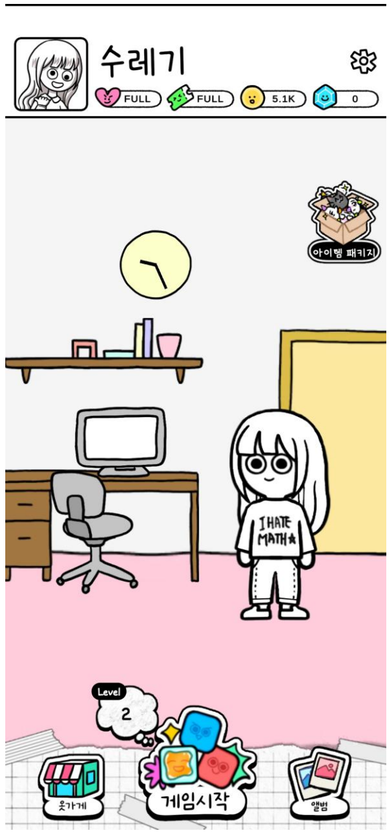
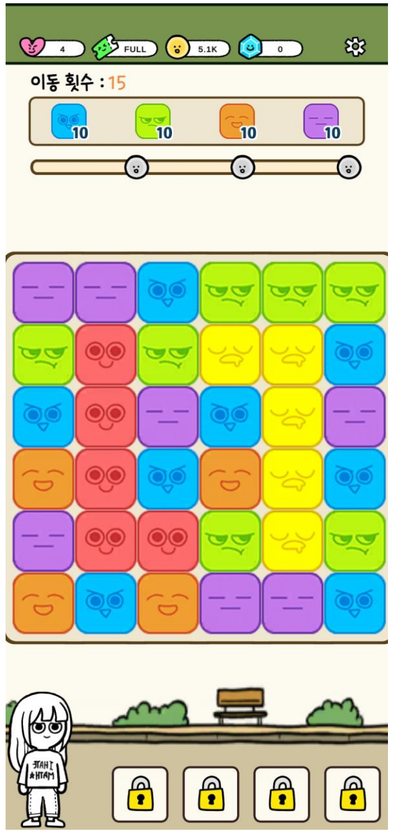
2025.07- 2025.09

수레기 머학생 #(주)메타빌 클로즈베타 게임 개발 인턴 – 프레임워크 담당

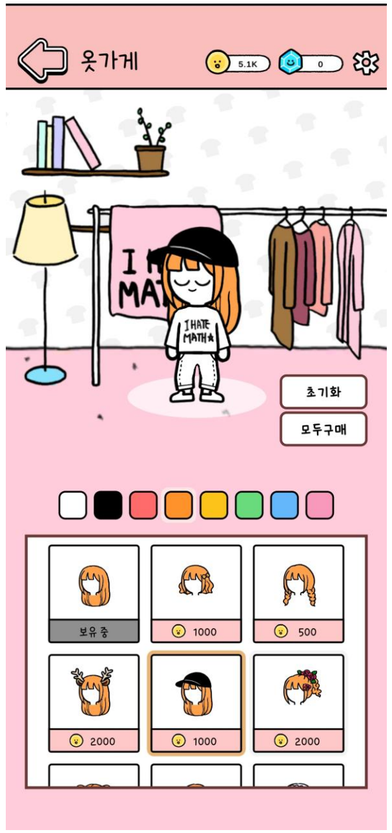


홈



메인 게임

미니 게임



상점

후기

게임 관점에서 시스템을 설계·구현하는 감각을 체득하고, 객체 지향 설계의 가치를 구체적으로 이해하게 된 프로젝트입니다. 주로 객체 지향적 프레임워크를 직접 설계하고 구현하며 많은 고민을 가졌고, 특히 SOLID 원칙을 근거로 기존 구조를 리팩토링하거나 새로운 구조를 구현하는데 집중했습니다.

작업

1. 개방 폐쇄 원칙과 의존성 역전 원칙 기반 리팩토링으로 소프트웨어 생명 연장
2. class diagram을 활용한 게임 전체 구조 문서화와 실 구현으로 유지보수성 확보
3. Unity Project Auditor를 활용한 프로젝트 점검으로 안정성 증가
4. 무한 스크롤과 오브젝트 풀링으로 성능 최적화
5. 다양한 모바일 해상도에 대응 가능한 반응형 UI 작업

재미 퍼즐로 고뇌하고 점수로 경쟁하라.
장르 캐주얼, 퍼즐, 경쟁
소속 (주)메타빌 인턴
참여 5인 (TA 2, 서버 1, 클라이언트 2)
역할 클라이언트 (리팩토링, 프레임워크, UI)
기간 2025.07 - 2025.09
도구 #Unity

수레기 머학생

#1 런칭 리스크를 제거하기 위한 게임 프레임워크 재설계

문제

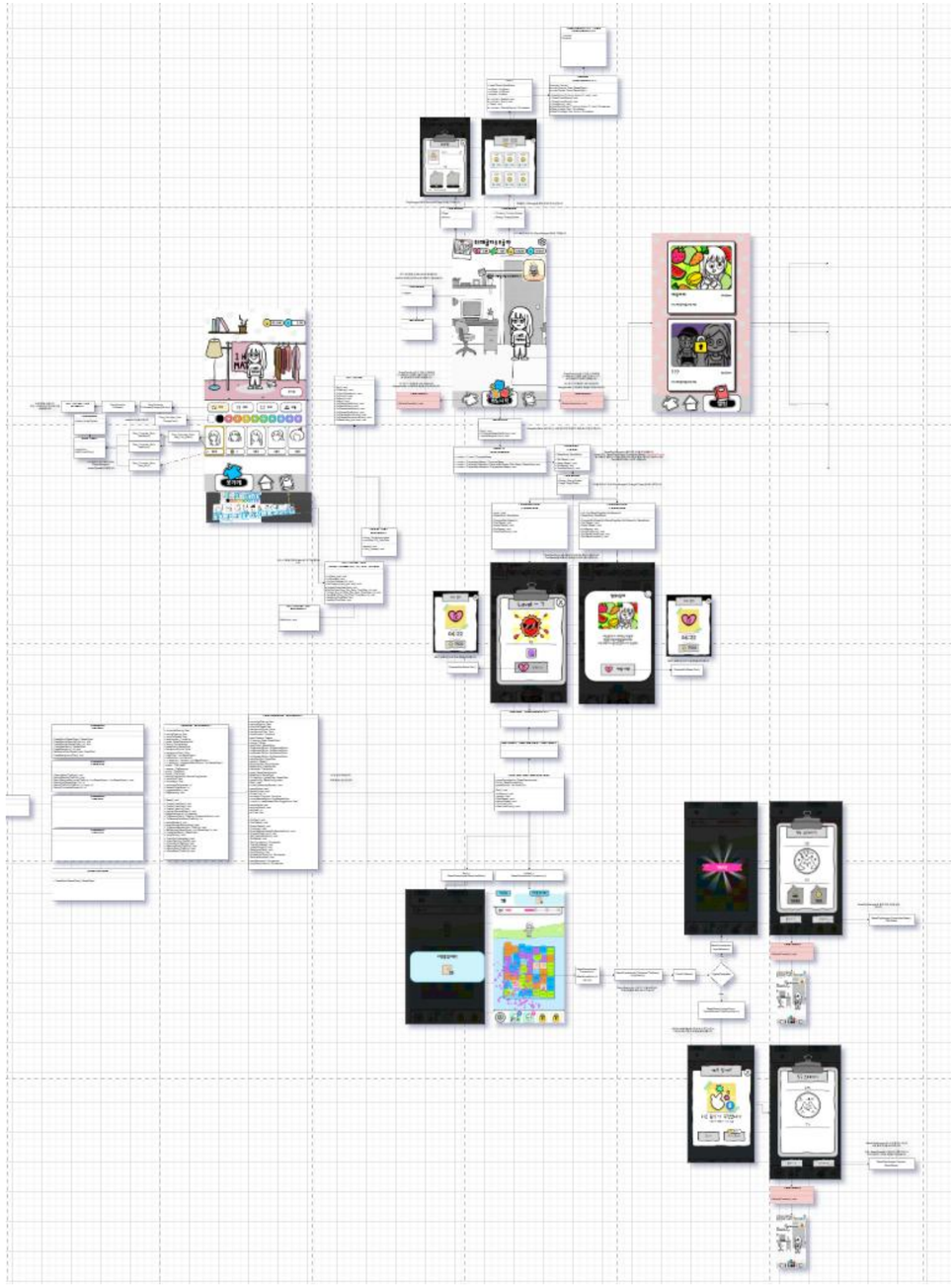
1. 프로젝트에 참여한 후 맡은 역할은 프로젝트 분석과 요구사항 정리였습니다.
2. 기존 구조를 검토해보니, 여러 에셋과 개발 결과물이 체계 없이 무작위로 배치되어 있었습니다.
3. 중복된 기능이 많았고, 전체를 통합 관리할 프레임워크가 없어 장기적으로 유지보수와 확장이 어려운 구조라고 판단했습니다.

해결

1. 게임의 플레이 흐름과 게임이 공통적으로 가지는 핵심 요소를 정리했습니다. (e.g. 게임 중 스마트폰 홈 화면 이동 / 유저가 원하는 재화 획득 타이밍 및 연출, 표기 데이터, 게임 진입 방식 등) (Draw.io)
2. 다년간 쌓아온 게임 경험을 적극 활용했습니다. (게임 아카이브)
3. 이를 바탕으로 전체 구조를 추상화하여 UML 다이어그램으로 문서화했습니다. (Draw.io)
4. UML 다이어그램에 따라 세부 코드 설계 및 구현을 진행했습니다.
5. SOLID 원칙을 적용해 각 클래스의 책임을 최대한 분리하고, 인터페이스를 활용해 의존성을 낮췄습니다.
6. 특히 메인 게임과 미니 게임을 통합 관리하며 게임 시작과 종료, 연속 클리어 처리, 보상 지급, 스마트폰 강제 종료 등 전반적인 흐름을 담당하는 GamePlayManager를 설계·구현했습니다. (GamePlayManager.cs)

영향

1. 프레임워크 정비 후 유지보수성이 이전보다 크게 향상되어 기능 수정과 신규 콘텐츠 추가가 가능했습니다. 또한 시스템 신뢰성이 높아지고 소프트웨어의 수명도 연장되었습니다.
2. 이 경험을 통해 객체 지향 프로그래밍, 특히 SOLID 원칙의 철학을 실무적으로 깊이 이해하게 되었습니다.
3. 또한 게임 프레임워크를 설계하려면 게임 자체에 대한 이해가 필수적임을 깨달았습니다.
4. 이후 게임을 플레이할 때도 "이 구조는 어떻게 설계되었을까?"라는 개발자적 시각을 가지게 되었습니다.



Draw.io를 활용한 프레임워크 도식화

수레기 머학생

#2 메인 게임/다양한 미니게임의 공통 규칙과 흐름을 추상화해, 하나의 통합 관리 구조로 구현

목표

1. 프로젝트에 포함된 많은 게임들을 매번 고려해가며 작업하기엔 효율이 너무 떨어졌습니다.
2. 미니 게임과 메인 게임의 플레이를 통합하여 관리할 수 있는 단 하나의 프레임워크를 개발하는 것이 목적이었습니다.

의도와 구현

1. 게임 전반에서 공통적으로 사용되는 생명, 재화, 서버 통신, 결과 처리 등의 기능을 추상화하기 위해 BasePreparedGame.cs 추상 클래스를 설계하고, 외부에서는 IPreparedGame 인터페이스를 통해 구현체에 의존하지 않고 접근하도록 구성했습니다.
2. 데이터 준비나 팝업 연출 같은 세부 동작을 게임에 따라 분리하기 위해 PreparedMainGame.cs 과 PreparedMiniGame.cs 클래스를 구현했습니다.
3. 수정 가능성이 높다고 판단된 게임 결과/보상/생명 등은 개방 폐쇄 원칙(+ 단일 책임 원칙)에 근거해 GameResult, GameReward, GameLife 같은 하위 모듈로 분리했습니다. 실제로 서버 통신 수정 작업 때 GameResult 구현체만 교체하여 다른 시스템 수정 없이 안전하게 대응할 수 있었습니다.
4. 빠른 시작, 메인 게임, 미니 게임 등 게임 시작 흐름을 관리하기 위해 GamePlayManager.cs 클래스를 작성했습니다. 초기에는 씬 전환 시 상태 유지를 위해 static 접근 방식을 사용했으나, 전역 상태로 인한 테스트 및 의존성 문제를 인식하고 이후 게임 씬 내부에서는 DI 방식을 통해 필요한 클래스를 주입 받아 사용하도록 설계했습니다.
5. 미니 게임 데이터는 난이도별 변수 값만 필요해서 ScriptableObject로 제어했습니다. 메인 게임 데이터는 100개가 넘는 스테이지에 맵 데이터 정보까지 필요해서 json 형태로 저장/로드했습니다. (MiniGameScriptableObject.cs)

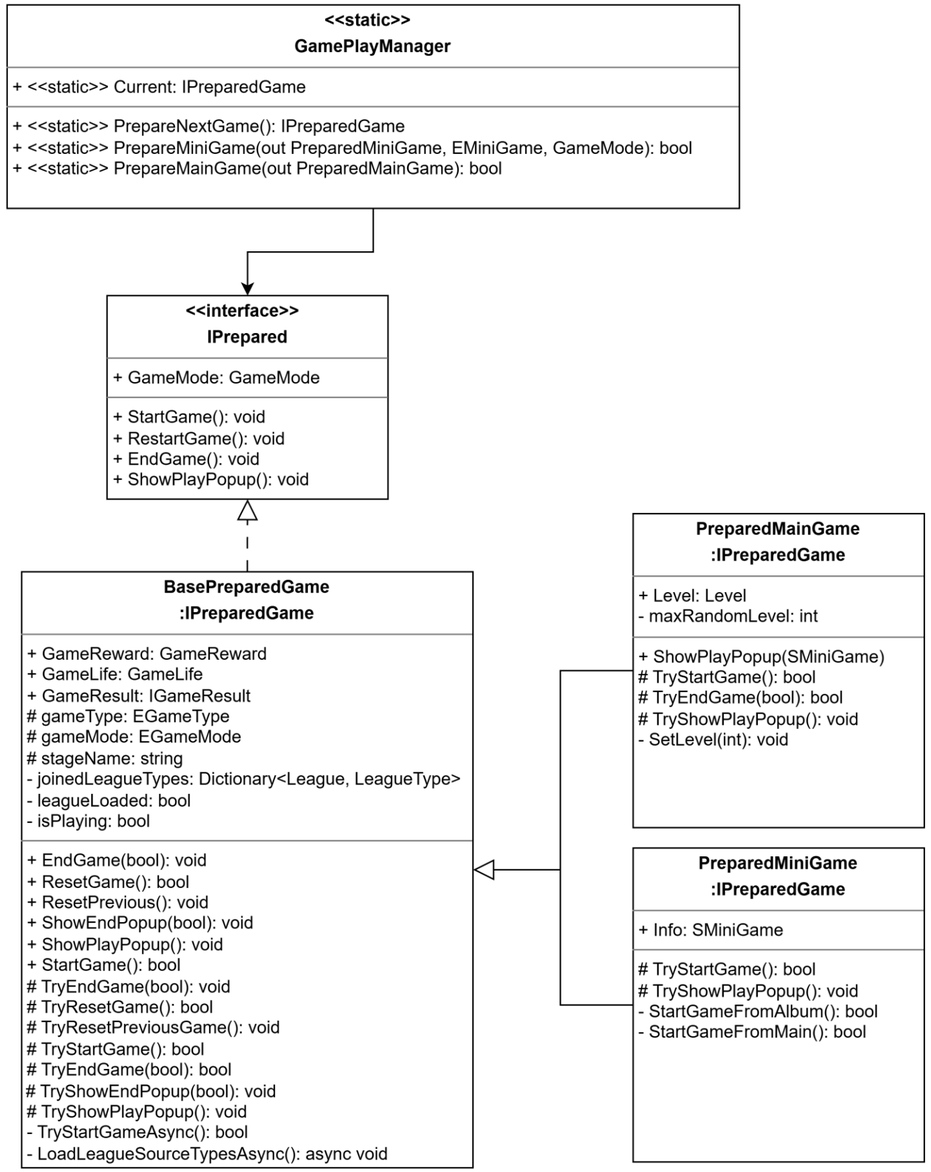
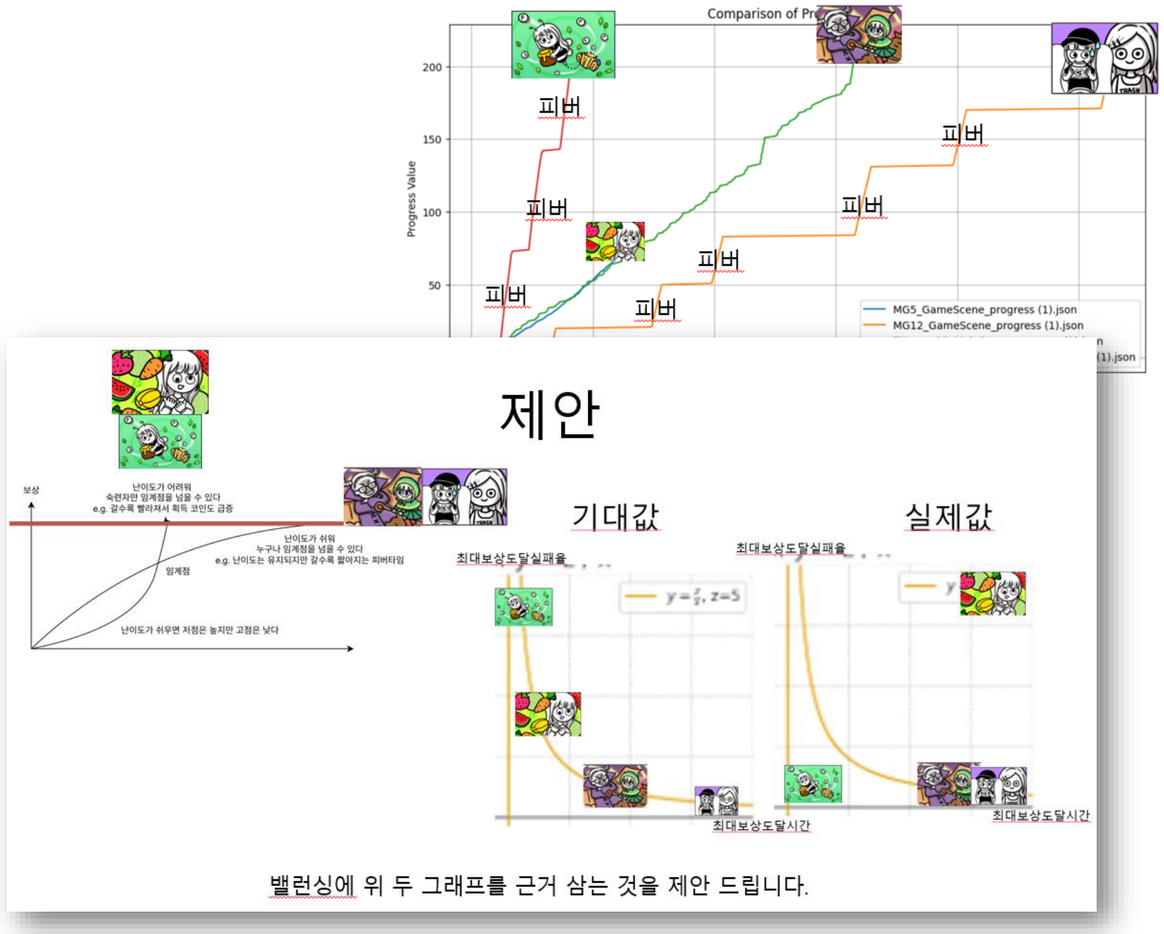


Fig. GameManager Class Diagram

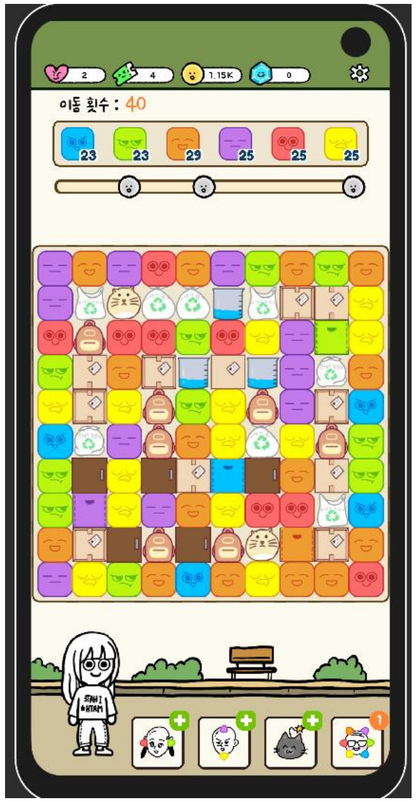
수레기 머학생

평범한 작업들

시간 당 실제 보상



미니 게임 플레이 데이터를 수집해 json으로 기록했고, 이를 python으로 도식화해 밸런스를 분석하고 앞으로의 방향성을을 제시했습니다. (PDF)



모바일 환경에서 다양한 화면 비율과 노치 문제를 고려해 고정 UI 구조를 반응형 UI로 전환했습니다. Anchor와 Pivot을 기준으로 UI 레이아웃을 재정의한 뒤, 노치 영역 보정과 게임 오브젝트–UI 간 좌표 매칭은 스크립트로 제어해 기기별 차이에 유연하게 대응했습니다. (코드)