

김진우

KIM JINWOO

게임 클라이언트 프로그래머

입력에서 전투까지,
플레이 규칙을 구현합니다.

TECH STACK

Unity · C# / C++ · SFML

주요 구현 경험

01 전투와 행동 상태

Knight Only

공격 판정 · 장비 스탯 · 적·보스 AI

02 C++ 게임플레이

MapleLikeClient

이동·충돌 · 입력 보정 · 레벨 로딩

03 게임 진행과 데이터

Wheel of Fortune

노드 연결 · 룰렛 추천 · 진행 복원

GITHUB

github.com/jinu217

게임 시스템과 서비스 구현 경험

직접 담당한 기능을 프로젝트별로 정리한 상세 포트폴리오입니다.

Knight Only	행동·전투·HashSet·장비·적 AI·튜토리얼	03-13
전기톱 키우기	입력·과열·행 진행·성장·보스 통합	14-17
MapleLikeClient	충돌·전투·카메라·vector·레벨	18-26
Indie Go	참가 입력·캐릭터 선택·UI·오디오·협업	27-30
کم공과 키우기	보스 구조·페이지·탄막 계산	31-32
Wheel of Fortune	맵·그래프·룰렛·진행 복원	33-37
Tishoo	PHP·DB·후기·프로필·공통 화면	38-40

대표 기술 사례 · 클릭하면 해당 페이지로 이동

전투 판정 · 08

여러 Collider를 하나의 피해 대상으로 통합

충돌 처리 · 19

이전 위치와 이동 방향으로 착지 조건 판별

맵 구조 · 35

이동 경로를 인접 리스트로 표현하고 비용 분석

Knight Only · 전투와 성장 시스템

2025.12 시작 | 4인 팀 · 프로그래머 | Unity · C# · New Input System | 2D 액션 로그라이크



담당: 플레이어 이동·전투, BPM/SP, 적·보스 AI, 인벤토리·장비, 튜토리얼
팀 구성: 프로그래머 3명 · 디자이너 1명

[EquipmentManager.cs / 스탯 갱신 >](#)

[PlayerController.cs / 피해 전달 >](#)

핵심 클래스 관계 · 호출 방향



대시·넉백의 실행 우선순위

대시·넉백 중에는 일반 이동을 건너뛰고, 대시가 끝나면 중력과 상태를 복구합니다.

이동 처리

대시나 넉백 중에는 일반 이동이 속도를 덮어쓰지 않도록 반환합니다.

일반 이동 진입 조건

```
1872 if (isDashing || isKnockback) return;
```

피해 처리

대시 중에는 피해 처리를 건너뛵니다. 넉백 중에는 새로운 대시와 공격을 제한합니다.

피해 처리 진입 조건

```
1335 if (isDashing) return;
```

대시 종료

진입 전에 보관한 중력을 복원하고 대시 상태를 해제합니다.

대시 종료 시 복구

```
926 rb.gravityScale = prevGravity;  
927 isDashing = false;
```

하향 점프와 발판 충돌 복구

하향 점프에서 시작 발판의 Collider를 기억하고, 통과한 뒤 다시 충돌하도록 처리합니다.

통과 대상

플레이어가 내려가기 시작한 발판과 몸체 Collider 사이의 충돌만 무시합니다.

복구 조건

플레이어 몸체의 윗부분이 발판의 아랫부분보다 충분히 내려가면 충돌을 다시 겁니다.

착지 처리

하향 점프 종료 시 남아 있는 충돌 무시 상태를 정리합니다.
강화 하강이면 낙하 거리를 사용한 착지 충격도 처리합니다.

[PlayerController.cs / 하향 점프와 충돌 복구 >](#)

발판 위에서 아래로 통과하는 실제 장면

01 발판 위



02 아래로 통과 중



플레이어 `max.y` < 발판 `min.y` - 여유값

```
847 if (playerBounds.max.y < floorBounds.min.y - 0.02f)
    ...
849 Physics2D.IgnoreCollision(bodyCollider, ignoredThinFloorCollider, false);
```

BPM 배율과 SP 소비

BPM은 공격·대시로 변하는 전투 상태값이며, SP는 대시에 소비하고 회복하는 자원입니다.

BPM 변화

고정 효과가 없으면 공격·대시 시 BPM이 증가합니다. 평소에는 자연 감소하며, 명상 중에는 추가로 감소합니다.

성능과 탈진

B·C 구간은 행동 배율을 높입니다. 200을 넘으면 탈진에 진입하고, 정해진 시간이 지나면 BPM을 91로 되돌립니다.

탈진 중에는 경과 시간에 따라 이동과 공격·대시의 허용 여부가 달라집니다. 시간과 배율은 설정값으로 관리합니다.

[PlayerStatus.cs / 구간과 행동 배율 >](#)

BPM 구간

적용 규칙

A · 130 이하

기본 배율

B · 130 초과~170

공격력·대시 속도·거리 배율

C · 170 초과~200

B 효과 + 공격·이동 속도 배율

D · 탈진

초기 이동 정지, 공격·대시 제한

SP 소비와 회복

대시 비용이 부족하면 실행을 거절합니다. 자연 회복과 공격 적중으로 회복하며, C 구간과 명상 상태가 회복량에 영향을 줍니다.

[PlayerStatus.cs / 회복·BPM·대시 >](#)

장비 교체와 스탯 재계산

아이템 보유 상태와 장착 슬롯을 나누고, 장착 결과를 플레이어 스탯에 반영합니다.

장비 교체

장착 중인 아이템을 인벤토리로 돌려보내고 새 아이템으로 슬롯을 갱신합니다. 이후 외형·미리보기·스탯 갱신을 요청합니다.

재계산

baseStats를 복사한 뒤 현재 장착한 아이템의 값을 더합니다. 이전 계산 결과에 장비 효과를 계속 누적하지 않도록 구성했습니다.

변경 알림

계산 결과를 갱신한 뒤 onStatsChanged를 호출해 관련 화면에 변경을 알립니다.

[EquipmentManager.cs / 갱신 요청 >](#)



장착 상태 A



장착 상태 B

[PlayerController.cs / 스탯 재계산 >](#)

피해 대상 통합과 중복 판정

개별 Collider가 아닌 IDamageable 대상의 ID로 한 판정 안의 중복 피해를 구분합니다.

검출 단위와 피해 단위

물리 질의는 Collider 목록을 반환합니다. 각 Collider에서 상위 IDamageable을 찾아 실제 피해 대상을 결정합니다.

중복 판별

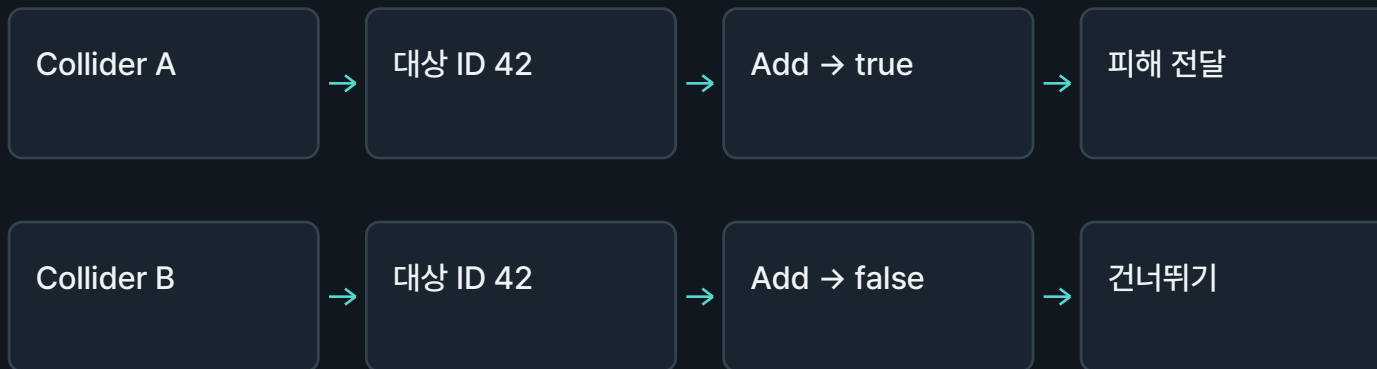
같은 대상 ID를 두 번째로 넣으면 HashSet.Add가 false를 반환합니다. 이 경우 피해 전달을 건너뛵니다.

집합의 범위

집합은 한 판정 안에서만 유지됩니다. 다음 페이지에서 연산 비용과 List를 비교합니다.

[PlayerController.cs / 대상 ID와 피해 전달 >](#)

구조 예시 · 같은 적에 Collider가 2개인 경우



검출 결과는 두 개지만, 이 예시의 피해 대상은 하나입니다.

ID 등록 결과로 분기

```

1194 int id = (target as UnityEngine.Object)?.GetInstanceID() ?? 0;
1195 if (!damagedIds.Add(id)) continue;
  
```

HashSet의 비용과 적용 범위

순서나 부가 데이터가 필요 없는 대상 ID 집합으로, 한 판정 안의 중복 피해를 막습니다.

구현과 구조의 관계

HashSet<int>.Add는 이미 들어 있는 값이면 false를 반환합니다. 피해 대상 ID를 넣는 동시에 중복 여부를 확인하므로 별도의 Contains 호출이 필요 없습니다.

비용이 달라지는 이유

해시로 후보를 좁혀 평균 $O(1)$ 에 추가합니다. 충돌이 물리거나 내부 용량을 늘릴 때는 $O(u)$ 가 들 수 있습니다. List의 ID 검색은 $O(u)$ 입니다.

적용 범위와 비용

판정마다 집합을 새로 만듭니다. 다른 판정의 ID는 유지하지 않으며, 재사용할 때는 초기화 시점을 정해야 합니다.

핵심 코드 발췌

```
1172 HashSet<int> damagedIds = new HashSet<int>();
    ...
1194 int id = (target as UnityEngine.Object)?.GetInstanceID() ?? 0;
1195 if (!damagedIds.Add(id)) continue;
```

연산·공간

비용

Add 한 번

평균·분할상환 $O(1)$, 최악 $O(u)$

m개 ID의 중복 검사

평균 $O(m)$, 공간 $O(u)$

List로 매번 검색

$O(m \times u)$, 최악 $O(m^2)$

m: 검사 ID 수 / u: 서로 다른 ID 수. ID 집합 연산만 분석합니다.

[PlayerController.cs / HashSet 생성·Add >](#)

공격 타이밍과 공간 판정

근접 적의 공격을 선딜 → 한 번의 타격 질의 → 후딜 순서로 실행합니다.



시간 규칙

공격 데이터의 preAttackDelay와 postAttackDelay를 사용합니다. 실제 타격 질의는 선딜이 끝난 시점에 실행합니다.

공간 규칙

방향과 공격 오프셋으로 판정 위치를 구하고, 공격 범위·높이로 박스 크기를 구성합니다. 범위 안의 IDamageable에 피해를 전달합니다.

핵심 코드 발췌

```
25 yield return new WaitForSeconds(data.preAttackDelay);
27 CheckHit();
...
32 yield return new WaitForSeconds(data.postAttackDelay);
```

공격 시간과 공격 범위를 별도 값으로 두어 타격 시점과 공간 판정을 각각 조정합니다.

적의 탐지·추격·공격 조건

탐지 범위·시야·공격 거리·공격 간격을 나눠 적의 행동을 결정합니다.



일반 적과의 전투 화면 · 사용자 제공

탐지



반응



추격



공격

처음 발견했을 때

평소에는 탐지 범위를, 추격 중에는 추격 범위를 사용합니다.
새로 추격을 시작할 때 반응 처리를 호출합니다.

공격할 때

근접 공격 범위에 들어오면 수평 이동을 멈추고, 공격 간격이 지났는지 확인한 뒤 공격을 시작합니다.

피격으로 추격할 때

강제 추격 시간이 남아 있으면 일반 시야 검사와 별도의 경로에서 플레이어 방향과 공격 거리를 판단합니다.

EnemyController.cs / 탐지·추격·공격 >

보스 스킬 실행과 애니메이션 연동

MidBossController가 선택과 실행 순서를 조정하고 BossSkillExecutor가 스킬 동작을 처리합니다.



1스테이지 중간 보스 전투 화면

선택 순서

목록을 앞에서부터 검사해 조건을 만족하는 첫 스킬을 선택합니다.
후보가 없으면 기본 공격을 검토하므로 목록 순서가 우선순위에 영향을 줍니다.

쿨타임 예외

최소 거리와 남은 쿨타임 조건을 만족하면 조기 실행 후보가 됩니다.
페이즈·시전 거리·버프 조건은 별도로 확인합니다.

실행 중 재선택 제한

isAttacking이면 AI의 새 스킬 선택을 건너뛵니다. 스킬 실행과 애니메이션 대기를 마친 뒤 공격 상태를 해제합니다.

튜토리얼 단계와 행동 제한

현재 단계 확인, 기능 허용 여부, 다음 단계 이동을 TutorialManager에서 처리합니다.

단계 전환

이벤트를 받았을 때 예상한 단계와 현재 단계가 같아야 다음 단계로 이동합니다. 다른 단계에서 들어온 알림은 반환합니다.

행동 제한

대시·공격·인벤토리 열기 등은 현재 단계가 허용하는지 확인합니다. 훈련용 표적도 단계에 맞춰 피해 가능 여부를 구분합니다.

화면과 연동

단계가 변경되면 onStepChanged를 호출합니다. 실제 행동의 완료 알림과 안내 화면을 같은 진행 상태에 연결합니다.

핵심 코드 발췌

```
71 public void AdvanceIfCurrent(TutorialStep expectedStep, TutorialStep
    nextStep)
72 {
73     if (!tutorialActive || currentStep != expectedStep)
74         return;
76     SetStep(nextStep);
77 }
...
84 public bool CanDash()
85 {
86     return !tutorialActive || currentStep >=
        TutorialStep.DashUnlocked;
87 }
```

행동 알림



현재 단계 확인

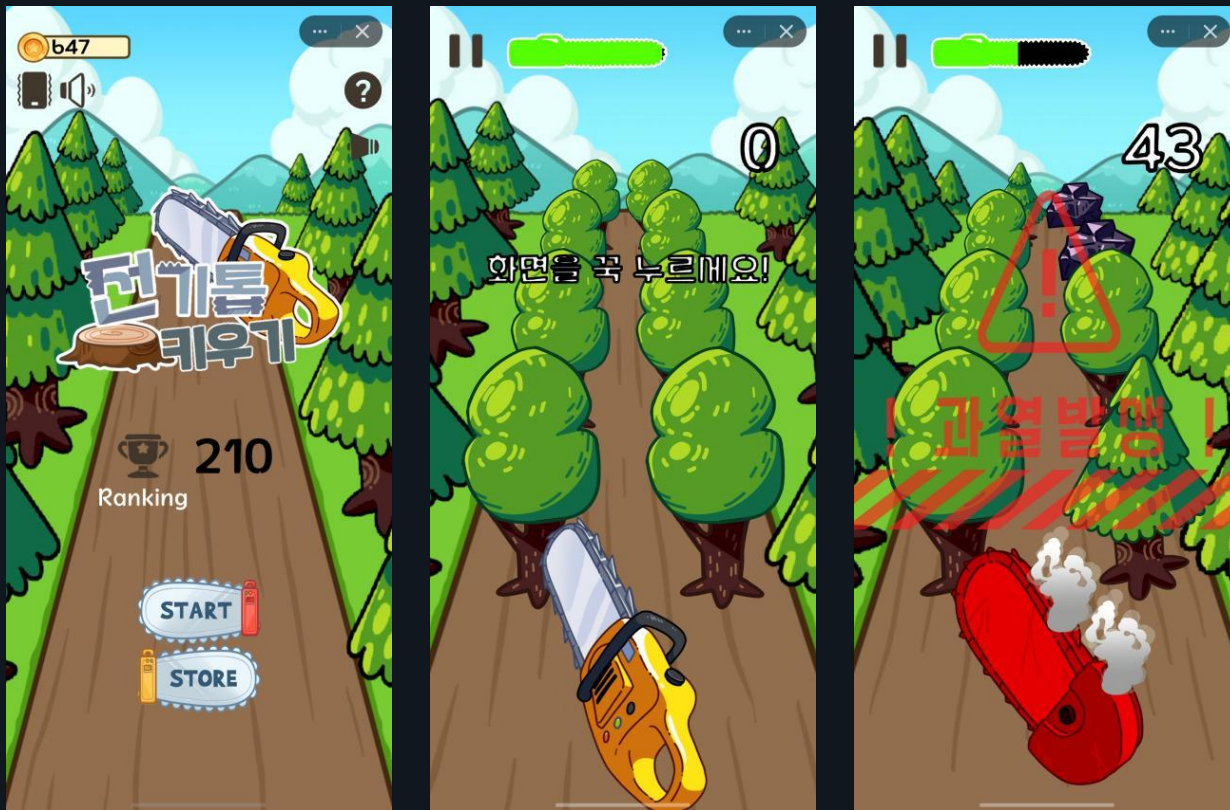


다음 단계 안내

[TutorialManager.cs / 단계·행동 조건 >](#)

벌목과 강화가 반복되는 캐주얼 게임

2025.10-2025.11 | 7인 팀 · 프로그래머 | Unity · C# · WebGL | 2D 성장형 게임



프로젝트

2025.10-2025.11 · 7인 팀

기획 1명, 프로그래머 4명, 디자이너 2명

담당 역할 · 게임 개발

전기톱 동작과 성장 수치 반영, 나무·보스·장애물, 행 진행과 객체
생명주기 처리

플레이 흐름

입력 유지 → 벌목 → 보상 → 강화
공격 성능과 과열 상태를 다음 플레이에 반영합니다.

전기톱 입력·목표·과열 상태

목표 변경과 입력 해제에 맞춰 공격 타이머와 전기톱의 위치를 갱신합니다.



과열

입력 유지 시간에 따라 열을 누적합니다. 한계에 도달하면 과열 상태로 전환하고 입력 상태와 타격 연출을 정리합니다.

목표 전환

가까운 나무가 바뀌면 UI 대상과 첫 타격 처리를 갱신합니다. 목표가 없거나 입력을 놓으면 공격 타이머를 초기화합니다.

핵심 코드 발췌

```
270 overhit += overhitRate * Time.deltaTime;
271 overhit = Mathf.Min(overhit, maxOverhit);
273 if (overhit >= maxOverhit)
274 {
275     isOverhited = true;
276     BackgroundManager.Instance.SetOverheat(true);
277     SetClicked(false);
278     ShowSmoke();
279     if (currTreeHitParticle) currTreeHitParticle.SetActive(false);
280     wasHolding = holding;
281     return;
282 }
```

[ChainSaw_Move.cs / 과열과 목표 전환 >](#)

행 진행과 오브젝트 재사용

행 진행을 제한하는 조건과, 나무를 다시 사용할 때 복구·정리하는 상태를 구분합니다.

행 진행

초기화 전이거나 행 이동 중이면 새 진행 요청을 처리하지 않습니다. 현재 구간을 정리한 뒤 다음 구간을 생성합니다.

풀에서 가져오기



상태 초기화



화면 슬롯 등록

활성화 시 복구

Tree는 등급 데이터를 읽고 체력 바의 색·폭을 복구합니다. 기반 클래스는 체력·생존 상태·원래 크기를 다시 설정합니다.

OnEnable / OnDisable

```
26 currentHealth = Mathf.Max(1f, maxHealth);
27 isAlive = true;
28 transform.localScale = initialScale;
...
33 StopAllCoroutines();
```

비활성화 시 정리

기반 클래스에서 코루틴을 중단하고, Tree에서 몸체와 체력 바의 트윈을 종료합니다.

행 제거 시 쓰러진 나무는 슬롯에서 먼저 빠지고, 자신의 연출 종료 경로에서 반환됩니다.

[Tree.cs / 활성화·비활성화 >](#)

[TreeGenerator.cs / 구간 반환 >](#)

성장 수치와 보스 통합

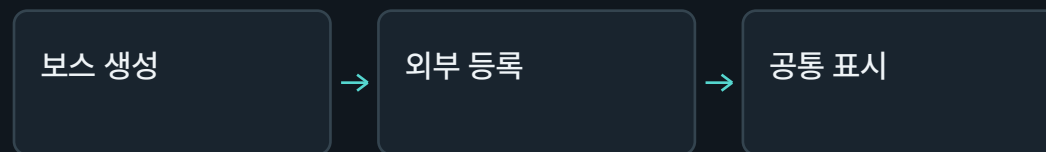
강화 결과는 전기톱의 동작 값으로, 보스는 나무 생성기의 공통 표시 흐름으로 연결합니다.

성장 수치 적용

공격력	기본 피해량 + 피해량 증가분
공격 간격	기본 간격에서 강화 비율만큼 감소
과열 한계	기본 한계에 강화 증가분 반영
값의 범위	공격 간격·과열·반동 값을 제한

보스 통합

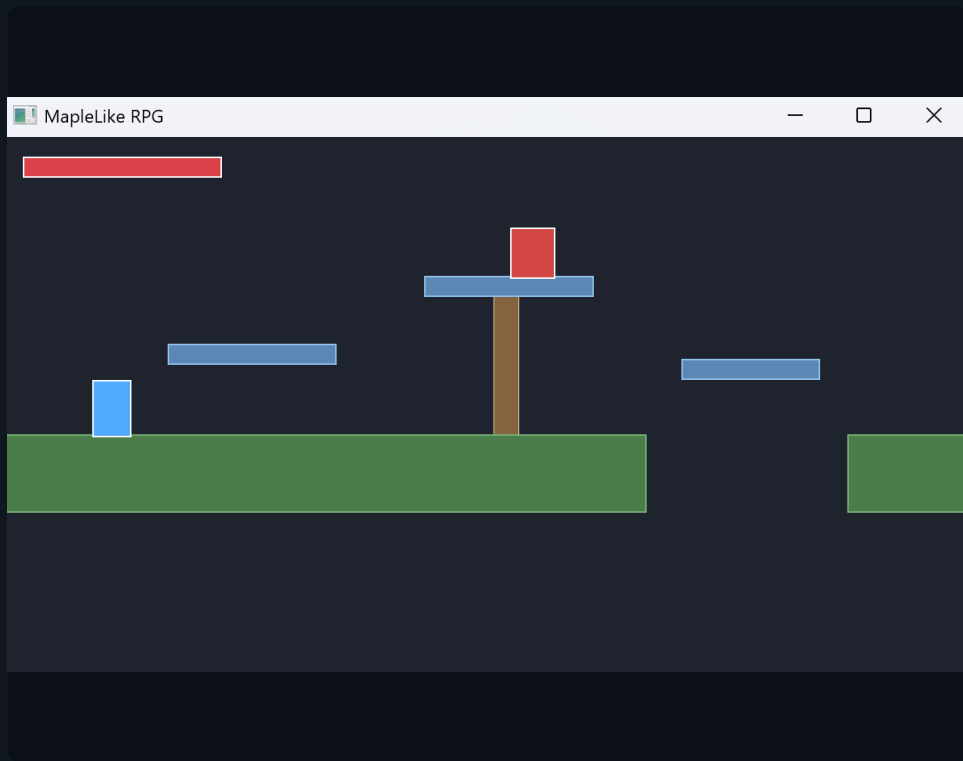
보스를 외부 나무 등록 경로에 넣어 기존 깊이·위치·정렬 규칙을 적용했습니다. 보스가 사용하는 구간은 별도로 기록해 일반 나무와의 배치를 구분합니다.



개별 보스와 일반 나무가 화면을 이동하는 규칙을 같은 생성기에서 처리합니다.

MapleLikeClient · C++ 게임플레이 구현

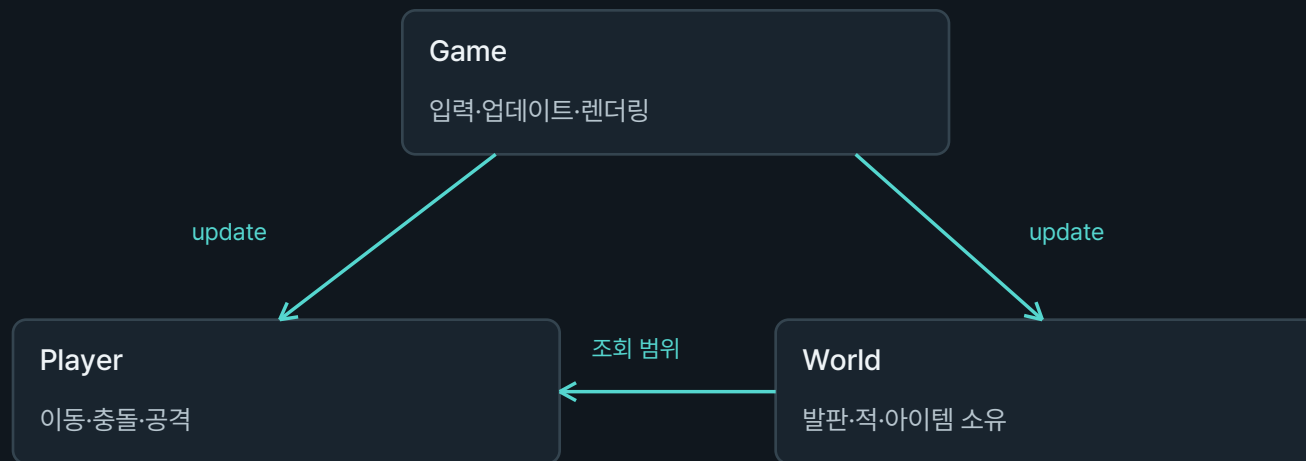
2026.07-2026.08 | 개인 프로젝트 | C++ · SFML · JSON | 2D 액션



이동·충돌·전투·레벨 로딩 직접 구현

[Game.h / 객체 소유 >](#)

핵심 실행 구조 · Game이 객체를 소유하고 갱신



Player 갱신에는 World의 발판·등반 객체를 전달합니다. World 갱신에는 Player와 Inventory를 전달합니다.

경과 시간은 최대 1/60초로 제한하는 정책을 사용합니다.

[Game.cpp / 업데이트 호출 >](#)

단방향 발판의 착지 조건

단방향 발판에서 상승 통과·하강 착지·아래로 내려가기를 처리합니다.

착지 조건

현재 겹침만으로 판정하면 발판 아래에서 올라오는 상황을 구분하기 어렵습니다. 이전 프레임의 발 하단 위치와 수직 속도를 함께 확인합니다.

아래로 내려가기

내려가기 타이머가 남아 있으면 단방향 발판의 착지 처리를 건너뛵니다.

상승 중

통과

위에서 낙하

착지

내려가기 입력

일시 통과

핵심 코드 발췌

```
491 if (platform.getType() == PlatformType::OneWay)
492 {
493     const bool landedFromAbove = velocity.y >= 0.f
494         && previousBottom <= platformPosition.y + 1.f;
495     if (dropThroughTimer > 0.f || !landedFromAbove)
496     {
497         continue;
498     }
499 }
```

이전 하단 위치



발판 위에서 접근



낙하 중 착지

수평 이동과 수직 이동의 충돌 해소를 나누어 처리합니다.

[Player.cpp / 단방향 발판 판정 >](#)

점프 입력 버퍼와 코요테 타임

착지 전 입력과 발판 이탈 후 입력을 타이머로 보정하고, 공중 점프 횟수와 연결합니다.

점프 입력 버퍼

착지 전 입력을 잠시 보관하고, 착지 후 버퍼가 남아 있으면 점프합니다.

코요테 타임

발판 이탈 직후에도 점프를 허용합니다. 지면에서 갱신한 타이머는 공중에서 감소합니다.

경계 조건

코드의 처리

점프 실행

두 타이머를 0으로 만들고 jumpsUsed를 증가시킵니다.

건다가 발판 이탈

유예 시간이 끝나고 사용 횟수가 0이면 1로 바뀌 공중 점프 수를 제한합니다.

등반 진입

접지 상태와 두 타이머를 정리하고 점프 사용 횟수를 0으로 초기화합니다.

[Player.cpp / 점프 조건·실행 >](#)

[Player.cpp / 등반 진입 >](#)

사다리·로프의 진입과 이탈

등반 객체를 찾고 진입 상태를 유지하며, 점프·좌우 입력·끝 지점에서 등반을 해제합니다.

진입

상하 입력이 있고 점프 입력이 없는 상태에서 접촉 가능한 등반 객체를 찾습니다. 진입하면 접지 상태와 점프 타이머를 정리합니다.

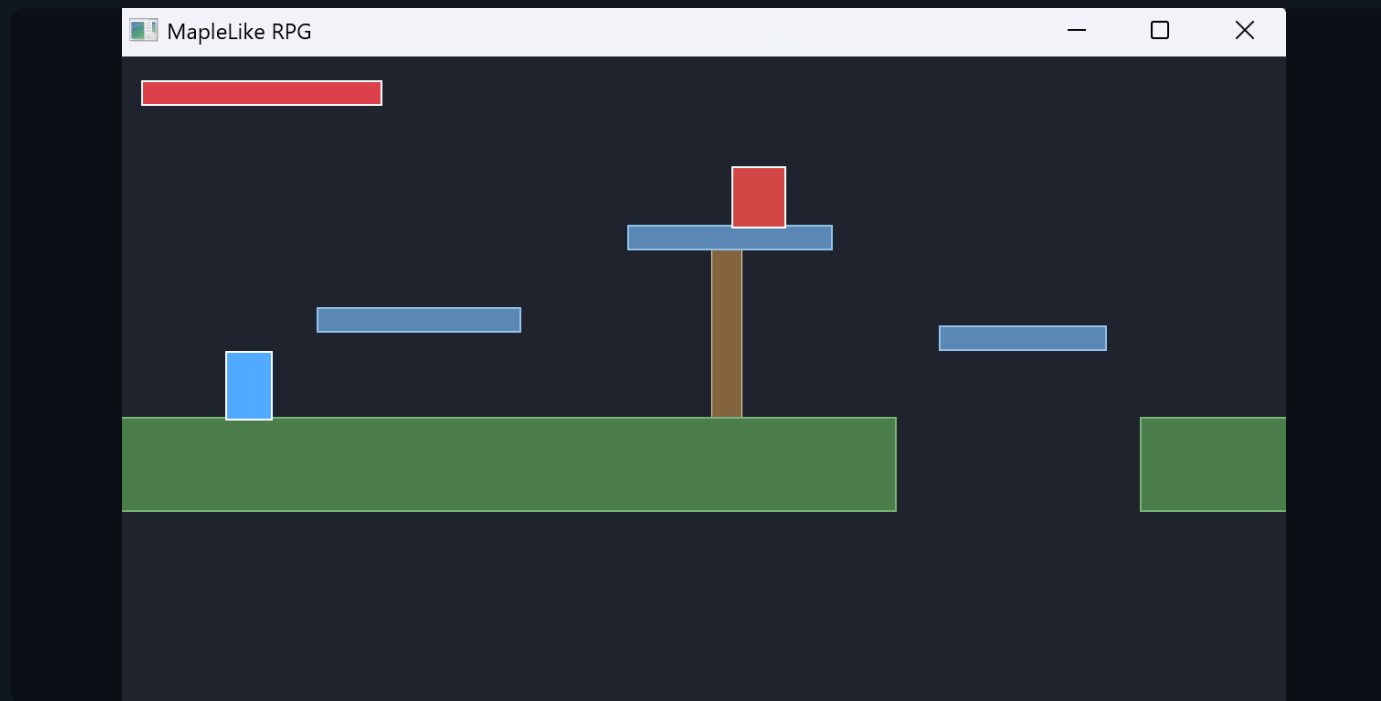
이동

등반 객체의 중심에 수평 위치를 맞추고 수직 속도를 적용합니다. 등반 중 업데이트는 일반 중력·발판 충돌 흐름과 구분합니다.

이탈

점프나 좌우 입력으로 등반을 종료하며, 위·아래 끝에서도 등반 상태와 참조를 해제합니다.

[Player.cpp / updateClimbing.findClimbable](#) >



발판과 등반 객체를 함께 배치한 레벨 화면

공격 ID로 반복 피격 구분

플레이어의 공격 범위와 적이 마지막으로 맞은 공격 ID를 함께 확인합니다.

공격 단위

새 공격이 시작될 때 공격 ID를 증가시킵니다. 적은 마지막으로 처리한 ID를 기록해 공격이 유지되는 여러 프레임을 구분합니다.

피격 이후

피격 시 체력·넉백·점멸 시간을 갱신합니다. World는 처치된 적의 아이템 생성과 제거를 처리합니다.

적용 조건

현재는 한 플레이어의 연속된 공격 ID를 구분합니다. 여러 공격자나 독립된 동시 공격으로 확장한다면 식별 키와 기록 범위를 다시 설계해야 합니다.

Enemy.cpp / 중복 판정·공격 ID 기록

```
46  if (isDead() || attackId == lastHitAttackId ||  
    !overlaps(attackBounds))  
47  {  
48      return false;  
49  }  
51  lastHitAttackId = attackId;  
52  --health;
```

[Enemy.cpp / 공격 ID 판정 >](#)

[World.cpp / 공격·드롭 처리 >](#)

카메라 추적과 화면 비율

카메라 중심은 월드 범위로 제한하고, 창 크기가 바뀌면 뷰포트를 조정합니다.

카메라 추적

플레이어 위치에서 목표 중심을 구하고, 월드 바깥이 보이지 않도록 화면 절반 크기를 고려해 제한합니다.

보간

현재 중심에서 목표 중심까지의 차이에 $1 - \exp(-\text{속도} \times \text{경과 시간})$ 을 곱해 이동량을 계산합니다.

핵심 코드 발췌

```
14 const sf::Vector2f halfView = viewSize / 2.f;
15 const sf::Vector2f targetCenter{
16     std::clamp(targetPosition.x, halfView.x, worldSize.x - halfView.x),
17     std::clamp(targetPosition.y, halfView.y, worldSize.y - halfView.y)
18 };
19 const float followAmount = 1.f - std::exp(-FollowSpeed * deltaTime);
```

창 크기 변경

창과 뷰의 가로세로 비율을 비교합니다. 가로가 넓으면 뷰포트 너비를, 세로가 길면 높이를 조절해 화면 비율을 유지합니다.

창 비율 비교



뷰포트 조절

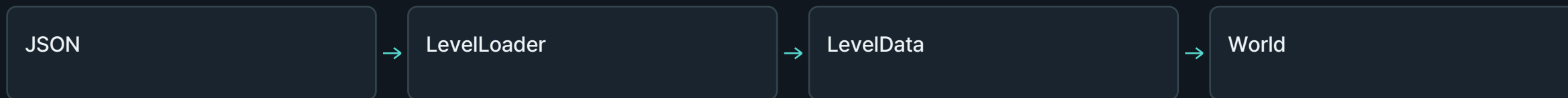


가운데 정렬

[CameraController.cpp](#) / [추적·리사이즈](#) >

레벨 데이터와 월드 객체 생성

JSON을 읽고 검증한 LevelData로 월드 객체를 구성합니다.



생성 흐름

LevelLoader가 파일을 읽어 LevelData를 만듭니다.

World는 전달받은 데이터로 발판·적·체크포인트를 생성합니다.

소유와 조회

환경 객체는 World가 소유합니다. Player에는 발판·등반 객체의 조회 범위를 전달해 충돌과 이동을 계산하게 합니다.

World.cpp / LevelData로 발판·적 생성

```
9 World::World(const LevelData& level)
10     : size(level.worldSize), enemySpawns(level.enemies)
11 {
12     platforms.reserve(level.platforms.size());
13     for (const PlatformData& platform : level.platforms)
14     {
15         platforms.emplace_back(platform.position,
16                                platform.size, platform.type);
17     }
18     resetEnemies();
```

[LevelLoader.cpp / 데이터 검증 >](#)

[World.cpp / 월드 객체 생성 >](#)

vector 소유와 span 조회

연속 저장 구조의 순회·삭제 특성과, 데이터를 소유하지 않는 조회 범위를 구분합니다.

연속 저장과 순회

World는 적·발판 등을 vector에 보관합니다. 원소가 연속해 있어 순회에 적합합니다. 적 N마리의 공격 범위 검사는 전체를 훑으므로 $O(N)$ 입니다.

크기와 용량

reserve로 용량을 확보한 뒤 emplace_back으로 원소를 추가합니다. reserve는 size를 바꾸지 않으며, 끝 추가도 재할당이 생기면 $O(N)$ 입니다.

복사 없는 조회

span<const Platform>으로 발판을 복사 없이 읽습니다. 원본이 소멸하거나 vector가 재할당되면 기존 뷰는 무효가 됩니다.

[World.h / vector 멤버 >](#)

핵심 코드 발췌

```
12 platforms.reserve(level.platforms.size());
13 for (const PlatformData& platform : level.platforms)
14     ...
38 std::span<const Platform> World::getPlatforms() const
39 {
40     return platforms;
41 }
```

vector 연산

비용·주의점

인덱스 접근 / 끝 추가

$O(1)$ / 분할상환 $O(1)$

erase_if로 적 제거

$O(N)$, 남는 원소 이동 가능

연결 리스트와 비교

연속 저장 / 노드별 연결·할당

[World.cpp / 적 일괄 제거 >](#)

레벨 입력 검증과 오류 처리

파일 읽기 → JSON 파싱 → 필드 검증 → LevelData 구성 순서로 처리합니다.

검사 항목	허용하는 입력
좌표·크기 배열	원소가 두 개이며 모두 숫자
월드·객체 크기	x와 y 모두 양수
발판 타입	solid 또는 oneWay
등반 타입	ladder 또는 rope

오류 지점

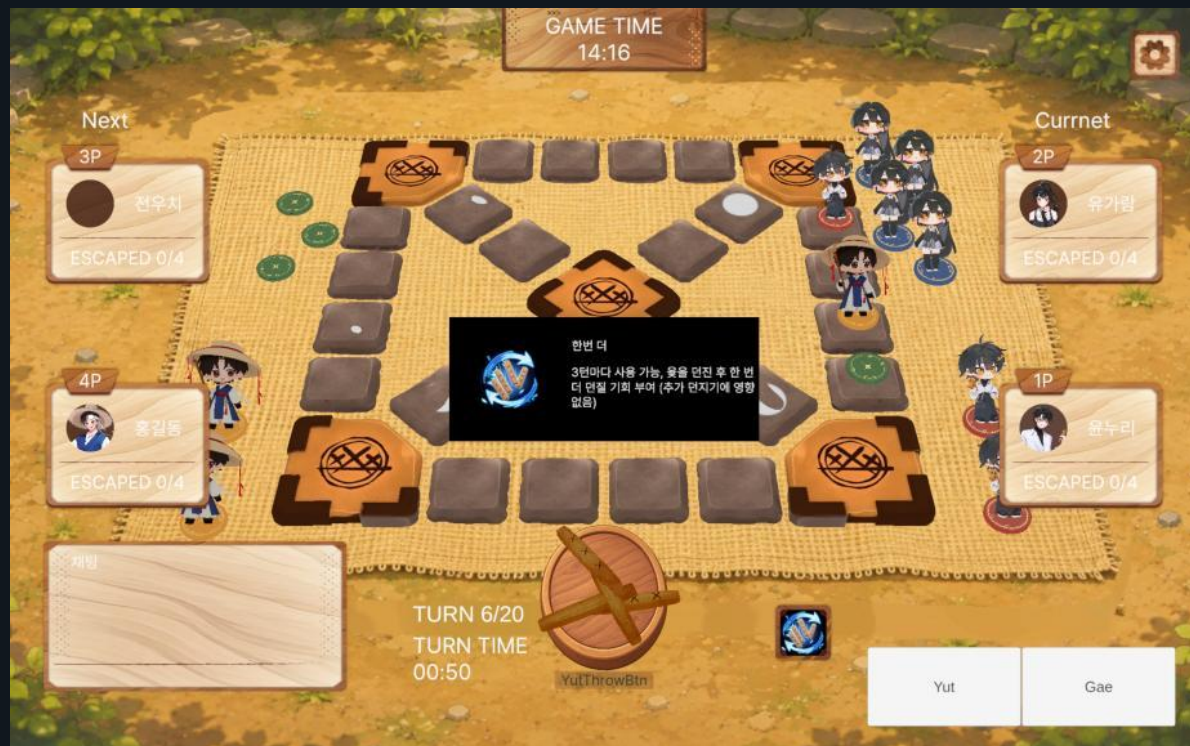
배열 형태나 값이 맞지 않으면 필드 이름을 포함한 예외를 발생시킵니다. 파일을 열 수 없는 경우도 읽기 단계에서 구분합니다.

객체 생성과 분리

검증과 변환은 LevelLoader에서 처리합니다. World는 변환된 LevelData를 받아 발판·적·체크포인트 등을 구성합니다.

로컬 입력과 게임 UI 연동

2026.04 시작 | 8인 팀 · PM / 프로그래머 | Unity · C# | 2.5D 보드게임



입력·선택 흐름

키보드·게임패드 참가 상태를 관리하고, 캐릭터 선택 결과를 다음 씬에 전달했습니다.

이벤트와 UI

턴 시작·단계 변경 이벤트에서 관련 화면을 갱신합니다. OnEnable에서 구독하고 OnDisable에서 해제하며, 타이머 등은 Update에서 갱신합니다.

팀 내 역할

공통 UI·입력·오디오와 화면 전환을 연결하고, PM으로 기능 분담과 통합 과정을 관리했습니다.

플레이어별 참가·선택 상태

키보드와 게임패드의 입력을 플레이어별 커서·선택 상태로 연결합니다.

참가

키보드 참가자를 기준으로 패드 참가와 연결 해제를 확인합니다. 참가 상태는 플레이어 슬롯 단위로 관리합니다.

선택

플레이어마다 커서 위치와 선택 완료 여부를 보관합니다. 확정 전에는 이동·선택, 확정 후에는 취소 입력을 처리합니다.

다음 화면

선택이 끝나면 캐릭터 ID와 CharacterData를 함께 전달합니다. 확정 중에는 중복 시작을 제한합니다.

[CharacterSelectController.cs / 플레이어별 입력 >](#)

[LobbyControllerJoinInput.cs / 참가 >](#)



음량 설정과 채널별 적용

BGM·클릭·토글 소리를 나누고, 사용자 설정을 실제 AudioSource 값으로 반영합니다.



채널	실제 적용 규칙
BGM	전체 또는 BGM 음소거면 0, 그 외 전체 × BGM 음량
클릭	전체 또는 클릭 음소거면 0, 그 외 전체 × 클릭 음량
토글	전체 음소거면 0, 그 외 전체 음량

음량 값은 0~1 범위로 제한합니다. PlayerPrefs에 음량과 음소거 상태를 보관하고, 초기화 시 불러와 적용합니다.

[AudioManager.cs / 설정·채널 음량](#) >

턴 이벤트와 게임 UI 연결

턴 시스템의 알림을 받아 플레이어 패널·현재 턴·스킬 화면을 갱신합니다.



연결 지점

OnTurnStarted와 OnTurnPhaseChanged를 구독합니다. 턴 시작 시 플레이어 패널·강조 표시·턴 정보·현재 스킬을 갱신합니다.

구독 수명

OnEnable에서 구독하고 OnDisable에서 해제합니다. 타이머와 탈출 횟수 표시는 Update에서 별도로 갱신합니다.

이벤트별 갱신 범위

```

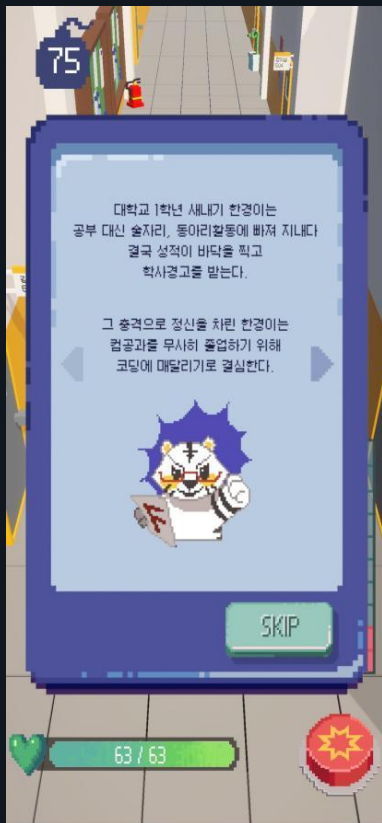
87 private void HandleTurnStarted(PlayerSlot player)
88 {
91     RefreshPlayerPanels();
92     RefreshTurnHighlights(player);
93     RefreshTurnInfo();
94     RefreshCurrentPlayerSkills(player);
95 }
97 private void HandleTurnPhaseChanged(TurnContext turn)
98 {
99     RefreshTurnInfo();
100 }
  
```

[InGameUIManager.cs / 구독·해제 >](#)

[InGameUIManager.cs / 이벤트 처리 >](#)

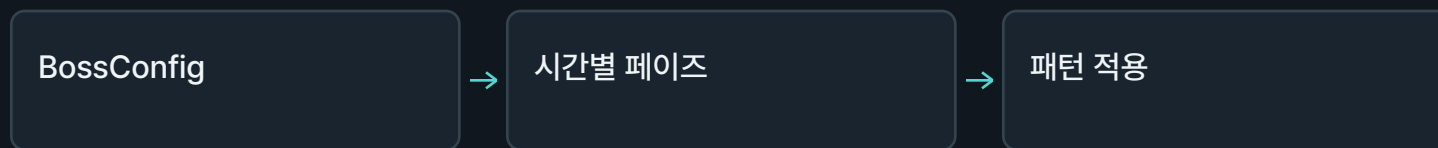
공통 보스 구조와 페이즈 데이터

2025.08-2025.11 | 7인 팀 · PM / 프로그래머 | Unity · C# | 캐주얼 액션



공통 실행

BossBase에서 체력·발사 주기·탄환 생성을 처리하고, 스테이지별 클래스에서 발사 패턴을 구현했습니다.



페이즈 시작 시간, 발사 간격, 탄속, 탄 수를 ScriptableObject에 둡니다. BossManager가 경과 시간에 맞는 페이즈를 적용합니다.

PM 역할: 기능 분담, 통합, 축제 시연용 APK 준비

부채꼴 탄막의 방향·각도 계산

플레이어 방향을 기준으로 직선 연사와 부채꼴 발사를 조합합니다.

부채꼴 계산

탄이 2개 이상이면 전체 각도를 탄 수 - 1로 나눕니다. 그 외에는 간격을 0으로 둡니다. 시작 각도는 -전체 각도 / 2입니다.

방향 적용

플레이어를 향하는 기준 회전에 각 탄의 yaw 회전을 적용하고, 회전된 전방 벡터로 탄을 생성합니다.

패턴 조합

혼합 페이지에서는 발사 횟수를 세어 일정 주기마다 부채꼴을 발사하고, 나머지는 직선 연사로 처리합니다.

발사 간격과 탄별 회전 계산

```
60 float step = count > 1 ? totalAngle / (count - 1) : 0f;
61 float start = -totalAngle * 0.5f;
62 ...
65 float yaw = start + step * i;
66 Quaternion rot = Quaternion.Euler(0f, yaw, 0f) * baseRot;
67 SpawnBullet(rot * Vector3.forward);
```

맵 선택과 전투 후 복귀

2026.08 게임잼 | 4인 팀 · 프로그래머 | Unity · C# | 로그라이크 턴제



프로젝트의 맵 화면

맵 구성

이전 층과 연결 가능한 이벤트 종류를 확인한 뒤 가중치를 적용해 후보를 선택합니다. 노드 사이의 연결도 함께 구성했습니다.

진행 상태

세션에 완료한 이벤트 위치와 플레이어 상태를 보관합니다. 맵으로 돌아오면 완료 상태를 복원하고 다음 노드를 선택할 수 있게 합니다.

선택



전투·이벤트



완료



다음 노드

이벤트 노드의 연결 조건

노드의 다음 경로와 동일 이벤트 연속 횟수를 이용해 연결 가능 여부를 판단합니다.

노드 상태

각 노드에 좌표, 이벤트 종류, 다음 노드 목록을 둡니다. 연결할 때 중복 연결을 확인하고 다음 노드의 연속 횟수를 갱신합니다.

이벤트 종류

같은 종류로 연결

상점·여관

연속 연결 제한

연속 제한

상점·여관은 같은 종류의 연속 연결을 제한합니다. 전투·랜덤 이벤트는 연속 횟수가 두 번에 도달했는지 확인합니다.

전투·랜덤

연속 횟수가 2 미만일 때 허용

연결 구성

아래층에서 가까운 연결 후보를 찾고, 위층에서도 연결 가능한 출발 노드를 찾아 경로를 구성합니다.

이전 노드 상태



허용 후보



가중치 추천



경로 연결

[InGameEventNode.cs / 연결 조건 >](#)

[InGameSceneController.cs / 추천·연결 >](#)

인접 리스트와 연결 검사 비용

각 이벤트를 정점으로, 이동 가능한 다음 이벤트를 방향이 있는 간선으로 표현합니다.

인접 리스트 표현

각 노드의 List에 다음 노드를 저장합니다. 실제 연결만 기록하므로 노드 V 개와 간선 E 개의 연결 구조를 $O(V+E)$ 공간으로 표현합니다.

중복 연결 검사

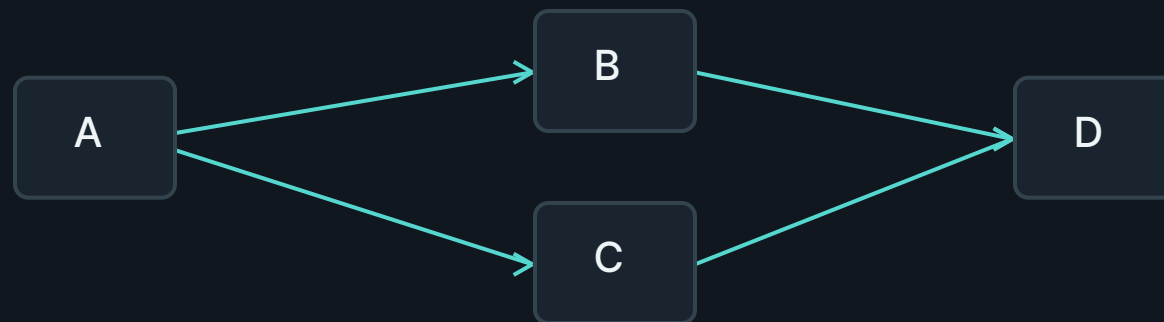
ConnectTo는 List.Contains로 중복을 확인합니다. 다음 노드가 d 개라면 $O(d)$ 이며, 이후 Add는 분할상환 $O(1)$ 입니다. 연결 시도 전체를 $O(1)$ 로 볼 수는 없습니다.

대안과 적용 조건

연결 여부를 자주 검사하고 d 가 커지면 HashSet을 검토할 수 있습니다. 인접 행렬은 조회가 $O(1)$ 이지만 $O(V^2)$ 공간이 필요해 연결 밀도도 함께 봐야 합니다.

[InGameEventNode.cs / nextNodes.ConnectTo](#) >

연결 구조 예시 · A의 nextNodes = [B, C]



$A \rightarrow B \cdot A \rightarrow C \cdot B \rightarrow D \cdot C \rightarrow D$

다음 노드 순회 $O(d)$

중복 여부 Contains $O(d)$

List.Add 분할상환 $O(1)$, 최악 $O(d)$

d : 해당 노드의 다음 노드 수. 맵 생성 전체의 비용은 아닙니다.

룰렛 추첨과 회전 연출

10칸의 효과 구성을 데이터로 관리하며, 추첨과 효과 적용의 호출 경로를 나눴습니다.

구성

효과별 칸 수의 합이 10인지 검사하고 각 칸에 효과 데이터를 배치합니다. 칸을 균등하게 추첨하므로 같은 효과의 칸 수가 확률에 반영됩니다.

회전 연출

회전을 시작할 때 결과 칸을 선택하고 목표 각도를 계산합니다. 곡선에 따라 회전한 뒤 결과 이벤트와 완료 콜백을 전달합니다.

호출 방식

결과만 얻는 경로와 효과를 바로 적용하는 경로를 구분하고, 회전 중의 재시작을 제한합니다.

[RouletteController.cs / 결과-연출-적용](#) >

SpinRoutine / 추첨·목표 각도·완료 통지

```
356 int selectedIndex = UnityEngine.Random.Range(0, SlotCount);
357 RouletteEffectData effect = appliedSlots[selectedIndex];
358 int value = effect == null ? 0 : effect.RollValue();
...
363 float startAngle = wheelTransform.localEulerAngles.z;
364 float clockwiseOffset = GetClockwiseOffsetToMarker(selectedIndex);
365 float targetAngle = startAngle - 360f * 5f + clockwiseOffset;
...
392 ResultSelected?.Invoke(selectedIndex, effect, value);
393 IsSpinning = false;
394 onCompleted?.Invoke(result);
```

칸 추첨



목표 각도



회전



결과 전달

싌 전환 후 진행 상태 복원

싌을 넘는 세션 상태와 현재 맵의 선택 가능 상태를 구분합니다.

세션에 보관

맵에서 다시 적용

완료한 이벤트 좌표

해당 노드를 완료 상태로 표시

현재 이벤트 좌표

복귀 기준이 되는 노드 탐색

현재 이벤트의 완료 여부

다음 노드 개방 또는 현재 노드 선택

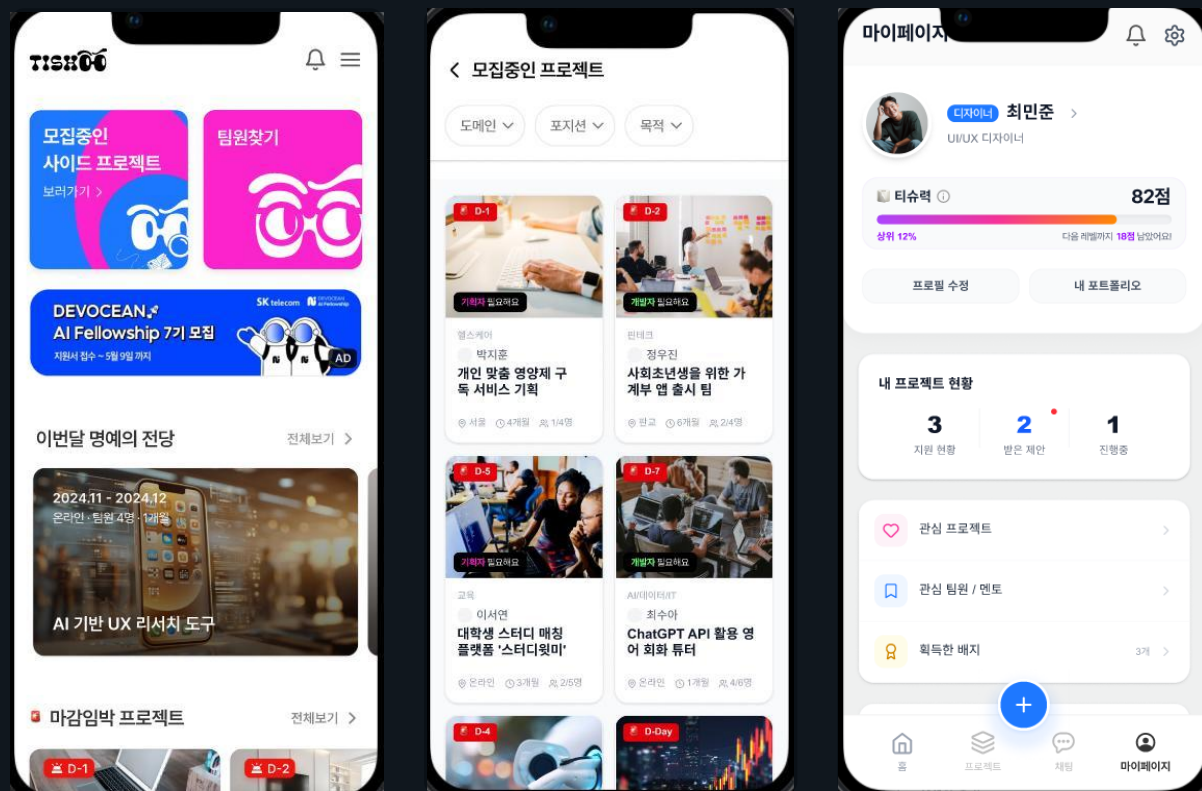
새 회차 시작

플레이어 상태·진행 기록·맵 시드 초기화

이벤트가 진행 중이면 다른 이벤트 시작을 제한합니다. 완료 후에는 진행 중 상태를 해제하고 연결된 다음 노드를 선택할 수 있게 합니다.

Tishoo · 화면과 DB 연동

2025.08-2025.11 | 7인 팀 · 프로그래머 | PHP · JavaScript · MariaDB | 협업 웹 서비스



프로젝트

7인 팀 · 기획 2명, 프로그래머 1명, 디자이너 4명

담당 범위

프론트엔드·백엔드·DB 연동을 맡았습니다. 프로필 수정과 동료 후기의 입력·저장·조회 흐름을 구현했습니다.

게임 개발 외 경험

사용자와 프로젝트의 관계를 데이터로 표현하고, 입력한 정보가 다른 화면에 표시되는 과정을 연결했습니다.

Tishoo / 소스 저장소 >

후기 저장과 프로필 집계

프로젝트·작성자·대상 사용자 정보를 후기 데이터와 연결합니다.



쓰기

평점 범위와 후기 내용이 비어 있는지 확인합니다. 매개변수를 바인딩한 INSERT로 프로젝트와 사용자 식별자, 평점, 내용을 저장합니다.

읽기

후기를 작성자·프로젝트 정보와 조인해 조회합니다. PHP에서 평점 합계와 후기 수, 4점 이상인 후기 수를 계산합니다.

평균 평점	평점 합계 / 후기 수
긍정 비율	4점 이상 후기 수 / 전체 후기 수 × 100
후기 없음	0으로 처리해 나눗셈을 구분

프로필 변경과 공통 레이아웃

페이지별 콘텐츠를 공통 레이아웃에 넣고, 프로필 변경을 DB에 반영합니다.

프로필 변경

사용자 프로필 행이 있는지 조회한 뒤 닉네임을 UPDATE하거나 새 행을 INSERT합니다.

페이지 설정

→ 공통 레이아웃

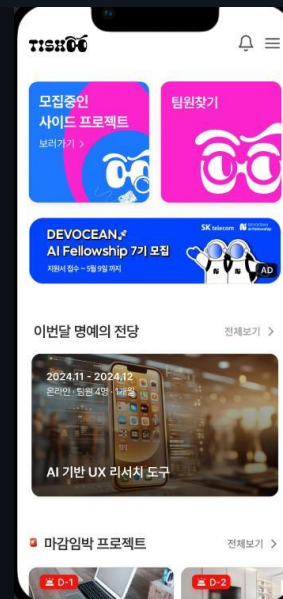
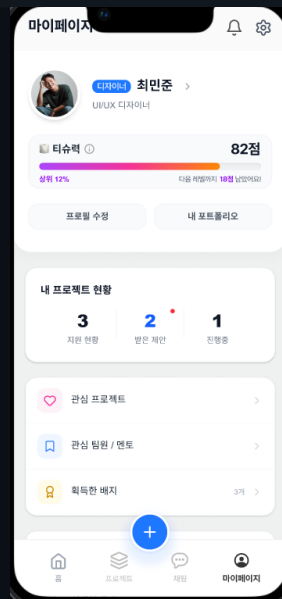
→ 본문·내비게이션

조회 화면

사용자와 프로필 정보를 LEFT JOIN으로 가져옵니다. 프로필 이미지 값이 없으면 기본 이미지를 표시합니다.

공통 레이아웃

페이지 제목, CSS, 본문 뷰, 내비게이션 표시 여부를 페이지에서 정하고 공통 레이아웃에 전달합니다.



03_02_view.php / 프로필 변경 >

layout.php / 공통 화면 >

기술 경험과 소스 안내

프로젝트별 코드 링크와 목차를 통해 필요한 기술 사례로 이동할 수 있습니다.

게임플레이	행동 조건, 공격 판정·타이밍, 장비 스탯, 적·보스 AI, 튜토리얼
클라이언트	입력·선택, 이벤트 UI, 오디오 설정, 카메라·뷰포트
데이터·진행	ScriptableObject, JSON 레벨 검증, 노드 연결, 롤렛, 회차 상태
언어·협업	C#·Unity / C++·SFML / PHP·SQL / Git·GitHub / PM·기능 통합

[Knight Only >](#)

[전기톱 키우기 >](#)

[MapleLikeClient >](#)

[Indie Go >](#)

[کم공과 키우기 >](#)

[Wheel of Fortune >](#)

[Tishoo >](#)