0401 회의

회의 준비

- **M + C 담당 3명, V 담당 2명**
- **1차**: Swing 기반 UI
- **2차**: JavaFX 또는 SWT 등으로 UI 교체 → **그때는 4명까지 V에 참여 가능**

✅ 팀원별 역할 분담 (5인 기준, M/C 3명 + V 2명)

이름 (예시)	1차 (Swing 중심) 역할	2차 (UI 교체 시) 추가 역할
A (M) 시윤	🎯 모델 전담- Player, Piece, Board- 이동/잡기/업기 로직- 윷 결과 생성 로직	- 모델 로직 리팩토링 지원- 뷰와 모델 연동 점검
B (M + C) 승언	🎯 컨트롤러 핵심- 사용자 이벤트 처리- 던지기 → 이동 → 결과 흐름 제어	- UI 교체에 맞춰 controller/view 연결 점 수정
C (M + 테스트) 영찬	- 모델 보조- JUnit 테스트- 경로 구조 설계- 승리 조건 판별	- UI 테스트 지원- 문서화 & GitHub 활동 관리
D (V: Swing 전담)	🎯 1차 UI 구현- 버튼, 판 렌더링- 사용자 말 선택 처리- 잡기/업기 시각화	- 2차 뷰 구조 설계 참여- 애니메이션, 효과 등 확장 기능
E (V: Swing 보조 → 2차 주도)	- Swing 보조 UI 작업- 재시작/종료 UI- 텍스트 출력/말 이미지 등	🎯 2차 UI 프레임워크 주도 개발- 새로운 뷰 구현- Swing 코드 이식 or 개선

🔁 2차 전환 시 인력 재배치

역할 구분	참여 인원	설명
새 UI 구현 (JavaFX 등)	**E, D, B, C** (4명)	View 교체 시 병렬 작업 가능
모델/컨트롤러 리팩토링	A, B	View 교체에 따른 최소 조정 작업

✅ 역할 배분 포인트 정리

- **A**는 모델 로직의 핵심 담당 → 구조적 일관성 유지
- **B**는 Controller를 중심으로 M과 V 연결을 책임 → UI 바뀌어도 변경 지점 최소화하도록 설계

- **C**는 테스트 담당 + 보조 설계자 → 코드의 신뢰성, GitHub 정리 등 전반 관리
- **D/E**는 UI 파트너 → 1차는 Swing, 2차는 새로운 도구로 전환

  → 2차에서는 팀원 확대 투입으로 빠르게 마이그레이션 가능

🔎 참고: 흐름 기반 분담 예시

흐름 단계	담당자	비고
게임 초기화 / 플레이어 구성	A	Model 중심
윷 던지기 / 결과 판단	B	Controller
이동 처리 / 업기 / 잡기	A + C	Model + 테스트
UI 그리기 / 선택 / 결과 표현	D + E	Swing: D 중심, JavaFX: E 중심
문서화 / 발표 자료 / 테스트 코드	C	전체 서포트 역할

👣 진행 전략

1단계. 모델 설계 협의 (A, B, C)

- 먼저 `Player`, `Piece`, `Board`, `Cell`, `YutResult` 등의 **클래스 이름, 책임, 메서드 목록**을 대략 정의해
- 이걸 바탕으로 **"모델 인터페이스 명세"**를 만들어

```
// 예시 (초안)
class Player {
    int getPlayerId();
    List<Piece> getPieces();
}

class Piece {
    Cell getCurrentCell();
    void move(int steps);
}

class YutGame {
    YutResult throwYut(boolean isRandom, String forcedResult);
    boolean isGameOver();
```

```
    void movePiece(int pieceId, int steps);
}
```

📌 **이 인터페이스가 컨트롤러의 발판**이 돼

2단계. 컨트롤러는 "인터페이스 기반"으로 먼저 설계

- 모델의 내부 구현이 완성되지 않아도, **약속된 메서드 시그니처만 있으면** 컨트롤러에서 호출할 수 있어
- 실제 내부는 `TODO`로 채워도 됨

```
// Controller 내부
YutResult result = game.throwYut(true, null); // 모델 호출
view.showYutResult(result);
```

3단계. 모델 구현 진행 → 실제 동작으로 연결

- A, C가 모델을 구현하면, B는 기존의 컨트롤러 호출 로직에 연결만 하면 됨
- **B는 로직 흐름 중심, A는 세부 동작 구현 중심**으로 협업 가능

✅ 병렬 작업을 가능하게 하는 포인트

조건	설명
모델 클래스/메서드 설계 협의	컨트롤러가 호출할 구조만 명확하면, 내부는 나중에 구현해도 됨
인터페이스 중심 개발	내부 구현 없이, 외부 호출 로직만 먼저 짤 수 있음
예외/결과 구조만 명확히 공유	예: `throwYut()`이 `YutResult` 반환한다는 것만 확실하면 테스트 가능

💡 팁: 초기에 같이 정리하면 좋은 인터페이스

- `YutGame` 클래스의 메서드 목록
- `Piece`, `Player`의 기본 구조
- `Cell`과 이동 경로 표현 방식
- 이동, 잡기, 업기 로직의 호출 흐름

✅ 결론

> 컨트롤러는 모델이 완성되어야 "진짜 동작"은 가능하지만,
>
> **설계와 코드 구조는 미리 작업 가능하다!**

협의된 모델 인터페이스만 있으면 병렬 진행 가능하니,

초기에 A+B+C가 모여서 클래스 구조/메서드 정의 정도는 확실히 잡아두는 걸 추천