액션핏 클라이언트 개발 과제 문서

요구 사항

1. **Code Refactoring**
   - `MVC` 또는 `MVP` 패턴 기반으로 입력, 상태, 렌더링 로직 분리 및 최적화
   - `BoardController`의 역할을 세분화 하여 서브 컨트롤러 또는 헬퍼 클래스로 분리
2. **Stage Editor 구현**
   - 비개발자도 쉽게 사용할 수 있는 `UI/UX` 제공
   - 블록 배치 및 색상 설정 기능
   - Gimmick 정보 설정 및 `향후 기획 확장 고려`
   - Wall 및 출구 설정 기능 포함
   - Editor Play 기능 포함
3. **Visaul Effect 최적화**
   - 기존 `Quad`를 이용한 가림 처리 방식 → `Stencil Buffer 기반 셰이더`로 전환
   - 실제 오브젝트가 아닌 `버텍스 기반 스텐실`로 구현

분석

▼ BoardController

- 생성
- 레벨 이동
- 파괴 판단

▼ BlockDragHandler

- 마우스 이벤트
- 블록이동 및 충돌 체크

1. BlockDragHandler
    a. 마우스 이벤트 (`Down`, `Up`, `FixedUpdate`로 이동, 충돌 감지)
    b. 움직임 로직
    c. `BlockGroup`에 달려있음
2. BoardBlockObject
    a. 파괴 판정 및 파괴 연출 (Board의 하나의 Cell들)
3. BlockObject
    a. 블록 좌표 설정 및 BoardBlockObject의 playingBlock 갱신
4. BoardController
    a. 레벨에 따른 보드 만들기 로직
5. StageEditor
    a. 기존 Json To StageData 로직 최대한 재활용

방향성

- 시간이 많지 않으므로 기존 로직은 최대한 재활용한다.
- 리팩토링 및 추가를 하더라도 기존 플레이는 해치지 않는다.

Code Refactoring

▼ MVC, MVP

- 현재 상황에서 MVC, MVP 패턴으로 바꾸는 것이 적절한가?
    - 현재 BoardBlockObject, BlockObject, BlockDragHandler 등 서로 의존성이 높음
    - 현재 로직을 그대로 활용하며 Block의 Model, View, Controller(Presenter)로 역할을 명확하게 구분하기 어려움 → 각자 Data, View, Logic을 가지고 있음 → 분리 하려면 우선 서로의 의존성 제거가 우선
    - 서로 의존성을 제거하려면 로직 수정도 필수로 들어가야 함(로직안에 의존성이 있기 때문) → 시간이 오래걸림

    - 이벤트로 인한 데이터(Model)의 변화가 자주 일어남 → 실시간으로 Model - View 또는 View - Presenter - Model 통신이 너무 자주 일어난다 → 오히려 비효율적

▼ BoardController 역할 분리

- 생성 로직 → Factory로 분리
- 초기화 로직 → BoardInitializer로 분리
- BoardController는 최대한 간단하게 : 레벨 변경, 파괴 체크
- 시간이 부족하여 로직 및 멤버 변수 모두 의존성을 제거하기는 힘듦

▼ BlockDragHandler 이벤트, 물리 역할 분리

- `MouseEventHandler`를 별도 컴포넌트로 분리하여, Unity 이벤트(`OnMouseDown`, `OnMouseUp`)를 **구독 방식으로 전달**하도록 구현했습니다.
- 이 방식은 특정 클래스(BlockDragHandler 등)에 묶이지 않기 때문에, 마우스 입력이 필요한 **다른 오브젝트에서도 그대로 재사용 가능**한 구조입니다.
- 결과적으로 `BlockDragHandler`는 블록의 물리 역할인 이동 및 충돌 체크만 담당
- Mouse 이벤트 발생 → `MouseEventHandler` → `BlockDragHandler`에게 알림 → 해당 정보로 물리 연산

Stage Editor 구현

▼ Stage Editor

- 2차원 배열 버튼 그리드를 생성하여 원하는 위치 버튼 클릭하여 블록 설정
- 벽 정보 또한 2차원 배열 버튼 그리드를 활용하여 보기 쉽게 설정

▼ BlockInfoWindow

- Color 설정
- 버튼 그룹 설정

▼ WallInfoWindow

- BlockInfoWindow와 같이 버튼 그리드를 활용하기에는 Wall 설정이 복잡합
- 모서리에 경우 여러개의 벽이 들어갈 수 있기 때문 → 오히려 난잡해짐
- Direction의 의미가 헷갈림 벽이 바라보는 방향이지만 Length를 설정하여 벽을 설정할 때 어느 쪽으로 늘어날 것인지 기준이 없다시피함 → 프리팹의 Pivot이 가운데에 있어서 문제가 되는 듯함

- ToolTipWindow를 추가하여 ToolTip 버튼 클릭시 해당 파라미터 정보를 알려주는 방식으로 설정할 수 있게 설정

▼ Gimmick

- GimmickData에 로직을 담고 추후 기믹을 사용할 때 해당 기믹에 접근하여 로직 사용
- `GimmickData` 상속, `GimmickContainer` 에서 가져오기

PR PLAN `epic/actionfit_client`

1. Code Refactoring `feature/code_refactoring`

   > https://github.com/BoKangKim/Employment-Task/pull/1

   1. BoardController 역할 분리 (완료)
      a. `Factory` 구현 (생성 담당)
      b. `BoardInitializer` 구현 (Board 초기화 담당 및 Board 데이터 담당)
   2. `BlockDragHandler` 역할 분리
      a. `MouseEventHandle` 로 이벤트만 따로 관리
      b. 물리 연산은 그대로 `BlockDragHandler` 에서만 하도록

2. **Stage Editor 구현** `feature/editor_tool`

   > https://github.com/BoKangKim/Employment-Task/pull/2

   1. 최상단 Window (Stage Editor) 구현
   2. 블록 정보 설정 Window (BlockInfoWindow) 구현
   3. 벽 정보 설정 Window (WallInfoWindow) 구현
   4. Gimmic 설정 → GimmickData에 로직을 넣어서 사용