PROJECT WORM KNIGHT · 기술문서

# 몸통은 경로를 따라 움직이고, 잘린 조각은 물리 체인으로 남습니다.

컨보이 이동, 절단과 재결합, 세그먼트 무기 확장을 코드와 플레이 화면으로 설명합니다.

| | |
|---|---|
| 개발자 | **전찬우** |
| 프로젝트 | **3D 디펜스 · Unity 6** |
| 기간 | **17일 팀 프로젝트** |
| 직접 범위 | **팀장 · 코어 설계 · 컨보이 · 통합** |

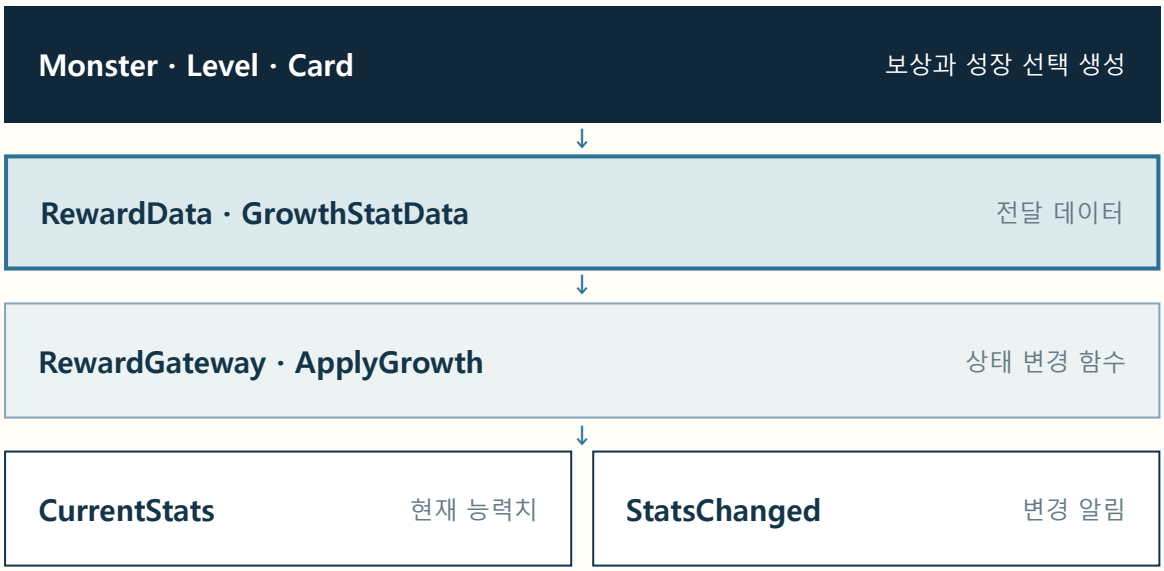
00 공용 계약·코어 상태

# 보상과 성장 선택을 CoreStatProvider에 반영합니다

몬스터는 RewardData를, 레벨업은 GrowthStatData를 전달합니다. **CoreStatProvider**가 현재 런의 상태를 갱신하고 컨보이·세그먼트·HUD가 새 값을 읽습니다.

DATA FLOW

## 데이터 전달 순서




CoreStatProvider가 누적 순서와 변경 알림을 관리합니다.

EXECUTION

## 한 번의 적용 순서

| | |
|---|---|
| **보상·성장 선택** | 플레이 결과를 값으로 만듭니다. |
| **입력 검사** | 비어 있거나 적용할 수 없는 값을 거릅니다. |

| | |
|---|---|
| **코어 전달** | Gateway 또는 ApplyGrowth를 호출합니다. |
| **현재 상태 갱신** | 재화·능력치·세그먼트를 반영합니다. |
| **변경 알림** | StatsChanged를 한 번 발행합니다. |
| **게임 반영** | 전투 계산과 화면이 새 상태를 읽습니다. |

상태 변경 코드를 CoreStatProvider에 모아 모든 기능에 같은 적용 순서를 사용했습니다.

문제

### 직접 참조가 늘어났습니다

몬스터·레벨·UI가 서로의 컴포넌트를 찾으면 한 기능을 수정할 때 다른 시스템의 코드까지 확인해야 했습니다.

선택

### 전달 데이터와 상태 변경 코드를 분리했습니다

RewardData와 GrowthStatData가 값을 전달하고, RewardGateway와 CoreStatProvider가 현재 상태를 바꿉니다.

결과

### CurrentStats를 기준으로 계산합니다

보상·성장·세그먼트 강화·HUD가 CurrentStats를 읽고 StatsChanged에 맞춰 갱신됩니다.

RewardData와 GrowthStatData는 입력값을 전달하고, CoreStatProvider는 현재 런 상태를 보관합니다.

## 몬스터는 보상값만 넘깁니다

RewardData는 경험치·골드·다이아·발생 위치를 담습니다. RewardGateway가 값의 유효성을 확인한 뒤 CoreStatProvider에 전달합니다.

## 성장 선택은 ApplyGrowth에서 반영합니다

ApplyGrowth는 공용 능력치, 세그먼트 추가, 세그먼트 강화를 순서대로 적용합니다. 처리가 끝난 뒤 StatsChanged를 한 번 발행합니다.

직접 구현

### 계약·진입점·현재 상태

RewardData, RewardGateway, CoreStatProvider의 적용 순서와 변경 알림을 연결했습니다.

확장 지점

### 새 성장값의 소비자 연결

새 능력치를 추가할 때 계약 필드, 코어 적용, 실제 전투 계산과 HUD 표시를 함께 연결합니다.

## 관련 코드

**보상 전달값을 작은 계약으로 고정** `RewardData.cs` +

**보상 계약을 코어 상태로 전달** `RewardGateway.cs` +

**성장 선택을 코어 상태에 한 번 반영** `CoreStatProvider.cs` +

01 컨보이 이동 · 경로와 자동 궤도

# 머리가 지나온 거리를 기록해 긴 몸통의 간격을 지킵니다

각 세그먼트는 머리 뒤의 **누적 거리**에서 자신의 위치를 찾고 앞뒤 소켓 방향에 맞춰 정렬됩니다.

이동 계산

## 몸통은 머리 뒤의 누적 거리에서 위치를 찾습니다




머리 위치를 기록하고 각 세그먼트를 누적 거리와 소켓 방향에 맞춰 배치합니다.

설계

## 문제와 해결 방식

| 문제 | 적용 방식 | 다른 방법 |
| --- | --- | --- |

### 속도에 따라 간격이 흔들렸습니다

프레임 위치를 순서대로 따라가면 속도 변화와 프레임 간격에 따라 몸통이 늘어나거나 겹쳤습니다.

### 거리로 경로를 기록했습니다

머리의 이동 거리가 기준을 넘을 때만 샘플을 남기고, 각 세그먼트는 자신의 누적 거리에서 위치를 찾습니다.

### Transform 연쇄 보간

긴 체인에서 오차가 누적되고, 큰 위치 보정이 뒤쪽까지 그대로 전파돼 사용하지 않았습니다.

---

**처리 흐름**

01 **조향 입력** — 수동·자동 방향 ›
02 **머리 이동** — 속도·넉백 ›
03 **경로 기록** — 최소 거리 검사
04 **빈 구간 복원** — 제한된 직선 샘플 ›
05 **몸통 배치** — 누적 거리·소켓 ›
06 **궤도 갱신** — 체인 길이 반영

**자동 궤도** 마지막 세그먼트까지의 경로 길이와 넥서스 Collider 반경을 비교해 필요한 회전 공간을 구합니다.

구현

## 코드에서 처리한 내용

- 정상 이동은 최소 간격을 넘긴 위치만 경로에 추가합니다.
- 넉백이나 위치 보정이 만든 큰 간격은 제한된 수의 중간 샘플로 채웁니다.
- 몸통 길이를 원둘레로 환산하고 넥서스 반경과 조작 여유를 더합니다.
- 길이 변화가 임계값을 넘으면 현재 위치에서 새 자동 궤도로 다시 진입합니다.

각 세그먼트는 머리에서 떨어진 누적 거리에 해당하는 경로 위치를 읽습니다.

### 큰 위치 차이를 중간 샘플로 채웁니다

비정상 간격을 한 점으로 남기면 꼬리가 그 지점을 통과할 때 순간이동합니다. RepairedPathSampleStep 간격으로 중간 지점을 만들고 MaxRepairedPathSamples로 한 번의 작업량을 제한했습니다.

### 몸통이 길어지면 궤도도 넓어집니다

체인 길이를 원둘레로 환산한 반경, 넥서스 충돌체 반경, 최소 조작 반경 중 가장 큰 값을 사용합니다.

보정 범위

### 직선 샘플로 빈 구간을 채웁니다

직선 보정은 꼬리의 순간이동을 막는 용도입니다. 이동 직전의 곡선 정보는 남기지 않습니다.

성능 점검 항목

### 긴 체인의 경로 조회 비용

세그먼트 수가 늘어나는 구간에서는 경로 검색 횟수와 오래된 샘플 정리 비용을 계측 대상으로 둡니다.

## 관련 코드

**큰 위치 차이를 연속 경로로 복원** — `ConvoyController.Path.cs` +

**몸통 길이에 맞춰 자동 궤도 반경 계산** — `ConvoyController.AutoOrbit.cs` +

02 절단·물리·재결합

# 잘린 몸통을 물리 체인으로 바꾸고 플레이 안에서 다시 연결합니다

자기 몸과 충돌하면 충돌 지점 뒤의 세그먼트가 필드에 남습니다. 분리된 조각은 Rigidbody와 ConfigurableJoint로 움직이고, 안정된 뒤에만 회수 영역을 엽니다.

상태 전환

### 부착·분리 상태마다 사용할 컴포넌트를 바꿉니다

| | |
|---|---|
| **ATTACHED** | 경로 추종과 전투 |
| **DETACHED** | Rigidbody·Joint 전환 |
| **SETTLING** | 선형·각속도 검사 |
| **REJOIN READY** | 회수 영역 표시 |
| **ATTACHED** | 런타임 체인 복구 |

안착 조건을 통과한 조각만 현재 꼬리 끝에 다시 붙습니다.

설계

## 문제와 해결 방식

| 문제 | 적용 방식 | 다른 방법 |
|---|---|---|

### 경로 추종 중인 몸통을 물리로 넘겨야 했습니다

붙어 있는 몸통은 경로를 따르고, 잘린 몸통은 필드에서 충돌하며 굴러야 했습니다.

### 절단 인덱스로 목록과 컴포넌트를 전환했습니다

충돌 지점 뒤를 부착 목록에서 분리하고 Rigidbody와 Joint를 추가합니다. 안착 뒤에는 물리 컴포넌트를 제거하고 런타임 상태를 복구합니다.

### 잘린 조각 즉시 삭제

즉시 삭제하면 필드에 남은 조각과 전력을 되찾는 회수 플레이가 사라집니다.

---

처리 흐름

| | | |
|---|---|---|
| 01 **자기 충돌** 절단 인덱스 계산 | 02 **목록 분리** 부착·런타임 정리 | 03 **물리 준비** Rigidbody |
| 04 **체인 연결** ConfigurableJoint | 05 **안착 판정** 속도·대기 시간 | 06 **재결합** 꼬리 끝에 복구 |

| | |
|---|---|
| **회수 조건** | 최소 분리 시간과 안착 시간을 모두 채우고, 현재 부착된 꼬리 끝이 회수 영역에 들어와야 합니다. |

---

구현

## 코드에서 처리한 내용

- 스타터 구간을 보호하고 충돌 지점부터 마지막 세그먼트까지 한 그룹으로 옮깁니다.
- 앞뒤 소켓을 조인트 앵커로 사용해 조각 사이의 연결 길이를 유지합니다.

- 이동 축을 잠그고 회전 축을 제한해 체인 형태로 무너지게 합니다.
- 재결합할 때 조인트와 물리 컴포넌트를 정리하고 세그먼트 런타임을 다시 붙입니다.

절단 순간 경로 추종을 끝내고, 잘린 조각을 물리 상태로 전환합니다.

### 절단 범위를 한 번에 정리합니다

부착 목록, 지면 검사 목록, 세그먼트 런타임이 같은 인덱스 범위를 제거합니다. 남은 체인의 개수 변경도 마지막에 한 번 알립니다.

### 소켓을 물리 연결점으로 사용합니다

FrontSocket과 RearSocket을 ConfigurableJoint의 앵커로 연결합니다. 인접 조각 충돌은 끄고 Projection으로 큰 이탈을 보정합니다.

### 안착 뒤에만 회수 영역을 엽니다

속도가 기준을 넘으면 안착 시간을 초기화합니다. 최소 시간과 안정 조건을 채운 조각만 회수할 수 있습니다.

---

현재 비용

## 분리 수만큼 물리 컴포넌트가 생깁니다

절단된 세그먼트마다 Rigidbody와 Joint를 준비하고 재결합할 때 제거합니다.

풀링 후보

## 반복 절단 시 생성 비용을 줄입니다

긴 꼬리가 자주 잘리는 전투에서는 분리 그룹과 물리 컴포넌트를 재사용할 수 있습니다.

---

## 관련 코드

충돌 지점 뒤의 세그먼트를 분리 그룹으로 전환 ConvoyController.Tail.cs +

분리된 조각을 물리 조인트로 연결 ConvoyController.Tail.cs +

안착과 접근을 통과한 꼬리만 재결합 ConvoyController.Tail.cs +

03 데이터 기반 세그먼트 전투

# 데이터 조합으로 세그먼트 무기를 확장합니다

세그먼트의 정체성, 레벨별 프리팹, 공격 방식을 나눴습니다. 제작자는 Definition과 Profile을 연결하고, 런타임은 같은 생성·피해·쿨타임 계산을 사용합니다.

제작에서 실행까지

## Definition과 Profile로 공격값을 구성합니다

| | |
|---|---|
| **SegmentDefinition** | ID·표시 정보 |
| **LevelDefinition** | 레벨·Prefab |
| **AttackProfile** | 이동·판정·수치 |
| **Catalog** | 선택·생성 등록 |
| **Generic Runtime** | 공격·피해·쿨타임 |

레벨별 외형과 공격 Profile을 묶고 GenericSegmentWeapon에서 실행합니다.

설계

## 문제와 해결 방식

| 문제 | 적용 방식 | 다른 방법 |
|---|---|---|

### 무기마다 계산 순서가 달라졌습니다

공격별 MonoBehaviour가 능력치를 직접 계산하면 런 보너스와 강화가 서로 다른 순서로 적용됐습니다.

### Definition과 Profile을 나눴습니다

레벨별 프리팹과 AttackProfile을 데이터로 연결하고 GenericSegmentWeapon이 최종 공격값을 계산합니다.

### 무기별 완전 독립 구현

고유 동작은 빠르게 만들 수 있지만 공통 강화 규칙을 바꿀 때 모든 무기 코드를 다시 고쳐야 합니다.

| | | |
|---|---|---|
| 01 Segment Definition | 02 Level Definition | 03 Attack Profile |
| 04 Catalog | 05 Generic Runtime | |

구현

## 코드에서 처리한 내용

- 세그먼트 ID와 표시 정보는 상위 Definition이 소유합니다.
- 레벨별 프리팹과 공격 Profile은 SegmentLevelDefinition에서 함께 선택합니다.
- Profile 기준값에 런 보너스, 무기 강화, 세그먼트 강화를 정해진 순서로 적용합니다.
- 데이터로 표현하기 어려운 고유 동작은 별도 partial 구현으로 분리합니다.

GenericSegmentWeapon이 모든 Profile에 같은 피해·쿨타임 적용 순서를 사용합니다.

## 레벨과 공격 Profile을 함께 고릅니다

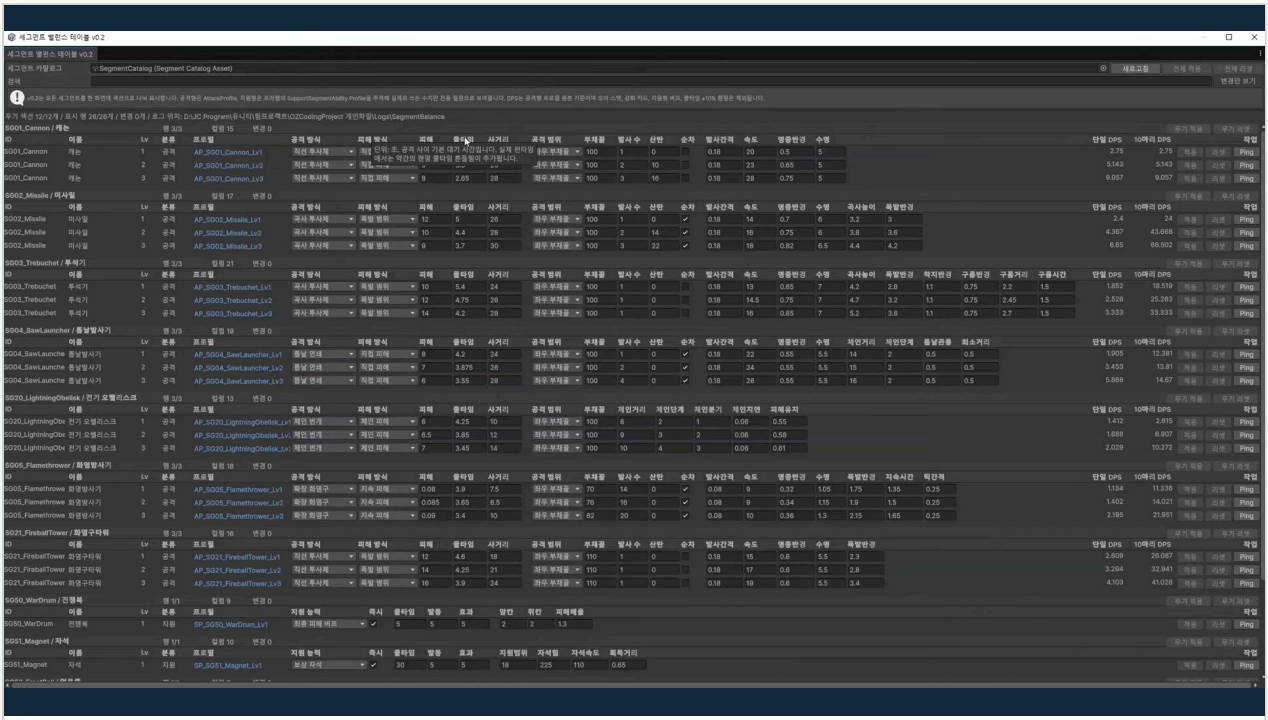
SegmentLevelDefinition은 레벨별 Body·Head Prefab과 SegmentAttackProfile을 묶습니다. 런타임은 현재 레벨에 해당하는 정의를 읽습니다.

## 최종 공격값은 공용 순서로 계산합니다

Profile의 기준 피해에 런 보너스, 공용 기본 피해, 무기 강화, 세그먼트 강화와 지원형 버프를 차례로 적용해 DamageData를 만듭니다.

## DPS와 전투 감각을 함께 봅니다

런타임 DPS 미터는 전체·웨이브·세그먼트별 피해를 나눠 보여줍니다. 사거리, 조준 시간, 과잉 피해와 군중 제어는 같은 전투를 반복해서 함께 조정합니다.



밸런스 도구

### 세그먼트·카드 능력치 표

Profile과 성장값을 한 화면에서 비교하고 수정합니다.

공용 런타임

## Profile로 표현 가능한 공격

독립 구현

## 별도 실행 수명이 필요한 공격

이동 방식, 판정 모양, 피해 단계와 쿨타임을 Profile로 조합하면 GenericSegmentWeapon이 실행합니다.

자체 상태와 종료 조건이 필요한 동작은 partial 구현으로 분리해 공용 계산 코드가 커지지 않게 했습니다.

## 관련 코드

**세그먼트의 레벨별 프리팹과 공격 Profile 연결** `SegmentLevelDefinition.cs` +

**Profile과 코어 능력치로 최종 공격값 계산** `GenericSegmentWeapon.cs` +