트:** 각 이벤트는 RandomEventData SO로 정의됩니다. 이 SO는 이벤트의 설명, 선택지, 그리고 가장 중요하게는 플레이어의 선택에 따라 발행해야 할 EventChannel SO에 대한 참조를 포함합니다. 이를 통해 이벤트 시스템 전체가 데이터 기반으로 작동하며, 프로그래머의 개입 없이 새로운 이벤트를 쉽게 추가할 수 있습니다.
* **적 세력:** 각 세력은 EnemyFactionData SO로 정의됩니다. 이 SO는 해당 세력에 속한 유닛들의 UnitData SO 리스트와 그들에게 공통적으로 적용될 StatModifier 리스트를 가집니다. 웨이브 시작 시 EnemySpawner는 무작위로 하나의 세력을 선택하고, 이 SO에 정의된 데이터를 사용하여 해당 웨이브의 적들을 구성하고 강화합니다.

### **Section 8: 영구 성장 시스템 아키텍처**

이 섹션은 메타 게임 루프의 핵심인 영구 성장 시스템의 아키텍처를 다룹니다.

* **마왕 권능 (마신전):** 이 시스템 1은 스킬 트리의 각 노드를 나타내는  
  GvonneungNodeData SO들의 집합으로 구현됩니다. 플레이어가 '권능 포인트'를 사용해 노드를 해금하면, GvonneungManager는 해당 노드와 연결된 영구적인 StatModifier를 전역 PlayerProfile 객체에 적용하고 저장합니다. 새로운 던전 플레이가 시작될 때, 이 영구 수정자들은 가장 먼저 모든 관련 유닛에게 적용되어, 인게임 효과가 더해지기 전의 기초 능력치 층을 형성합니다.
* **도전 과제:** AchievementSystem은 OnEnemyKilled, OnDungeonCleared 등 다양한 게임 이벤트를 구독하는 리스너들의 집합으로 작동합니다. 특정 과제의 조건이 충족되면, 이 시스템은 OnRewardGranted 이벤트를 발행하여 보상(재화, '권능 포인트' 1 등)을 지급합니다. 이 구조는 과제 시스템이 다른 시스템에 전혀 영향을 주지 않으면서 독립적으로 작동하도록 보장합니다.

## **Part III: Live Operations, Expansion, and Optimization**

이 파트에서는 게임 출시 이후의 라이브 운영, 향후 콘텐츠 확장, 그리고 모바일 환경에서의 성능 최적화 방안을 다룹니다.

### **Section 9: 밸런싱 작업: 개발과 라이브 서비스를 위한 데이터 관리**

#### **핵심 원칙: 단계별 데이터 관리 전략**

게임에 존재하는 수백 개의 변수를 효과적으로 밸런싱하는 것은 매우 중요한 과제입니다. 이를 위해 개발 단계와 라이브 서비스 단계를 구분하는 2단계 전략을 제안합니다. 이는 빠른 프로토타이핑과 안정적인 라이브 운영이라는 두 마리 토끼를 모두 잡기 위함입니다.37

* 1단계 (개발 단계): Google Sheets를 이용한 신속한 이터레이션  
  개발 초기 및 프로토타이핑 단계에서는 기획자가 가장 익숙하고 빠르게 데이터를 수정할 수 있는 Google Sheets를 활용합니다.39 에디터 전용 툴을 개발하여, Google Sheets API 40를 통해 기획자가 관리하는 스프레드시트의 데이터를 다운로드하고, 이를 프로젝트 내의 로컬  
  ScriptableObject 애셋에 자동으로 채워 넣습니다. 실제 게임은 이 로컬 SO를 읽어 실행되며, 런타임에 Google Sheets에 직접 연결하지는 않습니다. 이 방식은 기획자가 프로그래머의 도움 없이 실시간으로 밸런스를 수정하고 즉시 테스트하는, 매우 빠른 이터레이션 사이클을 가능하게 합니다.38
* 2단계 (라이브 서비스 단계): Addressables를 통한 안전하고 확장 가능한 업데이트  
  게임이 정식으로 출시된 후에는 보안과 성능, 확장성을 고려해야 합니다. 라이브 환경에서는 Google Sheets 직접 연동 방식의 잠재적 위험(API 호출 제한, 보안 키 노출 등)을 피해야 합니다.37 대신, 모든 밸런스 데이터가 담긴 SO들을 어드레서블 애셋 시스템 9을 사용하여 애셋 번들로 빌드합니다. 밸런스 변경이 필요할 경우, 새로운 애셋 번들을 생성하여 CDN(Content Delivery Network, 예: Unity Cloud Content Delivery 9)에 업로드합니다. 게임 클라이언트는 시작 시 이 번들을 다운로드하여 최신 데이터로 갱신합니다. 이 방식은 라이브 모바일 게임의 표준적인 업데이트 방식으로, 안전하고 확장 가능하며 사용자에게 앱 전체 업데이트를 강요하지 않습니다.

| 데이터 관리 전략 | 기획자 이터레이션 용이성 | 런타임 성능 | 보안 | 라이브 운영 확장성 | 구현 복잡도 |
| --- | --- | --- | --- | --- | --- |
| **로컬 SO (기본)** | 낮음 (코드 수정 필요) | 높음 | 높음 | 낮음 (클라이언트 업데이트 필요) | 낮음 |
| **Google Sheets (라이브 연동)** | 매우 높음 | 낮음 (네트워크 지연) | 매우 낮음 (API 키 노출 위험) | 낮음 (확장성 문제) | 중간 |
| **Google Sheets (에디터 전용) + Addressables (라이브)** | **매우 높음 (개발 시)** | **높음** | **높음** | **매우 높음** | **높음 (초기 설정)** |

### **Section 10: 비동기 PvP를 위한 청사진**

이 섹션에서는 GDD에서 제안된 비동기(Asynchronous) PvP 시스템의 기술적 설계를 제공합니다.1 이는 실시간 통신 없이 다른 유저의 데이터를 기반으로 한 AI와 대결하는 방식입니다.42

* **방어 데이터 구조:** 플레이어가 자신의 방어 진형을 설정하면, 서버는 해당 방어팀의 스냅샷을 저장합니다. 이 스냅샷에는 배치된 몬스터들의 '베이스 카드' ID, 해당 카드의 영구 성장 레벨, 그리고 던전 방의 배치 정보 등이 포함됩니다.
* **매치메이킹:** 서버는 플레이어의 랭킹 점수나 전투력과 같은 지표를 기반으로 비슷한 수준의 상대방을 매칭하는 간단한 알고리즘을 사용합니다.42
* **전투 시뮬레이션:** 공격하는 플레이어의 클라이언트는 방어하는 플레이어의 방어 데이터를 서버로부터 다운로드합니다. 이후 모든 전투 과정은 공격자 측의 디바이스에서 로컬로 시뮬레이션됩니다. 방어측 유닛들은 사전에 정의된 AI 로직에 따라 행동합니다. 이 방식은 실시간 네트워크 동기화의 복잡성을 완전히 회피할 수 있습니다.
* **결과 보고 및 검증:** 시뮬레이션이 종료되면, 공격자 클라이언트는 승/패 결과를 서버에 보고합니다. 서버는 이 결과를 바탕으로 랭킹 점수를 갱신하고 보상을 분배합니다. 이때, 비정상적인 전투 시간이나 데미지 수치 등을 검증하는 기본적인 안티-치트 로직이 반드시 포함되어야 합니다.

### **Section 11: 전략적 깊이를 위한 UI/UX: 복잡성의 시각화**

이 섹션은 게임의 깊이 있는 시스템들을 플레이어에게 정보 과부하 없이 효과적으로 전달하는 중대한 과제를 다룹니다.43

* **핵심 원칙:** 명확성과 맥락에 맞는 정보 제공을 최우선으로 합니다. 툴팁, 아이콘, 그리고 깨끗한 시각적 계층 구조를 적극적으로 활용해야 합니다.45
* **시너지 시각화:** 단순히 텍스트로 시너지 효과를 나열하는 대신, Teamfight Tactics와 같이 전장에 배치된 유닛들 사이에 시각적인 연결선이나 그룹 하이라이트를 표시하여 직관적으로 시너지를 인지할 수 있도록 해야 합니다.46
* **능력치 상세 정보:** 플레이어가 유닛을 선택하여 상세 정보를 볼 때, 최종 적용된 능력치 값만 보여주는 것이 아니라, 해당 값을 탭하면 [기본값 + 장비 + 버프]와 같이 세부 내역을 볼 수 있는 기능을 제공해야 합니다. 이 정보는 CharacterStat 클래스와 그 안의 StatModifier 리스트에서 직접 데이터를 가져와 표시합니다.
* **전투 로그:** 전략 게임에서 필수적인 기능입니다. 스크롤 가능한 간단한 텍스트 로그를 통해, 누가 누구를 공격하여 얼마의 피해를 입혔는지, 어떤 버프가 적용/만료되었는지, 적의 어떤 의도(intent)가 발동되었는지 등의 핵심 정보를 알려주어야 합니다. 이는 플레이어가 전투의 흐름을 이해하고 게임의 메커니즘을 학습하는 데 결정적인 역할을 합니다.48

### **Section 12: 필수 모바일 성능 및 최적화**

모바일 플랫폼을 타겟으로 하는 만큼, 타협할 수 없는 필수 최적화 기법들의 체크리스트를 제공합니다.

* **렌더링 최적화:**
  + 복잡한 구조의 던전에서는 \*\*오클루전 컬링(Occlusion Culling)\*\*을 사용하여 카메라에 보이지 않는 오브젝트가 렌더링되지 않도록 합니다.50
  + 폴리곤 수가 많은 몬스터 모델에는 \*\*LOD(Level of Detail)\*\*를 적용하여 거리에 따라 단순한 모델로 교체되도록 합니다.51
  + 가능한 경우, 여러 개의 메시를 하나로 결합하여 렌더링에 필요한 **드로우 콜(Draw Call)** 횟수를 줄여야 합니다.51
* **CPU 최적화:**
  + Update()나 FixedUpdate() 루프 안에서 복잡한 로직을 매 프레임 실행하는 것을 피해야 합니다. 본 제안서의 이벤트 기반 아키텍처는 이벤트가 발생했을 때만 로직이 실행되므로 이 문제 해결에 큰 도움이 됩니다.
  + GetComponent<T>()와 같은 비용이 높은 함수 호출은 Awake()나 Start()에서 한 번만 호출하여 결과를 캐싱해 두어야 합니다.52
  + 다수의 유닛에게 적용되는 독(poison)과 같은 주기적인 효과는 각 유닛이 자신의 Update() 타이머를 갖게 하는 대신, 단일 매니저가 모든 유닛의 효과를 시간차를 두고(Time Slicing) 갱신하는 방식으로 구현하여 CPU 스파이크를 방지해야 합니다.52
* **메모리 최적화:**
  + ScriptableObject와 플라이웨이트 패턴을 사용하는 것 자체가 주요 메모리 최적화 기법입니다.3
  + 어드레서블 시스템을 사용하여 애셋을 동적으로 로드하고 언로드함으로써, 현재 씬에 필요하지 않은 애셋이 메모리를 점유하지 않도록 관리해야 합니다.9 이는 메모리가 제한적인 모바일 환경에서 매우 중요합니다.

## **Part IV: 결론 및 전략적 권장사항**

### **Section 13: 우선순위가 적용된 구현 로드맵**

본 보고서의 모든 제안을 종합하여, 명확하고 실행 가능한 단계별 계획을 제시합니다. 모든 기능을 한 번에 구현하기보다는, 안정적인 기반을 우선적으로 구축하는 단계적 접근 방식을 강력히 권장합니다.

* **1단계: 핵심 기반 구축 (The Core Foundation)**
  1. **아키텍처 구현:** SO 기반 클래스, 이벤트 채널 시스템, 통합 스탯 시스템 등 본 제안서 Part I에서 설명한 foundational architecture를 최우선으로 구현합니다.
  2. **핵심 게임 루프 완성:** 단일 몬스터 타입, 단일 플레이어 유닛, 기본적인 전투 로직, 그리고 '베이스 카드' 강화 화면을 구현하여 [전투 → 보상 → 강화 → 재도전]의 핵심 루프를 완성합니다.1
  3. 이 단계가 완료되면, 게임은 비록 단순하지만 완전히 플레이 가능한 상태가 되어야 합니다. 이는 핵심 아키텍처의 유효성을 검증하는 중요한 마일스톤입니다.
* **2단계: 시스템 및 콘텐츠 확장 (System and Content Expansion)**
  1. **주요 시스템 개발:** 1단계에서 구축한 기반 위에 '위협도', '유물', '적 세력', '마왕 권능' 등 GDD의 핵심 시스템들을 개발합니다.
  2. **콘텐츠 제작 병행:** 프로그래머가 시스템을 개발하는 동안, 기획자들은 수십 종의 UnitData, ArtifactData SO 등을 생성하며 게임의 콘텐츠를 채워나갑니다. 데이터 주도 아키텍처 덕분에 이러한 병렬 작업이 가능합니다.
* **3단계: 폴리싱 및 메타 기능 (Polish and Meta-Features)**
  1. **부가 시스템 구현:** 도전 과제, 고급 UI/UX 기능(전투 로그, 상세 정보 창 등), 소셜 기능 등 부가적인 시스템들을 구현합니다.
  2. **장기 확장 콘텐츠 개발:** 비동기 PvP와 같이 장기적인 관점의 확장 콘텐츠 개발에 착수합니다.

이러한 로드맵은 가장 중요하고 리스크가 큰 아키텍처 작업을 초기에 완료하여 모든 후속 개발을 위한 견고한 토대를 마련하고, 반복적인 가치 전달을 중시하는 애자일 원칙에도 부합하는 가장 안정적인 개발 경로입니다.23

#### 참고 자료

1. 던전 마스터 (가제) - 통합 게임 기획서 v3.0.docx
2. 유니티 스크립터블 오브젝트( ScriptableObject ) - Notyu - 티스토리, 8월 3, 2025에 액세스, <https://notyu.tistory.com/65>
3. | 니앙팽이 - 스크립터블 오브젝트 (게임 개발 : Unity) | ScriptableObject — 니앙팽이 블로그, 8월 3, 2025에 액세스, <https://felipuss.tistory.com/entry/%EB%8B%88%EC%95%99%ED%8C%BD%EC%9D%B4-%EC%8A%A4%ED%81%AC%EB%A6%BD%ED%84%B0%EB%B8%94-%EC%98%A4%EB%B8%8C%EC%A0%9D%ED%8A%B8-%EA%B2%8C%EC%9E%84-%EA%B0%9C%EB%B0%9C-Unity-ScriptableObject>
4. Architect your code for efficient changes and debugging with ... - Unity, 8월 3, 2025에 액세스, <https://unity.com/how-to/architect-game-code-scriptable-objects>
5. 협업과 코딩 측면에서 유용한 스크립터블 오브젝트 활용법 6가지 - Unity, 8월 3, 2025에 액세스, <https://unity.com/kr/blog/engine-platform/6-ways-scriptableobjects-can-benefit-your-team-and-your-code>
6. 스크립터블 오브젝트로 게임을 설계하는 세 가지 방법 - Unity, 8월 3, 2025에 액세스, <https://unity.com/kr/how-to/architect-game-code-scriptable-objects>
7. How to architect code as your project scales | Avoiding technical debt - Unity, 8월 3, 2025에 액세스, <https://unity.com/how-to/how-architect-code-your-project-scales>
8. Explore Unity's best practices, 8월 3, 2025에 액세스, <https://unity.com/how-to>
9. Simplify your content management with Addressables | Video ... - Unity, 8월 3, 2025에 액세스, <https://unity.com/how-to/simplify-your-content-management-addressables>
10. Level Up Your Asset Management With Unity Addressables - Diversion, 8월 3, 2025에 액세스, <https://www.diversion.dev/blog/level-up-your-asset-management-with-unity-addressables>
11. 이벤트 기반 아키텍처란 무엇인가요? - Google Cloud, 8월 3, 2025에 액세스, <https://cloud.google.com/discover/what-is-event-driven-architecture?hl=ko>
12. 이벤트 기반 아키텍처란 무엇일까요? - Red Hat, 8월 3, 2025에 액세스, <https://www.redhat.com/ko/topics/integration/what-is-event-driven-architecture>
13. 이벤트 기반 아키텍처(EDA)란 무엇인가요? - AWS, 8월 3, 2025에 액세스, <https://aws.amazon.com/ko/what-is/eda/>
14. 이벤트 기반 마이크로서비스 아키텍처: 더 나은 확장성과 유연성을 위한 해법, 8월 3, 2025에 액세스, <https://digitalbourgeois.tistory.com/308>
15. Improve your Game Architecture with Scriptable Objects - YouTube, 8월 3, 2025에 액세스, <https://www.youtube.com/watch?v=bO8WOHCxPq8>
16. Use ScriptableObjects as event channels in game code - Unity, 8월 3, 2025에 액세스, <https://unity.com/how-to/scriptableobjects-event-channels-game-code>
17. Unity Architecture: Scriptable Object Pattern | by Simon Nordon | Medium, 8월 3, 2025에 액세스, <https://medium.com/@simon.nordon/unity-architecture-scriptable-object-pattern-0a6c25b2d741>
18. Outplayed.tv, 8월 3, 2025에 액세스, <https://outplayed.tv/>
19. Outplayed - Desktop App on Overwolf, 8월 3, 2025에 액세스, <https://www.overwolf.com/app/overwolf-outplayed>
20. Best practices for organizing your Unity project, 8월 3, 2025에 액세스, <https://unity.com/how-to/organizing-your-project>
21. Effective Project Management in Game Development - Argentics, 8월 3, 2025에 액세스, <https://www.argentics.io/effective-project-management-in-game-development>
22. Best Agile Practices in Game Development | Starloop Studios Blog, 8월 3, 2025에 액세스, <https://starloopstudios.com/best-agile-practices-in-game-development/>
23. A Blueprint For Successful Game Development - Assembla, 8월 3, 2025에 액세스, <https://get.assembla.com/blog/game-development-best-practices/>
24. Agile Game Dev Best Practices - Number Analytics, 8월 3, 2025에 액세스, <https://www.numberanalytics.com/blog/agile-game-dev-best-practices>
25. Getting started - GameCI, 8월 3, 2025에 액세스, <https://game.ci/docs/gitlab/getting-started/>
26. Character Stats/Attributes in Unity (pt. 2) — Additive Modifiers ..., 8월 3, 2025에 액세스, <https://medium.com/@kryzarel/character-stats-attributes-in-unity-pt-2-additive-modifiers-optimizations-1dfb2d42f3c8>
27. c# - Designing a stats system - Game Development Stack Exchange, 8월 3, 2025에 액세스, <https://gamedev.stackexchange.com/questions/118083/designing-a-stats-system>
28. RPG Statistics Architecture : r/unity - Reddit, 8월 3, 2025에 액세스, <https://www.reddit.com/r/unity/comments/ym9w1g/rpg_statistics_architecture/>
29. A Stat System for Unity part 1, 8월 3, 2025에 액세스, <https://giannisakritidis.com/blog/Stat-System-Part1/>
30. Tips/guides for managing complex character stats and modifiers? : r/godot - Reddit, 8월 3, 2025에 액세스, <https://www.reddit.com/r/godot/comments/14emjwr/tipsguides_for_managing_complex_character_stats/>
31. Make shader behavior conditional on keywords - Unity - Manual, 8월 3, 2025에 액세스, <https://docs.unity3d.com/6000.1/Documentation/Manual/SL-MultipleProgramVariants-make-conditionals.html>
32. How do you deal with stat/damage/etc. modifiers? : r/roguelikedev - Reddit, 8월 3, 2025에 액세스, <https://www.reddit.com/r/roguelikedev/comments/op1s16/how_do_you_deal_with_statdamageetc_modifiers/>
33. How to handle buffs and debuffs ? : r/Unity3D - Reddit, 8월 3, 2025에 액세스, <https://www.reddit.com/r/Unity3D/comments/565ii5/how_to_handle_buffs_and_debuffs/>
34. Buff System with Scriptable Objects for Unity - Unity3D Tutorial - Jonathan Yu, 8월 3, 2025에 액세스, <https://www.jonathanyu.xyz/2016/12/30/buff-system-with-scriptable-objects-for-unity/>
35. 로그라이크 - 나무위키, 8월 3, 2025에 액세스, <https://namu.wiki/w/%EB%A1%9C%EA%B7%B8%EB%9D%BC%EC%9D%B4%ED%81%AC>
36. What Makes A Good DECK BUILDING Roguelike? - YouTube, 8월 3, 2025에 액세스, <https://www.youtube.com/watch?v=UG_T3S6-qUw>
37. Integrating Google Sheets with Unity for Real-Time Data Updates : r/Unity3D - Reddit, 8월 3, 2025에 액세스, <https://www.reddit.com/r/Unity3D/comments/1cyjq7z/integrating_google_sheets_with_unity_for_realtime/>
38. Changing Scriptable Objects values remotely with Google Sheet : r/Unity3D - Reddit, 8월 3, 2025에 액세스, <https://www.reddit.com/r/Unity3D/comments/15ntfrk/changing_scriptable_objects_values_remotely_with/>
39. How To Send Data From Your Unity App to Google Sheets - RudderStack, 8월 3, 2025에 액세스, <https://www.rudderstack.com/guides/send-data-from-your-unity-app-to-google-sheets/>
40. How to get Google Spreadsheet data to Unity. Put Google Api + Auth + Sheets .dll into your project. Then put .p12 service account file somewhere in Assets. - GitHub Gist, 8월 3, 2025에 액세스, <https://gist.github.com/5argon/b01c355cd6635a010df4575b185a5b16>
41. How to create a simple Backend with Google Sheets and Unity: The easiest solution for small projects - Medium, 8월 3, 2025에 액세스, <https://medium.com/@xavidevsama/how-to-create-a-simple-backend-with-google-sheets-and-unity-the-easiest-solution-for-small-ef8a6de1bcd0>
42. [뒤끝팁] 비동기(로컬) PvP 매칭 시스템 만들기!, 8월 3, 2025에 액세스, <https://blog.thebackend.io/%EB%92%A4%EB%81%9D%ED%8C%81-%EB%B9%84%EB%8F%99%EA%B8%B0%EB%A1%9C%EC%BB%AC-pvp-%EB%A7%A4%EC%B9%AD-%EC%8B%9C%EC%8A%A4%ED%85%9C-%EB%A7%8C%EB%93%A4%EA%B8%B0/>
43. Game UI design: the mechanics of fun experiences - Justinmind, 8월 3, 2025에 액세스, <https://www.justinmind.com/ui-design/game>
44. What is the first step in the game UX/UI design process - PUNCHev Group, 8월 3, 2025에 액세스, <https://punchev.com/blog/what-is-the-first-step-in-the-game-ux-ui-design-process>
45. Game UI Design: Your Guide to Better UI - Page Flows, 8월 3, 2025에 액세스, <https://pageflows.com/resources/game-ui-design-leveling-up-your-ui/>
46. TFT Set 15 Classes: All Teamfight Tactics Classes - Mobalytics, 8월 3, 2025에 액세스, <https://mobalytics.gg/tft/synergies/classes>
47. [FINISHED] TFT synergy clusters, visualized (album) : r/TeamfightTactics - Reddit, 8월 3, 2025에 액세스, <https://www.reddit.com/r/TeamfightTactics/comments/cbyclh/finished_tft_synergy_clusters_visualized_album/>
48. Game UI.... - Pantheon Forums, 8월 3, 2025에 액세스, <https://seforums.pantheonmmo.com/content/forums/topic/4665/game-ui>
49. Change Log - Say the Spire Documentation, 8월 3, 2025에 액세스, <https://bradjrenshaw.github.io/say-the-spire/changes.html>
50. 모바일 게임 성능 최적화: 그래픽과 에셋에 관한 전문가 팁 - Unity, 8월 3, 2025에 액세스, <https://unity.com/kr/blog/games/optimize-your-mobile-game-performance-expert-tips-on-graphics-and-assets>
51. 모바일 게임 개발자를 위한 아트 최적화 팁 1부 - Unity, 8월 3, 2025에 액세스, <https://unity.com/kr/how-to/mobile-game-optimization-tips-part-1>
52. Advanced programming and code architecture | Unity, 8월 3, 2025에 액세스, <https://unity.com/how-to/advanced-programming-and-code-architecture>