**유니티 게임 개발자 가이드 (업데이트 버전)**

이 문서는 제공된 C# 스크립트들을 기반으로 유니티 프로젝트 내에서 게임 요소를 설정하고 상호작용하는 방법을 설명합니다.

**1. 개요**

이 프로젝트는 플레이어, 다양한 적(일반 적, 보스), 아이템, 상점, 그리고 게임 진행을 관리하는 시스템을 포함하는 3D 액션 슈팅 게임의 핵심 로직을 구현합니다.

**2. 스크립트 설명 및 설정**

**2.1. Player.cs**

플레이어 캐릭터의 움직임, 공격, 아이템 상호작용, 레벨업 시스템 등을 관리하는 핵심 스크립트입니다.

**주요 변수:**

* **speed**: 플레이어 이동 속도.
* **weapons**: 플레이어가 장착할 수 있는 무기 GameObject 배열. 인스펙터에서 할당합니다.
* **hasWeapons**: 플레이어가 각 무기를 소유하고 있는지 나타내는 bool 배열. 게임 시작 시 GameManager에서 로드됩니다.
* **grenades**: 플레이어가 가진 수류탄 GameObject 배열 (UI 표현용).
* **hasGrenades**: 현재 플레이어가 가진 수류탄 개수.
* **grenadeObj**: 수류탄 프리팹 (던질 때 인스턴스화).
* **followCamera**: 플레이어를 따라다니는 카메라.
* **manager**: GameManager 스크립트 인스턴스. 인스펙터에서 할당합니다.
* **level, currentExperience, maxExperience**: 레벨업 시스템 관련 변수.
* **ammo, coin, health, score**: 플레이어의 현재 상태 변수.
* **maxAmmo, maxCoin, maxHealth, maxHasGrenades**: 각 스탯의 최대값.
* **jumpSound, dodgeSound, interactionSound, swapSound**: 각종 액션 사운드. AudioSource 컴포넌트와 함께 설정해야 합니다.

**필수 컴포넌트:**

* Rigidbody
* Animator
* MeshRenderer (플레이어 모델에 포함된 모든 메시 렌더러)
* AudioSource (점프, 회피, 상호작용, 무기 교체 사운드를 위해 여러 개 필요할 수 있음)

**설정 가이드:**

1. 플레이어 GameObject에 Player.cs 스크립트를 추가합니다.
2. 모든 public 변수들을 인스펙터에서 적절히 할당합니다.
   * Weapons 배열에는 플레이어가 얻을 수 있는 무기 프리팹들을 순서대로 드래그하여 할당합니다. (예: 0번 인덱스: 망치, 1번 인덱스: 핸드건, 2번 인덱스: 머신건)
   * Grenades 배열에는 수류탄 개수를 시각적으로 표시할 UI 요소를 할당합니다.
   * GrenadeObj에는 수류탄 GameObject 프리팹을 할당합니다.
   * Follow Camera에는 메인 카메라를 할당합니다.
   * Manager에는 씬의 GameManager 오브젝트를 할당합니다.
   * AudioSource 컴포넌트를 추가하고 각 사운드 클립을 할당합니다.
3. 플레이어 GameObject에 Rigidbody 컴포넌트를 추가하고, Constraints에서 Freeze Rotation의 X, Y, Z를 모두 체크하여 회전을 고정합니다.
4. 플레이어 모델의 Animator 컴포넌트와 애니메이션 컨트롤러를 연결합니다. 스크립트에서 사용되는 애니메이션 파라미터(isRun, isWalk, isJump, doJump, doDodge, doReload, doSwap, doSwing, doShot, doDie)에 맞춰 설정해야 합니다.
5. 플레이어 모델의 모든 MeshRenderer 컴포넌트가 스크립트에서 참조될 수 있도록 자식 오브젝트에 적절히 배치되어 있어야 합니다.

**2.2. Enemy.cs**

일반 적의 기본적인 행동(추적, 공격, 피격, 사망)을 정의하는 스크립트입니다. 보스 몬스터의 부모 클래스로도 사용됩니다.

**주요 변수:**

* **enemyType**: 적의 종류 (A, B, C, D). 인스펙터에서 설정합니다. Type.D는 보스 몬스터를 위해 예약되어 있습니다.
* **maxHealth, curHealth**: 적의 체력.
* **score**: 적 처치 시 얻는 점수.
* **manager**: GameManager 스크립트 인스턴스.
* **Target**: 추적할 대상 (Player의 Transform).
* **meleeArea**: 근접 공격 범위 (BoxCollider).
* **bullet**: 원거리 공격용 총알 프리팹.
* **coins**: 드롭할 코인 아이템 프리팹 배열.
* **poisonGas**: Type A 적 사망 시 생성될 독가스 프리팹.
* **attackSound, hitSound, deathSound**: 적 관련 사운드.
* **isChase, isAttack, isDead**: 적의 상태 플래그.
* **experiencePoints**: 적 처치 시 플레이어에게 부여할 경험치 (현재는 OnDamage 메서드 내에서 enemyType에 따라 하드코딩).

**필수 컴포넌트:**

* Rigidbody
* BoxCollider (주요 충돌 감지용)
* NavMeshAgent
* Animator
* MeshRenderer (적 모델에 포함된 모든 메시 렌더러)
* AudioSource

**설정 가이드:**

1. 적 GameObject에 Enemy.cs 스크립트를 추가합니다.
2. enemyType을 적절히 설정합니다 (A, B, C).
3. 각 적 타입에 맞는 maxHealth, score, meleeArea, bullet, coins, poisonGas 등을 인스펙터에서 할당합니다.
   * meleeArea는 Is Trigger로 설정된 BoxCollider여야 하며, 공격 애니메이션 타이밍에 맞춰 활성화/비활성화됩니다.
   * bullet 프리팹에는 Bullet.cs 스크립트가 붙어있어야 합니다.
   * coins 배열에는 여러 종류의 코인 아이템 프리팹을 할당합니다.
   * poisonGas 프리팹에는 적절한 파티클 시스템이나 효과가 포함되어야 합니다.
4. NavMeshAgent 컴포넌트를 추가하고, NavMesh를 Bake하여 적이 이동할 수 있는 영역을 설정합니다.
5. Animator 컴포넌트와 애니메이션 컨트롤러를 연결하고, 스크립트에서 사용되는 애니메이션 파라미터(isWalk, isAttack, doDie, doShot, doSwing)에 맞춰 설정합니다.
6. AudioSource 컴포넌트를 추가하고 각 사운드 클립을 할당합니다.
7. 적 GameObject의 레이어를 "Enemy"로 설정합니다. (필요 시 LayerMask.GetMask("Player")와 같이 사용됨)

**2.3. Boss.cs**

Enemy.cs를 상속받아 보스 몬스터의 특수 패턴을 구현하는 스크립트입니다.

**주요 변수:**

* **missile**: 보스 미사일 프리팹.
* **missilePortA, missilePortB**: 미사일 발사 위치 Transform.
* **isLook**: 플레이어를 바라볼지 여부.

**필수 컴포넌트:**

* Enemy 스크립트에 필요한 모든 컴포넌트.

**설정 가이드:**

1. 보스 GameObject에 Boss.cs 스크립트를 추가합니다. (이때 Enemy.cs는 자동으로 추가됩니다).
2. Enemy.cs에서 설명한 대로 기본 Enemy 변수들을 설정합니다. enemyType은 반드시 Type.D로 설정해야 합니다.
3. missile 프리팹에는 BossMissile.cs 스크립트가 붙어있어야 합니다.
4. missilePortA와 missilePortB는 보스 모델의 자식 오브젝트로 생성하여 미사일이 발사될 정확한 위치를 지정합니다.
5. 보스 애니메이션 컨트롤러에 doShot, doBigShot, doTaunt 트리거를 추가하고 해당 애니메이션을 연결합니다.

**2.4. GameManager.cs**

게임의 전반적인 흐름, UI 업데이트, 스테이지 관리, 적 생성, 저장/불러오기 등을 담당하는 스크립트입니다.

**주요 변수:**

* **menuCam, gameCam**: 메인 메뉴 카메라와 게임 플레이 카메라 GameObject.
* **player, boss**: Player 및 Boss 스크립트 인스턴스.
* **itemShop, weaponShop, startZone**: 각 구역 GameObject.
* **stage**: 현재 스테이지.
* **playTime**: 현재 플레이 시간.
* **isBattle**: 전투 중인지 여부.
* **enemyCntA, enemyCntB, enemyCntC, enemyCntD**: 각 타입의 적 현재 개수.
* **enemyZones**: 적이 생성될 구역 Transform 배열.
* **enemies**: 적 프리팹 배열. 순서가 중요합니다 (0: Type A, 1: Type B, 2: Type C, 3: Type D).
* **UI 요소들**: 게임 내 다양한 UI 텍스트(TextMeshProUGUI) 및 이미지(Image), RectTransform (bossHealthGroup, bossHealthBar, expGroup, expBar), 버튼(itemUpgradeButtonA/B/C), levelUpGroup 등을 할당합니다.

**설정 가이드:**

1. 빈 GameObject를 생성하고 이름을 GameManager로 지정한 후 GameManager.cs 스크립트를 추가합니다.
2. 인스펙터에서 모든 public 변수들을 할당합니다.
   * 카메라와 플레이어, 보스(보스 프리팹에 있는 Boss 스크립트 인스턴스)를 할당합니다.
   * 상점 및 시작 존 GameObject를 할당합니다.
   * Enemy Zones에는 적들이 스폰될 위치를 나타내는 빈 GameObject들을 생성하여 할당합니다.
   * Enemies 배열에는 각 타입의 적 프리팹을 순서대로 할당합니다. (A, B, C, D 순)
   * 모든 UI 요소들을 씬에 배치하고 해당 변수에 연결합니다. TextMeshPro UI를 사용한다면 TextMeshProUGUI 타입을 사용하고, 프로젝트에 TextMeshPro 에셋을 임포트해야 합니다.
   * LevelUpGroup은 플레이어가 레벨업했을 때 나타날 패널입니다. 이 패널 안에 itemUpgradeButtonA/B/C를 포함시킵니다.
   * ExpGroup과 ExpBar는 플레이어의 경험치 바 UI입니다.

**게임 시작/종료 로직:**

* GameStart(): 게임을 시작하고 UI를 전환합니다.
* GameQuit(): 게임을 종료합니다. (에디터/빌드 환경에 따라 다르게 작동)
* GameContinue(): 저장된 데이터로 게임을 불러와 시작합니다.
* GameOver(): 게임 오버 시 호출됩니다.
* GameReturn(): 게임 중 메인 메뉴로 돌아갈 때 호출됩니다.
* Restart(): 현재 씬을 다시 로드하여 게임을 재시작합니다.

**스테이지 진행:**

* StageStart(): 현재 스테이지 전투를 시작합니다.
* StageEnd(): 현재 스테이지 전투를 종료하고 다음 스테이지 준비를 합니다.

**2.5. Item.cs**

게임 내에서 획득 가능한 아이템의 종류와 값을 정의하는 스크립트입니다.

**주요 변수:**

* **type**: 아이템의 종류 (Ammo, Coin, Grenade, Heart, Weapon).
* **value**: 아이템의 값 (예: 총알 개수, 코인 금액, 체력 회복량 등).

**필수 컴포넌트:**

* Rigidbody
* SphereCollider

**설정 가이드:**

1. 아이템 프리팹에 Item.cs 스크립트를 추가합니다.
2. type과 value를 인스펙터에서 설정합니다.
3. Rigidbody 컴포넌트를 추가하고 Is Kinematic을 체크 해제하여 물리 영향을 받도록 합니다.
4. SphereCollider를 추가하고 Is Trigger를 체크합니다. OnCollisionEnter와 OnTriggerEnter 모두 사용되므로, 플레이어와의 상호작용은 OnTriggerEnter에서 처리하고, 바닥과의 충돌은 OnCollisionEnter에서 처리합니다.
5. 아이템 GameObject의 레이어를 "Item"으로 설정합니다.

**2.6. Weapon.cs**

무기의 기본적인 속성(공격력, 연사 속도, 탄창, 현재 탄약)을 정의하는 스크립트입니다.

**주요 변수:**

* **type**: 무기의 종류 (Melee, Range).
* **damage**: 무기의 공격력.
* **rate**: 연사 속도 (공격 딜레이).
* **maxAmmo**: 최대 탄창 크기.
* **curAmmo**: 현재 탄창 내 탄약.
* **meleeArea**: 근접 공격 범위 (BoxCollider).
* **trailEffect**: 근접 무기의 공격 시 활성화되는 트레일 효과.
* **bulletPos**: 총알이 발사될 위치 Transform.
* **bullet**: 발사될 총알 프리팹 (Range 타입 무기용).
* **bulletCasePos**: 탄피가 배출될 위치 Transform.
* **bulletCase**: 탄피 프리팹.
* **attackSound**: 공격 시 재생될 사운드.

**필수 컴포넌트:**

* AudioSource
* (Melee 무기인 경우) BoxCollider (meleeArea 변수용)
* (Melee 무기인 경우) TrailRenderer (trailEffect 변수용)

**설정 가이드:**

1. 무기 프리팹에 Weapon.cs 스크립트를 추가합니다.
2. type을 설정하고 damage, rate, maxAmmo 등을 설정합니다.
3. Melee 타입 무기에는 공격 시 활성화될 BoxCollider를 추가하고 Is Trigger로 설정한 후 meleeArea에 할당합니다. TrailRenderer 컴포넌트를 추가하고 trailEffect에 할당합니다.
4. Range 타입 무기에는 발사될 bullet 프리팹을 할당하고, 총알이 발사될 bulletPos (Transform)를 무기 모델의 총구 위치에 자식 오브젝트로 생성하여 할당합니다.
5. 탄피 배출을 위해 bulletCasePos (Transform)를 무기 모델에 자식 오브젝트로 생성하여 할당하고, bulletCase에 탄피 프리팹을 할당합니다.
6. AudioSource 컴포넌트를 추가하고 attackSound 클립을 할당합니다.

**2.7. Bullet.cs**

총알의 기본적인 속성(공격력)과 수명, 충돌 처리를 정의하는 스크립트입니다. BossMissile과 BossRock의 부모 클래스입니다.

**주요 변수:**

* **damage**: 총알의 공격력.
* **isMelee**: 근접 공격인지 여부 (false면 원거리 공격).
* **isRock**: 보스 락(바위)인지 여부.

**필수 컴포넌트:**

* Collider (예: SphereCollider 또는 BoxCollider)
* Rigidbody

**설정 가이드:**

1. 총알 프리팹에 Bullet.cs 스크립트를 추가합니다.
2. damage를 설정합니다.
3. Rigidbody 컴포넌트를 추가하고 Is Kinematic을 체크 해제합니다.
4. Collider를 추가하고 Is Trigger를 체크합니다.
5. 총알 GameObject의 레이어를 "EnemyBullet" 또는 "PlayerBullet" 등으로 설정하여 적절한 충돌 감지를 합니다.
6. isMelee와 isRock은 해당 불릿이 근접 공격 판정인지, 보스 락인지에 따라 적절히 체크합니다.

**2.8. BossMissile.cs**

Bullet.cs를 상속받아 보스 미사일의 추적 기능을 구현하는 스크립트입니다.

**주요 변수:**

* **target**: 추적할 대상의 Transform (플레이어).

**필수 컴포넌트:**

* Bullet 스크립트에 필요한 모든 컴포넌트.
* NavMeshAgent

**설정 가이드:**

1. 보스 미사일 프리팹에 BossMissile.cs 스크립트를 추가합니다. (이때 Bullet.cs는 자동으로 추가됩니다).
2. NavMeshAgent 컴포넌트를 추가하고, NavMesh를 Bake하여 미사일이 추적할 수 있는 영역을 설정합니다.
3. target 변수는 보스 스크립트에서 할당됩니다.

**2.9. BossRock.cs**

Bullet.cs를 상속받아 보스 몬스터의 바위 공격을 구현하는 스크립트입니다. 발사 전 힘을 모으는 효과를 가집니다.

**주요 변수:**

* (없음, 내부적으로 angularPower, scaleValue, isShoot 사용)

**필수 컴포넌트:**

* Bullet 스크립트에 필요한 모든 컴포넌트.
* Rigidbody

**설정 가이드:**

1. 보스 락 프리팹에 BossRock.cs 스크립트를 추가합니다. (이때 Bullet.cs는 자동으로 추가됩니다).
2. Rigidbody 컴포넌트가 존재해야 합니다.
3. 바위의 시각적인 변화 (크기 증가)와 회전 효과를 위한 설정이 필요합니다.

**2.10. Grenade.cs**

수류탄의 폭발 및 주변 적에게 데미지를 주는 기능을 구현하는 스크립트입니다.

**주요 변수:**

* **meshObj**: 수류탄의 메시 GameObject (폭발 시 비활성화).
* **effectObj**: 폭발 이펙트 GameObject (폭발 시 활성화).
* **rigid**: 수류탄의 Rigidbody.

**필수 컴포넌트:**

* Rigidbody

**설정 가이드:**

1. 수류탄 프리팹에 Grenade.cs 스크립트를 추가합니다.
2. 수류탄 모델과 폭발 이펙트 GameObject를 자식으로 만들고, 각각 meshObj와 effectObj에 할당합니다. effectObj는 초기 상태에서 비활성화되어 있어야 합니다.
3. Rigidbody 컴포넌트를 추가하고, 초기에는 Is Kinematic을 해제하여 물리 영향을 받도록 합니다.
4. 폭발 범위(Physics.SphereCastAll의 15)와 영향 레이어(LayerMask.GetMask("Enemy"))를 필요에 따라 조정합니다.

**2.11. Missile.cs**

(이 스크립트는 단순히 미사일이 회전하는 기능만 포함하고 있어, 플레이어의 미사일 또는 다른 일반적인 회전하는 오브젝트에 사용될 것으로 보입니다. 특정 발사 로직은 포함되어 있지 않습니다.)

**주요 변수:**

* (없음)

**필수 컴포넌트:**

* (없음)

**설정 가이드:**

1. 회전하는 미사일 GameObject에 Missile.cs 스크립트를 추가합니다.
2. 특별히 설정할 public 변수는 없습니다.

**2.12. Follow.cs**

카메라와 같은 오브젝트가 특정 target을 따라다니도록 하는 스크립트입니다.

**주요 변수:**

* **target**: 따라다닐 대상의 Transform.
* **offset**: target으로부터의 상대적인 위치 오프셋.

**필수 컴포넌트:**

* (없음)

**설정 가이드:**

1. 카메라 GameObject에 Follow.cs 스크립트를 추가합니다.
2. target에 플레이어의 Transform을 할당합니다.
3. offset 값을 조정하여 카메라의 위치를 플레이어 기준으로 설정합니다.

**2.13. shop.cs**

상점 기능을 구현하는 스크립트입니다. 플레이어가 상점에 진입하면 UI가 나타나고, 아이템을 구매할 수 있습니다.

**주요 변수:**

* **uiGroup**: 상점 UI 패널의 RectTransform.
* **anim**: 상점 NPC의 Animator (상점 진입/퇴장 애니메이션).
* **itemObj**: 상점에서 판매할 아이템 프리팹 배열.
* **itemPrice**: 각 아이템의 가격 배열 (인덱스가 itemObj와 일치해야 함).
* **itemPos**: 아이템이 드롭될 위치 Transform 배열.
* **talkText**: 상점 NPC의 대화 텍스트 (TextMeshProUGUI가 아닌 Text 타입이므로, using UnityEngine.UI; 필요).
* **buySound**: 구매 시 재생될 사운드.

**필수 컴포넌트:**

* Animator
* AudioSource
* Collider (플레이어가 상점에 들어왔음을 감지하기 위한 트리거 콜라이더)

**설정 가이드:**

1. 상점 GameObject에 shop.cs 스크립트를 추가합니다.
2. uiGroup에 상점 UI 패널의 RectTransform을 할당합니다. 초기에는 비활성화 또는 화면 밖으로 이동시켜 놓습니다.
3. 상점 NPC 모델에 Animator 컴포넌트를 추가하고 anim에 할당합니다. doHello 트리거 애니메이션이 필요합니다.
4. itemObj 배열에 판매할 아이템 프리팹들을 할당하고, itemPrice 배열에 해당하는 가격을 설정합니다.
5. itemPos 배열에 아이템이 스폰될 위치를 나타내는 빈 GameObject들을 생성하여 할당합니다.
6. talkText에 상점 대화 UI 텍스트 컴포넌트를 할당합니다. (유니티 기본 Text 컴포넌트가 필요하며, TextMeshPro를 사용한다면 TextMeshProUGUI로 변경하고 스크립트도 수정해야 합니다.)
7. AudioSource 컴포넌트를 추가하고 buySound 클립을 할당합니다.
8. 상점 GameObject에 Collider 컴포넌트를 추가하고 Is Trigger를 체크하여 플레이어가 상점에 진입했는지 감지하도록 합니다. 이 콜라이더의 태그는 "shop"으로 설정해야 Player.cs에서 감지할 수 있습니다.

**3. 핵심 시스템 상호작용**

**3.1. 플레이어 레벨업 시스템**

* 플레이어가 적을 처치하면 Player.AddExperience() 메서드를 통해 경험치를 획득합니다.
* 획득한 경험치가 maxExperience를 넘어서면 Player.LevelUp()이 호출됩니다.
* LevelUp()에서는 GameManager.ShowLevelUpPanel()을 호출하여 게임을 일시 정지하고 레벨업 옵션 UI를 표시합니다.
* 플레이어는 GameManager의 OnUpgradeSelected() 메서드를 통해 업그레이드 선택을 하면, 해당 선택에 따라 Player.ApplyUpgrade()가 호출되어 무기의 공격력 등이 증가합니다.
* 업그레이드 적용 후 게임은 다시 재개됩니다.

**3.2. 적 스폰 및 스테이지 진행**

* GameManager.StageStart()가 호출되면 적들이 스폰되기 시작합니다.
* 스테이지 5의 배수일 때(stage % 5 == 0), 보스 몬스터(Enemy.Type.D)가 스폰됩니다.
* 일반 스테이지에서는 enemyList에 무작위로 적 타입(A, B, C)을 추가하고, 순차적으로 스폰합니다.
* 모든 적(A, B, C, D)이 처치되면(enemyCntA + enemyCntB + enemyCntC + enemyCntD > 0 조건이 거짓이 되면), GameManager.StageEnd()가 호출되어 다음 스테이지로 넘어갈 준비를 합니다.

**3.3. 저장 및 불러오기**

* GameManager.GameReturn() 시 PlayerPrefs를 사용하여 플레이어의 현재 스테이지, 플레이 시간, 수류탄 개수, 레벨, 경험치, 총알, 코인, 체력, 점수 및 무기 보유 여부를 저장합니다.
* GameManager.GameContinue() 시 저장된 데이터를 불러옵니다.
* PlayerPrefs는 단순한 데이터 저장 방식이므로, 더 복잡한 저장 시스템이 필요하다면 다른 직렬화 방식을 고려해야 합니다.

**3.4. UI 업데이트**

* GameManager.LateUpdate()에서 게임 내 모든 UI 요소(점수, 시간, 플레이어 스탯, 무기 아이콘, 적 숫자, 보스 체력 바, 경험치 바)를 실시간으로 업데이트합니다.

**3.5. 상점 시스템**

* 플레이어가 상점의 트리거 콜라이더에 진입하면 Player.OnTriggerStay()를 통해 nearObject가 상점 GameObject로 설정되고, 상호작용 키를 누르면 Player.Interaction()을 통해 shop.Enter()가 호출되어 상점 UI가 나타납니다.
* 상점 UI에서 버튼을 클릭하면 shop.Buy() 메서드가 호출되고, 플레이어의 코인이 충분하면 아이템이 구매되고 드롭됩니다.
* 상점 UI를 닫으면 shop.Exit()이 호출되어 UI가 사라지고 상점 NPC의 애니메이션이 재생됩니다.

**4. 추가 개발 고려사항**

저희 프로젝트는 현재 기본적인 게임 플레이 시스템을 갖추고 있지만, 사용자 경험과 성능을 더욱 향상시키기 위해 다음과 같은 추가 개발 사항들을 고려하고 있습니다.

* **사운드 관리 (Audio Management)**: 각 사운드 클립(공격, 피격, 사망 등)에 대해 AudioSource 컴포넌트를 체계적으로 설정하고, 볼륨 조절 및 오디오 믹서 그룹을 활용하여 게임 내 사운드를 더욱 효과적으로 관리할 예정입니다. 이를 통해 사용자에게 몰입감 있는 청각적 경험을 제공할 수 있습니다.
* **이펙트 관리 (Visual Effects Management)**: 공격, 피격, 적 사망 시 발생하는 파티클 이펙트를 추가하여 시각적인 피드백을 강화할 계획입니다. 이는 플레이어에게 전투 상황을 더 명확하게 인지시키고, 액션의 타격감을 높이는 데 기여할 것입니다.
* **게임 밸런싱 (Game Balancing):** 적의 체력과 공격력, 플레이어의 스탯(체력, 공격력, 이동 속도 등), 경험치 획득량, 아이템 드롭률 등을 지속적으로 조정하여 게임의 난이도와 재미를 최적화할 예정입니다. 플레이어가 적절한 도전과 성장을 느낄 수 있도록 세밀한 밸런싱 작업이 필요합니다.
* **애니메이션 이벤트 활용 (Animation Events)**: Unity의 애니메이션 이벤트 기능을 적극적으로 활용하여 공격 범위 활성화/비활성화, 총알 발사, 피격 타이밍 등 다양한 게임 로직을 애니메이션의 특정 프레임에 맞춰 정밀하게 제어할 것입니다. 이는 시각적/청각적 요소와 게임 플레이의 동기화를 통해 더욱 자연스럽고 완성도 높은 경험을 제공하는 데 필수적입니다.