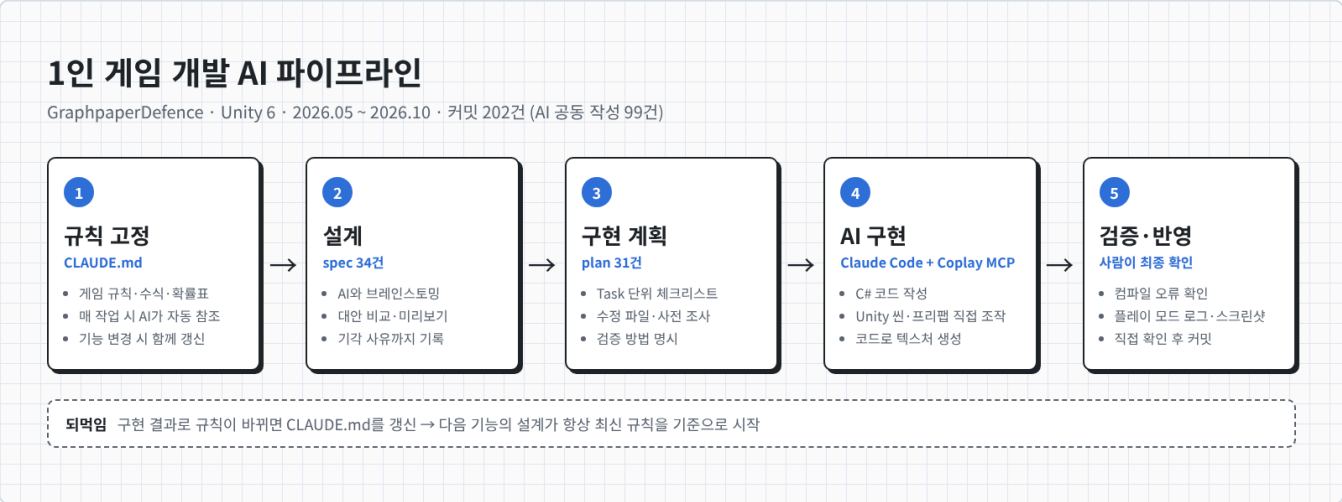
1인 게임 개발 AI 파이프라인 상세 자료

AI 활용 사례 공모전 · 업무 생산성 개선 분야 · GraphpaperDefence (Unity 6)

202	99	65	90
커밋 (2026.05.22~10.01)	AI 공동 작성 커밋	설계 34 + 계획 31 문서	C# 스크립트 (에디터 29)



1. 게임 개요

화면 중앙에 고정된 플레이어가 사방에서 오는 적을 카드를 직접 사용해 공격하는 디펜스 로그라이트입니다. 라운드 종료 후 증강 1개를 고르고, 준비 단계에서 카드를 구매·제거·조합한 뒤 다음 라운드로 넘어갑니다.

구성	내용
카드	기본 6종(연사·분사·저격·유도탄·폭발탄·튕기기) + 조합 전용 5종
증강	26종, 4개 범주(스탯·트리거·카드 시너지·유틸리티), 4개 등급
적 / 캐릭터 / 난이도	적 4종 · 캐릭터 3종(캐릭터별 전용 덱) · 난이도 4단계

2. 단계별 활용 방법

① 규칙 고정: CLAUDE.md

Claude Code가 작업을 시작할 때마다 자동으로 읽는 파일입니다. 사람용 기획서와 AI용 지침을 하나로 통일했고, 수식과 확률표를 그대로 적어 AI가 수치를 임의로 정하지 못하게 했습니다. 아래는 실제 파일 일부입니다.

```
Round System
One round = base 30 seconds + 4 seconds per round (Round 1 = 30s, Round 2 = 34s, ...)
Enemy stats and spawn count scale up every round
Scaling formula (Enemy.ApplyScaling(round), applied per enemy on spawn):
  hp *= 1.1^(round-1) * DifficultyManager.Instance.Multiplier
  attackDamage *= 1.05^(round-1) * DifficultyManager.Instance.Multiplier
Max concurrent enemy count (EnemySpawner.SpawnEnemy()): RoundToInt((4 + round) * DifficultyManager.Instance.Multiplie
r)
---
Card System
Player attacks by manually clicking cards in the hand (no auto-attack)
At round start: draw 4 cards from deck into hand (fewer if the deck has fewer than 4 cards)
When the hand is empty (every drawn card used): hand cooldown (base 2s) starts, then draw again (infinite, no per-roun
d limit)
  The trigger is "hand empty", not "4 cards used" — so thin decks (1-3 cards) still cycle (CardManager.UseCard / IsHa
ndEmpty)
Deck size: minimum 1 card (CardManager.MinDeckSize; RemoveCard refuses and the remove button disables at 1), maximum C
ardManager.deckCapacity (default 12)
Deck exhausted: shuffle discard pile back into draw pile and continue
Cards always visible at bottom of screen (HandPanel)
Card use is blocked when Time.timeScale == 0 (during level-up, augment select, prep phase)
```

② 설계 문서 (spec) 예시: 적 사망 파티클

브레인스토밍 결과와 함께 기각한 대안, 그 이유까지 기록합니다.

```
## 결정 사항

브레인스토밍에서 확정된 것:

1. **흩어지는 느낌: B(부드러운 확산)** — 브라우저 미리보기로 3안(방사형 파편 / 부드러운 확산 / 날카로운 비산)을 비교해 선택. 8개가 느리게 퍼지다 금방 멈추고, 서서히 작아지며 사라진다
2. **발생 시점: 적이 사라지는 모든 경우** — 체력 0 사망뿐 아니라 플레이어에게 닿아 자폭하는 경우에도 파티클을 낸다
3. **구현 방식: ParticleSystem 프리팹 3종(도형별)** — Enemy 프리팹에 인스펙터로 물려두는 방식. 스크립트 기반 스프라이트 파티클(DamageText 패턴)과 프리팹 1종 + 런타임 머티리얼 교체 안은 기각
   - 기각 근거: 스크립트 방식은 사망 1회당 GameObject 8개라 라운드 후반 대량 사망 시 부담. 런타임 머티리얼 교체는 인스턴스 누수 관리가 필요
   - 채택 근거: 수치 튜닝을 코드 수정 없이 인스펙터에서 할 수 있고, 이어질 이펙트 작업의 기반이 된다
```

③ 구현 계획 (plan) 예시

설계를 Task 단위로 나누고, 사용할 도구와 검증 방법을 명시합니다. Unity는 자동 테스트가 어려워 "컴파일 확인 + 플레이 모드 로그 + 스크린샷"을 검증 절차로 정했습니다.

```
# 적 사망 파티클 Implementation Plan

> **For agentic workers:** REQUIRED SUB-SKILL: Use superpowers:subagent-driven-development (recommended) or superpowers:executing-plans to implement this plan task-by-task. Steps use checkbox (`- [ ]`) syntax for tracking.

**Goal:** 적이 사라질 때(체력 0 사망 + 플레이어 자폭 모두) 자기 도형의 아웃라인 파티클 8개가 부드럽게 흩어지며 사라지는 연출을 추가한다.

**Architecture:** 도형별 `ParticleSystem` 프리팹 3종(원/삼각/사각)을 만들고 Enemy 프리팹에 인스펙터로 물려둔다. 적이 사라질 때 자기 위치에 프리팹을 하나 생성하고, 자기 `SpriteRenderer.color`를 파티클의 `main.startColor`로 넘겨 색을 맞춘다(지그재그 보라). 파티클은 `Stop Action = Destroy`로 스스로 정리된다.

**Tech Stack:** Unity 6000.3.19f1, URP 2D, C#, Coplay MCP(`execute_script`, `check_compile_errors`, `play_game`/`stop_game`, `capture_scene_object`). Unity 프로젝트라 pytest 스타일 자동 테스트가 없으므로 "테스트"는 `check_compile_errors`(컴파일 통과) + Play 모드 `execute_script` 런타임 로그 assert + 스크린샷 시각 검증으로 대체한다.

**Design doc:** `docs/superpowers/specs/2026-09-03-enemy-death-particles-design.md`
```

④ AI 구현: Claude Code + Coplay MCP

Coplay MCP는 Unity 에디터를 AI가 직접 조작할 수 있게 해 주는 플러그인입니다. 코드 작성 후 오브젝트 생성, 컴포넌트 연결, 프리팹 제작, 컴파일 오류 확인, 플레이 실행, 스크린샷 확인까지 AI가 수행합니다. 사람은 결과를 확인하고 커밋 여부를 결정합니다.

⑤ 코드로 그리는 이펙트 텍스처

이미지 생성 AI 대신, 픽셀 값을 수식으로 계산하는 에디터 스크립트를 AI와 함께 작성했습니다. 스타일이 일정하고, 값만 바꿔 다시 생성할 수 있으며, 학습 데이터 저작권 문제가 없습니다.

```
        var tex = new Texture2D(Size, Size, TextureFormat.RGBA32, false);
        float c = (Size - 1) * 0.5f;

        for (int y = 0; y < Size; y++)
        {
            for (int x = 0; x < Size; x++)
            {
                float dx = x - c, dy = y - c;
                float d  = Mathf.Sqrt(dx * dx + dy * dy) / c;   // 0 = 중심, 1 = 가장자리

                float a;
                if (d > 1f)
                {
                    a = 0f;
                }
                else
                {
                    a  = Mathf.Pow(d, 2.5f);                                        // 중심은 옅고 가장자리는 진하게
                    a *= Mathf.SmoothStep(1f, 0f, Mathf.InverseLerp(0.94f, 1f, d));  // 테두리 계단 제거
                }

                tex.SetPixel(x, y, new Color(1f, 1f, 1f, a));
            }
```

GraphpaperDefence 에셋 모음

코드로 생성한 이펙트 텍스처

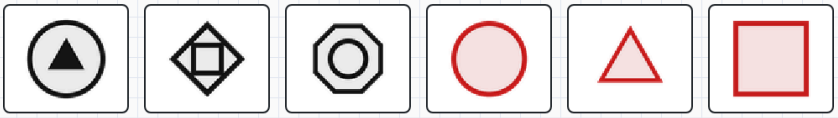
AI와 함께 작성한 Unity 에디터 스크립트가 수식으로 픽셀을 계산해 PNG로 저장 (GenerateCounterWaveSprite.cs, BuildStatusParticles.cs)



충격파 (반격 프로그램) | 상태이상 점 | 상태이상 링 | 둔화 사각형

캐릭터 · 적 스프라이트

모눈종이 배경에 맞춘 단순 도형 스타일



플레이어 | 캐릭터 2 | 캐릭터 3 | 적 Basic | 적 Fast | 적 Tank

카드 아이콘 (기본 카드 6종)



연사 | 분사 | 저격 | 유도탄 | 폭발탄 | 튕기기

3. 결과

항목	결과
개발 기간·규모	약 4개월 반, 커밋 202건(AI 공동 작성 99건), C# 90개 약 8,900줄
문서화	설계 34건 + 구현 계획 31건, 약 24,000줄이 개발 기록으로 남음
반복 작업 자동화	UI 배선, 스프라이트 설정, 이펙트 프리팹 생성 등을 에디터 스크립트 29개로 처리
기획 변경 대응	카드 등급 시스템을 도입(8월) 후 제거(9월)할 때 설계 문서 기준으로 영향 범위를 정리해 안전하게 철거
체감 효과	[본인 체감 수치 기입]

4. 따라 하는 방법

1. 프로젝트 루트에 CLAUDE.md를 만들고 게임 규칙, 수식, 폴더 구조, 작업 규칙(예: 요청 없이 커밋 금지)을 적는다.
2. 새 기능은 AI와 브레인스토밍해 설계 문서로 확정한다. 기각한 대안과 이유도 적는다.
3. 설계를 Task 체크리스트 형태의 구현 계획으로 바꾼다. 각 Task에 검증 방법을 넣는다.
4. Unity에 Coplay MCP를 설치해 AI가 에디터를 직접 조작하게 하고, 계획대로 구현시킨다.
5. 결과를 직접 플레이해 확인한 뒤 커밋하고, 규칙이 바뀌었다면 CLAUDE.md를 갱신한다.

5. 윤리·책임성

대상	출처·라이선스	처리
UI Pack 2.0	Kenney, CC0	상업 이용 가능, 크레딧 표기
RPG 아이콘(증강 아이콘 일부)	Lorc, [라이선스 확인]	[크레딧 표기 방식]
배경음악 Pixel Run	[출처·라이선스]	[처리 방식]
효과음 3종	[생성 서비스·요금제]	[상업 이용 조건 확인]
이펙트 텍스처	직접 작성한 에디터 스크립트로 생성	외부 저작물 미사용

게임 콘셉트, 시스템 규칙, 수치 밸런스, 설계 단계의 최종 선택은 본인이 결정했고, AI는 대안 제시·계획 작성·코드 작성·에디터 작업을 맡았습니다. 모든 결과는 직접 플레이로 확인한 뒤 반영했습니다. 게임은 개인정보를 수집하지 않습니다.