OVERDRIVE

Unity 6 · URP 기반 3인칭 웨이브 슈터 프로토타입

조현준

그림 1. 보스전 — Warden 의 피격 플래시와 호위 Grunt, 상단 보스 HP 바

유형 개인 프로젝트

사용 기술 Unity 6 · URP · C# · Input System

기간 2026.09 · 3일 집중 제작

실행 Windows x64

담당 기획 · 구현 · 검증 · 문서 전부 (AI 협업)

자료 구성 소스코드 · 실행 파일 · 자동 플레이 영상 · 이 문서

본인 기여

핵심 시스템인 카메라 기준 이동, 3인칭 카메라, Raycast 사격과 관통, 적 AI 와 Golden Angle 배치, Object Pool, 웨이브 진행, 강화 6종, 보스 Slam 은 AI 의 설명과 리뷰를 참고하되 코드를 직접 타이핑하고 디버깅했습니다. 마감 단계에서 AI 가 작성한 코드는 직접 검토 · 통합 · 테스트했고, 데이터 흐름과 클래스별 책임을 설명할 수 있는 수준으로 정리했습니다.

AI 수행

개념 설명, Unity API 확인, 코드 리뷰, 오류 원인 추적에 활용했습니다. 마감 단계의 대시, 보스 Charge 와 HP 임계 소환, HUD 와 Start / Pause / Result 화면, VFX 와 합성 SFX, 씬 자동 구성 스크립트, 자동 플레이 검증 · 녹화 봇, 빌드 · 인코딩 스크립트, README 초안은 AI 가 작성했습니다.

자료 기준 2026.09.07

2026 넥토리얼 서류 제출용 · Windows 빌드 · 자동 플레이 영상 · 검증 리포트 동봉

문서 구성

목차

게임의 범위와 조작부터 핵심 구현, 자동 플레이 검증, 개발 과정까지

한 판은 4~6분이며, 시작부터 클리어까지 끊기지 않고 돌아가는 전투 세로 슬라이스 하나를 3일 안에 완성하는 것을 목표로 했습니다.

01 게임 이해

목적과 범위

에너지 소총을 강화하며 세 차례의 군집 공세를 돌파하고 지휘 유닛 Warden 을 파괴하는 3인칭 웨이브 액션 프로토타입입니다. 한 판은 4~6분입니다.

무엇을 완성하려 했는가

목표는 큰 게임이 아니라, 시작부터 클리어까지 끊기지 않고 돌아가는 전투 세로 슬라이스 하나를 직접 설명할 수 있는 코드로 완성하는 것이었습니다. 3일 안에 끝내기 위해 사람형 캐릭터와 애니메이션 대신 Unity Primitive 와 발광 Material 로 만든 호버 유닛과 기하학형 적을 사용했고, 외부 에셋은 쓰지 않았습니다.

플레이어는 관통탄과 처치 폭발을 조합해 적 무리를 한 번에 무너뜨리는 성장 흐름을 경험하고, 마지막에는 공격 예고(Telegraph)를 읽고 대시로 회피하며 보스를 처치합니다.

구현 범위

장르 · 플랫폼	3인칭 슈팅 + 웨이브 기반 로그라이트, 싱글플레이 / Windows PC
엔진 · 패키지	Unity 6000.5.10f1 · Universal RP 17.5.0 · Input System 1.20.0 · uGUI
플레이어	호버 전투 유닛 1, 에너지 소총 1 (HP 100 · 탄창 32 · 피해 12)
적	일반 적 Grunt / Heavy (같은 AI, 데이터 변형) · 보스 Warden (Slam · Charge)
진행	웨이브 3 → 강화 3택 3회 → 보스전 → Clear / Game Over, 강화 6종 (최대 3중첩)
에셋	외부 에셋 없음 — Primitive 모델, 코드 합성 효과음, LineRenderer 트레이서, URP Bloom

플레이 흐름

조작 안내 (ENTER) → Wave 1 → 강화 3택 → Wave 2 → 강화 3택 → Wave 3 → 강화 3택 → Warden → CLEAR

플레이어 HP 가 0 이 되면 즉시 GAME OVER 로 전환되고, Clear 와 Game Over 화면에서는 R 로 현재 씬을 재시작합니다. 웨이브 종료 후 강화 3택 동안은 `Time.timeScale = 0` 으로 전투를 멈춥니다.

웨이브	Grunt	Heavy	동시 활성 최대
Wave 1	24	0	14
Wave 2	36	6	22
Wave 3	50	10	32
보스전	호위 Grunt 6, 보스 HP 70% / 35% 에서 8 마리씩 추가 소환		

웨이브 수량은 기획값을 그대로 사용했습니다. 난이도 조정은 플레이 데이터가 더 쌓인 뒤로 미뤘습니다.

01 게임 이해

조작과 실제 화면

카메라 기준 WASD 이동과 마우스 조준, 좌클릭 연속 사격, SPACE 대시로 구성했습니다. 자동 플레이 검증도 아래와 같은 입력 경로를 그대로 사용합니다.

조작

W A S D	카메라 기준 이동 (6.0 m/s)
마우스	카메라 회전 / 조준 (오른쪽 어깨 3인칭)
좌클릭 (유지)	연속 사격 — 기본 발사 간격 0.125초, 사거리 70m
R	재장전 1.2초 (탄약 0 에서 사격 시 자동 재장전) · 결과 화면에서는 현재 씬 재시작
SPACE	이동 방향 대시 4m · 0.2초 · 쿨다운 1.2초 (대시 중 무적 / 사격 불가 / 재장전은 계속)
ESC · ENTER	일시정지 / 재개 · 시작 화면에서 게임 시작

피격 후 0.35초 무적과 대시 무적은 모두 `PlayerHP.TakeDamage` 가 조기 반환하는 같은 경로로 처리하므로, 보스의 Charge 와 Slam 도 대시로 회피할 수 있습니다.

실제 화면

그림 2. 시작 화면 — 조작 안내, ENTER 로 시작

그림 3. 웨이브 종료 후 강화 3택 — 1회차는 관통형 총알 고정

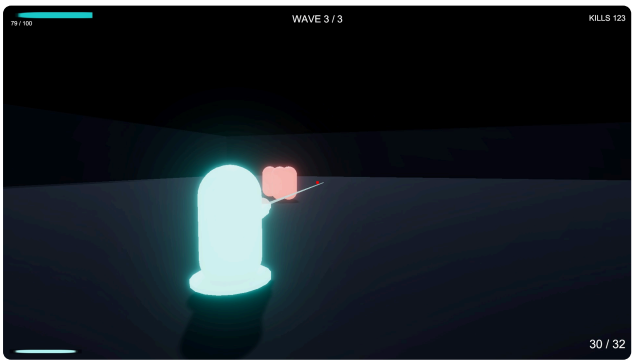

그림 4. Wave 3 전투 — Heavy 군집과 트레이서

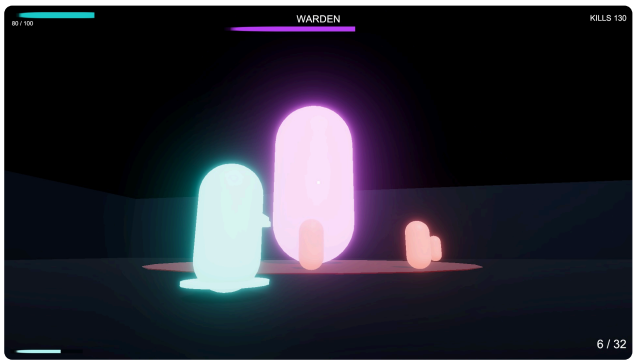

그림 5. 보스전 — 바닥의 원형 Slam Telegraph 와 Warden HP 바

HUD 는 좌상단 HP, 상단 중앙 웨이브(보스전에서는 WARDEN HP 바), 우상단 처치 수, 우하단 탄약, 좌하단 대시 게이지로 구성했고, 적중 시 히트 마커와 피격 펄스를 표시합니다.

02 핵심 구현 · 1/4

데이터 변형 적과 Golden Angle 군집

일반 적은 한 종류의 AI 코드에 수치만 바꿔 재사용하고, 군집 배치는 이웃 탐색 없이 스폰 순번만으로 분산시켰습니다.

a. 데이터 변형 기반 적 재사용

Grunt 와 Heavy 는 같은 `EnemyController` 를 쓰고 수치만 `EnemyConfig` ScriptableObject 로 분리했습니다. 풀에서 꺼낼 때 `Activate(config, spawnSerial)` 로 주입하므로 프리팹은 외형만 다릅니다.

항목	Config_Grunt	Config_Heavy
MaxHP	24	75
MoveSpeed	3.4	2.2
Scale	1.0	1.45
AttackDamage	7	14
AttackRange	1.4	1.7
WindupTime	0.25	0.40
AttackCooldown	0.95	1.20

적 상태는 `Chase → Windup → Cooldown` 을 반복하고 사망 시 `Dead` 로 갑니다. 코루틴 없이 `m_StateEndTime` 을 `Time.time` 과 비교하는 방식으로 통일해, 풀에서 재사용해도 남아 있는 코루틴이 없습니다.

b. Golden Angle 군집 배치

적마다 스폰 순번(`spawnSerial`)에 137.5° 를 곱한 고유 각도를 주고, 플레이어 중심 원 위의 개인 목표점으로 이동합니다.

EnemyController.cs · GetChasePosition() 일부

```
private Vector3 GetChasePosition()
{
    float radian = m_SpawnSerial * k_GoldenAngle * Mathf.Deg2Rad;
    Vector3 offset = new Vector3(Mathf.Cos(radian), 0.0f, Mathf.Sin(radian)) * m_Config.AttackRange;

    return m_Player.position + offset;
}
```

이웃 탐색이나 분리(Separation) 계산이 없어 적 한 마리당 O(1) 이며, 적끼리 한 점에 완전히 겹치는 현상을 줄입니다. Layer Collision Matrix 에서 Enemy ↔ Enemy 충돌을 꺼 두었기 때문에 물리로 서로를 밀어낼 필요가 없고, 적은 Rigidbody 없이 `transform.position` 으로 이동합니다.

`k_GoldenAngle` 은 EnemyController 의 상수 137.5f 입니다. 적 간의 정교한 회피는 이번 범위에서 제외했습니다.

02 핵심 구현 · 2/4

조합 가능한 총기 강화

`PlayerStats` 가 중첩 수만 저장하고 최종 수치는 계산 프로퍼티로 노출합니다. `PlayerWeapon` 은 발사 시점에 이 값을 읽기만 합니다.

강화 6종과 수식

카드	프로퍼티	수식 / 효과
데미지 강화	`Damage`	12 × (1 + 0.25 × DamageStack)
연사속도 증가	`FireInterval`	$\max(0.055, 0.125 \times 0.82^{\text{RateStack}})$
대용량 탄창	`MagazineSize`	32 + 16 × MagazineStack (선택 즉시 현재 탄약 +16)
관통형 총알	`MaxHitCount`	1 + PierceStack
폭발 프로토콜	`BlastStack`	처치 폭발 피해 24 / 36 / 48, 반경 2.5 / 2.8 / 3.1 m
체력 흡수	`HealPerKill`	2 × RepairStack

모든 강화는 최대 3중첩이며, 최대 중첩에 도달한 강화는 후보에서 제외합니다. 카드 3장은 `UpgradeManager.BuildChoices()` 의 약한 보장 규칙을 따릅니다. 1회차는 관통형 총알 고정 + 기본 강화 2장, 2회차는 폭발 프로토콜 고정, 3회차는 전체 풀에서 무작위입니다. 관통 → 폭발 순서가 반드시 보이도록 한 장치입니다.

관통 — 고정 버퍼와 거리순 정렬

관통은 `Physics.RaycastNonAlloc` 으로 고정 버퍼(16)에 받은 뒤 거리순 삽입 정렬(`SortHitsByDistance`)하고, `ApplyHits` 에서 `MaxHitCount` 만큼만 `IDamageable` 에 피해를 줍니다. 같은 대상은 `m_HitTargets` 리스트로 중복 제거하고, 카메라 Ray 의 원점은 플레이어 위치까지 앞으로 당겨 등 뒤의 적을 맞히지 않게 했습니다.

PlayerWeapon.cs · ApplyHits() 일부

```
for (int i = 0; i < count; i++)
{
    if (m_HitTargets.Count >= maxHits)
        break;

    RaycastHit hit = m_Hits[i];
    endPoint = hit.point;

    IDamageable target = hit.collider.GetComponent<IDamageable>();
```

처치 폭발 — 재귀 없는 1단계 연쇄

처치 폭발은 `ExplosionSystem.TryExplode(position, source)` 에서 `source != DamageSource.Bullet` 이면 즉시 반환합니다. 폭발로 죽은 적은 `DamageSource.Explosion` 으로 사망하므로 다시 폭발하지 않고, 연쇄는 1단계로 끝납니다. 반경 검색은 `OverlapSphereNonAlloc` 고정 버퍼를 사용합니다.

02 핵심 구현 · 3/4

보스 상태머신과 Telegraph

Warden 은 Slam 과 Charge 두 패턴을 Windup → 실행 → Recovery 로 나누고, 예고 표시와 실제 판정이 같은 값을 쓰도록 만들었습니다.

BossController 상태

```
Idle → Intro → Chase ─┬─ SlamWindup → SlamRecovery ─┐
                      └─ ChargeWindup → Charge → ChargeRecovery ─┘ → Chase …
                                                                 → Dead
```

01 패턴 선택

Chase 2.0초 후 플레이어와 수평 거리 12m 이상이면 Charge, 미만이면 Slam 을 고릅니다. 같은 패턴 3연속은 금지하고 뒤집습니다.

02 Slam

Windup 1.0초 동안 원형 Telegraph 가 `m_SlamRadius`(9m) 까지 커지고, 시간이 끝나면 같은 반경 안의 플레이어에게 20 피해를 줍니다. 반경 값 하나가 표시와 판정을 함께 결정하므로 예고와 실제 범위가 어긋날 수 없습니다.

03 Charge

Windup(0.85초) 시작 시 플레이어 위치를 목표점으로 저장하고 아레나 안쪽으로 Clamp 합니다. 납작한 Cube Telegraph 의 폭은 `m_ChargeHitRadius × 2`, 길이는 목표점까지 거리입니다. 0.4초 동안 `Vector3.Lerp` 로 돌진하며 피해(25)는 한 번의 Charge 에서 1회만 적용하고, 돌진 중에는 목표를 다시 추적하지 않습니다.

04 HP 임계 소환

`TakeDamage` 후 `HealthRatio` 가 70% / 35% 이하로 처음 내려갈 때 `OnSummonRequested(8)` 를 한 번씩 발행합니다. 실제 스폰은 `WaveDirector.SpawnBossEscort` 가 담당하고, 보스는 스폰을 모릅니다.

BossController.cs · DoSlam() 일부

```
float diameter = m_SlamRadius * 2.0f;
m_SlamTelegraph.localScale = new Vector3(diameter, k_TelegraphHeight, diameter);

Vector3 toPlayer = m_Player.position - transform.position;
toPlayer.y = 0.0f;

bool hit = toPlayer.magnitude <= m_SlamRadius;

if (hit)
    DamagePlayer(m_SlamDamage, DamageSource.BossSlam);
```

보스 HP 는 2000 이며, 대시 무적 중에는 `PlayerHP.TakeDamage` 가 조기 반환하므로 Charge 와 Slam 모두 회피할 수 있습니다. 위 수치는 BossController 의 Inspector 설정값입니다.

02 핵심 구현 · 4/4

Object Pool 과 이벤트 기반 책임 분리

적은 풀에서 꺼내 쓰고 두 단계로 돌려보내며, 전투 객체는 이벤트만 발행합니다.

e. Object Pool 과 2단계 회수

`SimplePool` 은 Prefab 과 Prewarm 수량을 받는 Queue 기반 풀입니다(Grunt 40, Heavy 12). 고갈 시 `LogWarning` 후 1개만 확장하고, `Activate` 시 HP · 상태 · Collider · Scale · 플래시 · 회수 플래그를 초기화해 재사용 잔재를 없앴습니다.

적 회수 순서

```
EnemyController.Die → OnDied 즉시 발행        : WaveDirector 가 alive--, kill++, 폭발 판정, 처치 회복
0.2초 사망 연출       → Scale 축소 + 피격 방향으로 밀림
연출 종료             → OnDespawnReady 발행     : WaveDirector 가 SimplePool.Release
```

WaveDirector.cs · HandleEnemyDied() 일부

```
enemy.OnDied -= HandleEnemyDied;

m_AliveCount--;
m_KillCount++;

OnKillCountChanged?.Invoke(m_KillCount);

m_ExplosionSystem.TryExplode(enemy.transform.position, source);
m_PlayerHP.HealOnKill();
```

카운트와 폭발은 `OnDied` 시점에 즉시 처리해 웨이브 종료 판정이 지연되지 않습니다.

f. 이벤트 기반 책임 분리

전투 객체는 `public event Action<...> OnXxx` 만 발행하고, 게임 흐름 결정은 `GameManager` 한 곳에서만 합니다. 구독은 `OnEnable`, 해제는 `OnDisable` 에서 짝을 맞추며, 필드 `m_PascalCase` · 상수 `k_PascalCase` · 코루틴과 LINQ 미사용을 규약으로 했습니다.

발행	구독
`PlayerWeapon.OnFired / OnAmmoChanged`	HUDController, CombatFeedback, ProceduralSfx
`PlayerHP.OnDied`	GameManager (GameOver)
`EnemyController.OnDied / OnDespawnReady`	WaveDirector
`WaveDirector.OnWaveCompleted`	GameManager (Reward)
`BossController.OnSummonRequested`	GameManager → WaveDirector.SpawnBossEscort
`UpgradePanel.OnCardSelected`	GameManager → UpgradeManager.Apply
`GameManager.OnStateChanged`	ScreenController (패널 전환)

03 검증과 성능 · 1/2

자동 플레이 검증

가상 키보드 · 마우스로 실제 입력 경로를 태우는 드라이버가 한 판을 스스로 완주하고, 입력 · 상태 · 에러를 리포트로 남깁니다.

게임 코드에 훅을 두지 않은 검증

`Assets/_Overdrive/Scripts/Testing/` 의 `AutoPlayDriver` 는 가상 `Keyboard` / `Mouse` 장치를 Input System 에 등록하고 ENTER, WASD, SPACE, 좌클릭, R, ESC 를 상태 이벤트로 큐잉합니다. 게임 코드에는 테스트 전용 훅을 두지 않았고, 카메라 Yaw / Pitch 와 보스 Telegraph 는 리플렉션으로 읽습니다. Testing 폴더는 에디터와 Autoplay 빌드에서만 컴파일되므로 제출 빌드에는 포함되지 않습니다.

AutoPlayDriver.cs · QueueKeyboard() 일부

```
private void QueueKeyboard()
{
    Key[] keys = new Key[m_HeldKeys.Count + m_TapKeys.Count];
    m_HeldKeys.CopyTo(keys, 0);
    m_TapKeys.CopyTo(keys, m_HeldKeys.Count);

    InputSystem.QueueStateEvent(m_Keyboard, new KeyboardState(keys));
}
```

두 가지 모드

모드	용도	실행
Verify	헤드리스 회귀 검증. 치트 처치로 시작 → 웨이브 3 → 보상 3 → 보스 → Clear → 재시작을 빠르게 돌고 입력 · 상태 · 에러 0건을 확인	`AutoPlayTests` PlayMode 테스트
Showcase	영상용. 실제로 조준 · 사격 · 카이팅 · 대시 · Telegraph 회피를 하며 하이라이트 구간만 30fps 프레임으로 녹화	`Overdrive.exe -autoplay`

리포트 결과

항목	Verify	Showcase
결과	PASS	PASS
웨이브 시작 / 보상 선택	3회 / 3회	3회 / 3회
발사 (명중)	5발 — 입력 확인용	878발 (명중 377발, 43%)
보스 소환 요청 / Slam / 플레이어 피격	2회 / 2회 / 4회	2회 / 4회 / 7회
Clear 시 게임 시간 / 처치	55.5초 / 148	156.0초 / 143
에러 / 예외	0건	0건

Verify 는 이동 · 대시 · 재장전 · 일시정지 검사를 함께 통과했고, 두 모드 모두 재시작 후 Boot 상태를 확인했습니다.

03 검증과 성능 · 2/2

영상 제작과 성능 개선

제출 영상은 Showcase 봇이 플레이한 실제 빌드 화면을 Unity 만으로 인코딩한 것이고, 성능 개선은 구조 변경과 스폰 로그 수치로 설명합니다.

영상 제작 방식

Showcase 봇은 가장 가까운 적을 조준하고 위협과의 거리에 따라 카이팅하며, Slam 과 Charge Telegraph 가 켜지면 대시로 회피합니다. 녹화는 하이라이트 구간(시작, 강화, Wave 3, 보스, 클리어)에서만 30fps 프레임을 덤프하고, `OverdriveBuild.EncodeAutoplayVideo` 가 `UnityEditor.Media.MediaEncoder` 로 H.264 MP4 로 묶습니다. 외부 인코더는 쓰지 않았습니다. 이번 영상은 2구간 2223프레임(74.1초)이며, 봇의 안전 회복(HP 30% 미만에서 회복)은 0회였습니다. 보상은 관통형 총알 → 폭발 프로토콜 → 체력 흡수 순서로 선택했습니다.

그림 6. CLEAR 화면 — Showcase 봇 완주 결과 (TIME 02:36 · KILLS 143), R 로 재시작

성능 개선 사례

01 Instantiate / Destroy → Object Pool

스폰마다 `Instantiate`, 사망 시 `Destroy` 하던 것을 `SimplePool` 이 Awake 에서 Prewarm(Grunt 40, Heavy 12)하고 Get / Release 로 재사용하도록 바꿨습니다. 개발 중 남긴 스폰 로그 기준, 한 판 총 212회 스폰에 실제 생성된 객체는 120개, 나머지 92회는 재사용이었습니다.

02 Enemy 물리 충돌 제거

적끼리 Collider 로 충돌해 군집이 서로를 밀어내던 것을 Layer Collision Matrix 에서 Enemy ↔ Enemy 비활성화로 없앴습니다. 적은 Rigidbody 없이 `transform.position` 으로 이동하고 겹침 방지는 Golden Angle 개인 접근 각도로 대체했으며, Collider 는 총탄 Raycast 용으로만 유지합니다.

03 프레임당 할당 제거

`RaycastNonAlloc` 과 `OverlapSphereNonAlloc` 고정 버퍼, `MaterialPropertyBlock` 재사용, 트레이서 · 폭발 VFX 풀링, 코루틴과 LINQ 미사용을 규약으로 정했습니다.

Unity Profiler 프레임 수치는 측정하지 않아 싣지 않았습니다. 위 사례는 구조 변경과 스폰 로그를 기준으로 한 설명입니다.

04 개발 과정과 AI 협업

3일 타임라인과 역할

생성형 AI 를 페어 프로그래밍 방식으로 사용하되, 핵심 시스템은 직접 타이핑하고 마감 단계의 AI 코드는 검토 · 통합 · 테스트했습니다.

Day 1 **총으로 적을 죽이는 상태**
카메라 기준 이동, 오른쪽 어깨 3인칭 카메라, Raycast 연속 사격과 탄창 · 재장전, 추적 · 근접 공격하는 적, 플레이어 HP 와 Game Over 를 연결했습니다.

Day 2 **한 판이 끝까지 돌아가는 상태**
`SimplePool` 과 `EnemyConfig` 분리, `WaveDirector` 웨이브 3개, 강화 6종과 3택 UI, 관통과 처치 폭발, 보스 Slam, Clear 와 재시작을 연결했습니다.

Day 3 **제출 가능한 상태**
대시, 보스 Charge 와 HP 임계 소환, HUD 와 Start / Pause / Result 화면, 트레이서 · 폭발 · Shake 와 합성 SFX, 씬 자동 구성(`OverdriveSceneSetup`), 자동 플레이 검증 · 녹화 봇, Windows 빌드와 영상 인코딩, README 를 마쳤습니다.

직접 작성한 것과 AI 가 수행한 것

구분	내용
직접 타이핑 · 디버깅	Day 1~2 핵심 시스템 — 카메라 기준 이동, 3인칭 카메라, Raycast 사격과 관통, 적 AI 와 Golden Angle 배치, Object Pool, 웨이브 진행, 강화 6종, 보스 Slam (AI 의 설명과 리뷰 참고)
AI 작성 → 본인 검토 · 통합 · 테스트	마감 단계 — 대시, 보스 Charge 와 HP 임계 소환, HUD 와 Start / Pause / Result 화면, VFX 와 합성 SFX, `OverdriveSceneSetup`, 자동 플레이 봇, `OverdriveBuild`, README 초안
AI 공통 활용	개념 설명, Unity API 확인, 코드 리뷰, 오류 원인 추적

한계와 확장 계획

전투 핵심과 시스템 연결을 우선해 콘텐츠 수를 의도적으로 제한했습니다. 맵 1개, 무기 1개, 일반 적 행동 1종, 보스 1개(패턴 2), 강화 6종이며, 애니메이션 대신 기하학형 비주얼을 쓰고 카메라 충돌 처리와 영구 성장은 없습니다. 영상은 봇이 플레이한 실제 빌드 화면이며, 봇이 플레이어 실력을 대신하지는 않습니다. 확장 시에는 무기별 공격 방식 → 원거리 적 → 스테이지 분기 → 추가 보스 → 영구 해금 순서로 진행할 계획입니다.

자료 구성

소스코드 `OVERDRIVE_소스코드.zip` — Assets · ProjectSettings · Packages, Unity 6000.5.10f1 에서 바로 열립니다.
실행 파일 `OVERDRIVE_Windows_빌드.zip` — 압축 해제 후 `Overdrive.exe`, 별도 설치가 없습니다.
자동 플레이 영상 `OVERDRIVE_플레이영상.mp4` — 1920×1080 · H.264 · 74초 · **문서** 이 PDF 와 `README.md`