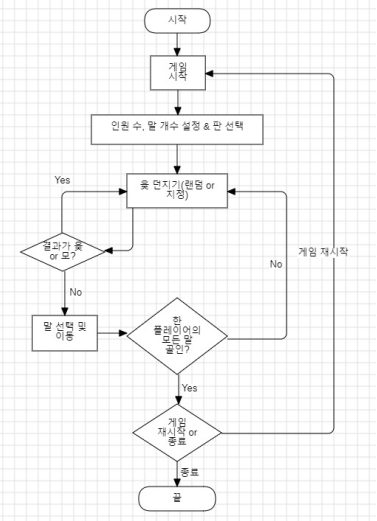
소프트웨어 공학 1 분반 Team 25

김성재, 김찬우, 김현진, 이현택, 전서영

1. 프로젝트 개요

본 프로젝트는 전통 윷놀이 게임을 객체지향 분석 및 설계(Object-Oriented Analysis and Design, OOAD) 기법을 기반으로 Java 언어로 구현한 소프트웨어이다. OOAD 의 대표적인 분석 기법인 UseCase 모델링, Sequence Diagram, Operation Contract 등을 활용하였으며, MVC(Model-View-Controller) 패턴을 적용해 유지보수성과 확장성을 확보하였다. 오른쪽 <그림 1>은 본 프로젝트의 흐름도이다.



<그림 1> 흐름도

2. 게임 설계 원칙

본 프로젝트를 구현해 나가면서 강의에서 배운 설계 원칙들을 적재적소에 사용하였는지에 대한 설명이다. 이를 각각의 코드 파일(.java)들에 대해 적용된 설계 원칙들에 대해 기술할 예정이다.

① YutGameLauncherApp / UIMode

설계 원칙/패턴	설명
SRP	YutGameLauncherApp 클래스는 애플리케이션의 시작을 담당하는 단 하나의 책임만을 갖는다. UI 초기화 및 컨트롤러 연결은 앱 실행 시점에만 수행하며, 이후 로직은 각 UI 구현체에 위임하여 단일 책임 원칙을 충실히 따름.
GRASP (Creator)	YutGameLauncherApp 은 실행 시 선택된 UIMode 에 따라 적절한 UI View 와 Controller 객체를 생성한다. 이처럼 객체 생성을 책임질 '타당한 정보 보유 클래스'로서 GRASP 의 Creator 역할을 수행함. 또한 내부 구현 대신 생성 책임만 갖고, 실제 로직은 위임하는 구조로 책임 분산을 실현.
MVC Pattern	해당 코드는 MVC 패턴의 시작점 역할을 하도록 설계. View : GameSetUpView(FX) - 사용자에게 UI 제공. Controller : GameEvent Launcher(YutGameLauncherApp) : MVC 의 연결자/조립자역할.
Seperation of Concerns	실행 로직(main), UI 초기화(View), 게임 흐름 제어(Controller)를 명확히 분리하였다. YutGameLauncherApp 은 UI 내부 동작에 전혀 개입하지 않으며, 선택된 UI 모드에 따라 적절한 객체만 주입함.

Threading	Swing UI 모드의 경우 SwingUtilities.invokeLater()를 통해 UI 초기화는 EDT(Event Dispatch Thread)에서 수행되도록 설계. JavaFX 모드에서는 FX Application Thread 를 이용함으로써 각 플랫폼의 UI 스레드 안전성 확보함.
OCP	UI 를 선택적으로 교체할 수 있는 구조로 확장성을 고려함. UIMode enum 을 사용하여 JavaFX/Swing UI 선택이 가능하고, UI 외의 컨트롤러/모델 로직은 변경 없이 재사용됨. 새로운 UI 모드 추가 시 기존 코드를 수정하지 않고 확장만으로 대응 가능함.

- 개선 여지: 현재는 enum 기반 분기를 통해 UI 를 생성하고 있지만, 향후 DI(Dependency Injection) 프레임워크나 팩토리 메소드 패턴을 적용하면 더 유연하고 테스트 가능한 구조로 확장 가능. 추상 팩토리 또는 전략 패턴과의 결합도 고려할 수 있음.

② GameController

설계 원칙/패턴	설명
SRP	View 가 Game 로직에 접근하는 경로를 정의하는 responsibility 하나만을 갖고, 해당 인터페이스는 게임 로직 구현의 중재자 역할만을 하도록 설계.
DIP	해당 인터페이스에만 의존하고 추상 인터페이스를 통해 로직을 제어하여 DIP 를 만족하도록 설계.
LSP	확장 가능하고 어떠한 GameControllerImpl 이라도 대체할 수 있도록 설계. 잘못된 전제 조건이나 예외 없이 명확한 contract 를 제공.
OCP	인터페이스를 변경 없이 확장 가능하도록 설계. 예를 들어 새로운 게임 규칙을 도입하려면 GameControllerImpl 을 상속하거나 교체하면 되고, 기존 인터페이스는 수정할 필요 없도록 설계.
Observer Pattern	registerView/unregisterView 는 View 를 등록하고, 이후 GameEvent 를 broadcasting 하도록 설계.
Interface-Based Programming	View 나 다른 계층은 해당 인터페이스에 대해서만 알고 구체적인 구현은 몰라도 되도록 설계.
GRASP	Controller : View 와 Model 사이를 중재하는 진입점 역할 수행 Polymorphism : 여러 구현을 인터페이스로 추상화하여 다형성 적용 Low Coupling : View 는 구현이 아닌 추상에 의존

- 개선여지: GameController 인터페이스가 다소 많은 메소드를 갖기 때문에 관심사별 인터페이스로 조금 더 분리해두는 것이 명확하고 좋아보임.

③ GameControllerImpl

설계 원칙/패턴	설명

SRP	게임의 상태 변경과 이벤트 중재자 역할에만 집중하도록 설계. Model 의 세부 로직은 위임하고 View 는 별도로 관리함.
OCP	GameController 인터페이스 기반으로 설계하여 기능 변경이 필요하면 상속이나 대체로 확장 가능.
DIP	View 는 GameController 인터페이스를 통해 Impl 에 접근하고 Impl 은 내부적으로 GameView 인터페이스만 알도록 하여 의존성 낮춤. LSP 모든 메서드의 올바른 구현과 예외처리를 통한 안정성을 확보하도록 설계 View 는 GameController 인터페이스를 통해 Impl 에 접근하고 Impl 은 내부적으로 GameView 인터페이스만 알도록 하여 의존성 낮춤.
LSP	모든 메서드의 올바른 구현과 예외처리를 통한 안정성을 확보하도록 설계.
Observer Pattern	View 객체를 registerView/unregisterView 로 등록하고 게임 상태 변화 시 publish()를 통해 모든 View 에 알리도록 설계.
GRASP	Controller : 사용자 입력에 대한 모든 로직을 Impl 이 처리하도록 설계. Low Coupling : View 와의 통신은 GameView 인터페이스를 통해 이루어지고, 내부 구현은 외부에 노출되지 않도록 설계.

- 개선여지: 인터페이스를 조금 더 세분화 할 수 있다. publish()를 EventDispatcher 클래스로 추출하면 테스트 용이성이 향상될 것 같다. publish()와의 Logger 연동을 고려한다.

④ GameEvent

설계 원칙/패턴	설명
SRP	“이벤트를 표현한다”는 하나의 responsibility 를 가지도록 설계.
Observer Pattern	해당 클래스는 이벤트 발행자 → Viewer 간의 메시지를 전달하는 “이벤트 메시지” 객체 역할로 설계.
Immutable Object	테스트에 용이하고 디버깅의 편리함을 위해 type 과 data 를 final 필드로 선언하여 생성 이후 변경 불가하도록 설계.
Encapsulation	외부가 내부 구현을 몰라도 사용 가능하도록 설계.

⑤ GameEventType

- 게임 흐름 중 발생할 수 있는 다양한 이벤트의 Type 을 정의하며 Controller -> View 로 이벤트를 전달할 때 사용됨. 설계 원칙으로는 SRP, OCP, Type-Safety 을 이용하였고, 패턴으로는 Event-Driven Arichitecture 를 사용함. 이를 통해 매직 문자열 제거, 컴파일 시의 안정성, IDE 자동완성의 이점을 가져올 수 있도록 설계함.

⑥ Board

- 게임 말의 이동 규칙 Capsulation. 게임 엔진과 뷰를 명확히 분리하는 것에 초점을 두어 설계함. 이를 통해 SquareBoard 뿐만 아니라, Pentagon, HexagonBoard 에서도 유연하게 사용 가능.

⑦ GameBoardViewFX

설계 원칙/패턴	설명
MVC	UI 구성과 Event 처리를 위한 View 클래스이며, 실제 로직처리는 콜백 인터페이스에 위임하며 View 와 model 이 분리되도록 설계.
DIP	Controller 를 바꾸거나 테스트용으로 교체하기 쉽도록 추상화된 인터페이스인 UiCallback 설계.
SRP	UI 의 생성 및 갱신, 사용자 Event 위임 처리를 위해 설계.
Low Coupling	게임의 내부 상태 구조에 대한 직접적인 정보 없이 오직 필요한 데이터만 전달하거나, 콜백을 호출하도록 설계.
Controller	버튼 클릭 시의 처리 책임을 View 가 하지 않도록 하기 위해 외부에서 전달된 UiCallback 에 위임하도록 설계.

⑧ GameSetUpViewFX

- 이전 클래스와 마찬가지로 MVC-View, SRP, DIP, GRASP-Controller,Low Coupling 에 기반하여 설계하였고, panel 메소드들을 통해 UI 의 구성코드를 모듈화하여 재사용 가능하도록 설계, 내부 UI 구성요소들을 Encapsulation 에 기반하여 설계.

⑨ GameSetUpView

설계 원칙/패턴	설명
MVC	View 로 게임 시작 전의 UI 만 구성하도록 설계.
SRP	사용자의 설정 입력만 받는 화면'만을 담당하도록 설계.
Information	Expert 에 기반하여 플레이어 수, 말 개수, 판 형태로 선택된 객체가 정보를 반환하도록 설계. UI 구성을 별도 메소드로 분리하여 재사용성과 가독성 높이도록 설계.
Encapsulation	외부에선 공개된 getter 만 통해 접근하도록 설계.

⑩ GameCompleteView

설계 원칙/패턴	설명
MVC	View 로 게임 종류 후의 UI 만 담당하도록 설계. 또한, 마찬가지로 SRP 에 기반하였고, Observer 패턴에 기반하여 ActionListener 를 통해 사용자의 액션을 감지하도록 설계

⑪ GameBoardView

- 위의 클래스들과 같이 MVC-View, DIP, SRP, GRASP-Controller 에 기반하여 설계하였고, UI 초기화 과정을 메서드로 분리하여 View 생성의 모듈성과 가독성을 높이도록 설계.

⑫ PieceView

- MVC(View),SRP, Information Expert 에 기반하여 설계하였고, High Cohesion 으로 랜더링 관련 별로 묶어서 설계. Low Coupling 으로 유연한 구조를 갖도록 설계.

⑬ SwingGameViewAdapter

- MVC 의 View 계층의 Adapter 로서, 사용자 인터페이스와 도메인 컨트롤러를 연결하도록 설계. 본 클래스는 View 업데이트만 담당하여 MVC 의 View-Controller 경계를 명확히 유지하도록 설계.

⑭ GameView

- View 계층이 Controller 로 부터 게임 Event 를 받기 위해 설계한 표준 인터페이스이다. ISP 에 기반하여 다양한 View 구현체가 이 인터페이스를 기반으로 유연하게 구현하도록 설계.

⑮ BoardPanel

설계 원칙/패턴	설명
SRP	오직 보드를 그리고 UI 에 표현하는 일에만 책임을 갖도록 설계되었다. 보드의 형태(사각형, 오각형, 육각형)에 따라 동적으로 도형을 구성하고, 말 위치도 그래픽적으로 출력함. 게임 로직이나 상태 변화는 외부 클래스가 담당하고, 본 클래스는 해당 정보를 바탕으로 시각적으로 그려주기만 함.
Controller	비록 주요 컨트롤러 클래스(GameControllerImpl)가 로직을 담당하지만, BoardPanel 은 사용자 인터랙션(말 위치 업데이트 등)에 따라 화면을 재구성하고 이벤트를 반영하는 역할을 일부 수행.
Encapsulation	posIdToPixel, pieceManager 등 내부 핵심 상태는 private 으로 은닉하고, 필요한 기능만 public 메서드로 노출함.
GRASP	BoardPanel 은 내부에서 PieceView.Manager 객체를 생성하고 보유함. 해당 객체는 보드에 출력될 말 정보들을 다루며, BoardPanel 이 말의 시각적 상태를 관리하는 책임을 갖기에 자연스러운 Creator 역할을 수행.

⑯ Square, Pentagon, HexagonBoard

- 보드의 구조 및 경로 계산 책임을 수행하는 코드이며 Board 인터페이스를 기반으로 하여 확장에 용이하고, 다형적 사용이 가능함. 이동 경로는 내부 Map 으로, 외부는 next()만 제공하여 정보를 은닉한다.

⑰ Game

설계 원칙/패턴	설명
SRP	오직 게임 규칙에 따른 전달만 담당하고 UI, 입력 처리, 이벤트 통지는 포함하지 않도록 설계.

OCP	새로운 보드가 생기더라도 switch 문만 확장하면 되도록 설계.
Encapsulation	Players, pieces 와 같은 collections 는 외부에서 직접 수정이 불가하고 읽기 전용으로 노출되도록 설계.
GRASP	Game 은 핵심 데이터 보유자로서, Information Expert Pattern 으로 모든 말, 플레이어, 그룹, 보드 정보를 알고 있고 이동, 잡기, 업기 등에 책임을 갖는 것이 자연스러움으로 그에 맞게 설계.
데이터 일관성 유지	그룹을 병합하거나 제거할 때 스택 공유를 항상 동기화하도록 설계했고, 잡힌 말도 그룹에서 제거하고 위치를 복원시킴.

⑱ MoveOutcome

- Game 과 UI 간의 이동 결과 공유를 위해 설계한 클래스이며, UI 는 해당 클래스를 바탕으로 말 위치 갱신, 잡기 애니메이션, 추가 턴 등을 처리함. 또한, SRP 를 적용하여 Game.move()는 게임 상태를 갱신하고, 결과만 MoveOutcome 으로 요약해 반환하도록 함.

⑲ Piece

- Stack 기반의 이동 경로 기록으로 윷놀이의 독특한 경로 구조를 구현. 공유 스택 구조로 Grouping 을 간결하게 처리. 단순한 외부 API(setPosition, pushPath 등)로 Controller 와 View 에서 쉽게 조작할 수 있도록 설계하였다.

⑳ Player

- 게임에서 플레이어 한 명을 모델링한 불변 객체로써 플레이어 단위 정보 관리, 말과의 연결(ownerID), 게임 흐름 처리에 용이하도록 설계하였다.

㉑ YutResult

- Enum Pattern 을 통해 각 결과들을 안전하고 풍부하게 표현함. 또한, GRASP 의 Information Expert 에서 윷의 이동 거리와 추가 턴 여부는 YutResult 가 가장 잘 알고 있으므로 해당 정보와 관련된 responsibility 를 가짐. 캡슐화를 통해 YutResult 값을 외부에서 접근하지 못하고 내부 메소드를 통해 조회할 수 있도록 설계함.

3. Use Case Model

① 게임 초기 설정

Actor	플레이어
Trigger	프로그램 실행 직후
Pre-condition	프로그램이 정상적으로 구동됨
Main Flow	1) 플레이어는 참여자 수(2–4 명)와 말 개수(2–5 개)를 입력한다. 2) 각 플레이어 이름/말 색상이 UI 에 표시된다. 3) 시작을 누르면 게임이 초기화된다.

Post-condition	초기 말 배치 및 차례 확정

② 윷 던지기

Actor	플레이어
Trigger	**랜덤 윷 던지기** 또는 **지정 윷 던지기** 버튼 클릭
Main Flow	1) 플레이어가 버튼을 선택한다. – **지정**: 빽도/도/개/걸/윷/모 중 하나를 선택(테스트용). – **랜덤**: 시스템이 무작위 결과를 생성. 2) 결과가 UI에 표시된다. 3) 윷/모가 나오면 다시 윷을 던진다.
Post-condition	결과가 이동 가능 리스트에 등록됨

③ 이동 결과 적용

Actor	플레이어
Trigger	이동할 말을 클릭
Main Flow	1) 시스템이 해당 말의 가능한 경로를 하이라이트한다. 2) 플레이어가 경로(분기 포함)를 결정하면 시스템이 이동을 수행한다. 3) **필요한 경우**(예: 모+걸 등 다중 결과) 시스템이 어떤 결과를 말에 적용할지 순차로 물어본다.
Post-condition	초기 말 배치 및 차례 확정

④ 말 업기 (Grouping)

Actor	시스템 (자동)
Trigger	동일 플레이어의 말이 같은 칸에 도착
Main Flow	1) 시스템이 말들을 **그룹**으로 표시한다. 2) 그룹은 이후 하나의 말처럼 이동한다.
Post-condition	초기 말 배치 및 차례 확정

⑤ 상대 말 잡기

Actor	시스템 (자동)
Trigger	이동 결과로 상대 말이 있는 칸에 도착
Main Flow	1) 상대 말을 판 밖으로 내보낸다. 2) 윷을 한 번 더 던진다.
Post-condition	잡힌 말 초기화, 윷 던지기 턴 추가 부여

⑥ 분기-경로 선택

Actor	시스템 (자동)
Trigger	오각형·육각형 판에서 **중심점** 또는 **P 분기점**에 도달

Main Flow	1) 플레이어가 선택한 말과 윷 적용 결과에 따라 이동한다. 2) 중심점에 멈추는 경우, 종료 지점과 첫 번째로 가까운 경로로 이동한다. 3) 중심점을 지나는 경우, 종료 지점과 두 번째로 가까운 경로로 이동한다.
Post-condition	선택된 경로로 이동

⑦ 말 이동 결과 관리

Actor	시스템
Trigger	윷/모 또는 잡기 등이 발생해 **추가 이동 결과**가 생길 때
Main Flow	1) 시스템이 결과를 리스트에 추가해 표시. 2) 플레이어가 결과-적용 대상을 지정할 때마다 결과 소모.
Post-condition	리스트가 비면 차례 종료

⑧ 게임 종료 및 재시작

Actor	시스템/플레이어
Trigger	한 플레이어의 모든 말이 도착 칸에 진입
Main Flow	4) 시스템이 해당 플레이어의 승리를 알린다. 4) **재시작** 또는 **프로그램 종료** 버튼을 제시한다.
Post-condition	선택에 따라 새 게임 초기화 또는 프로그램 종료

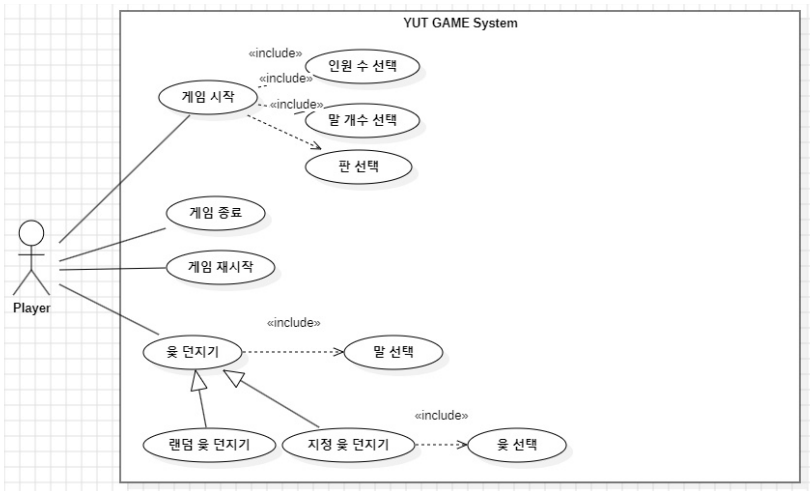
⑨ 판 형태 커스터마이즈

Actor	플레이어
Trigger	새 게임 설정 화면
Main Flow	1) **사각형 / 오각형 / 육각형** 중 하나를 선택한다.
Post-condition	설정된 판 정보가 게임 로직에 반영

⑩ UI 전환(개발/테스트용)

Actor	개발자/테스터
Trigger	UI 전환 옵션 선택 또는 빌드 옵션 변경
Main Flow	① 개발자는 Swing ↔ JavaFX/SWT **등** UI 레이어를 교체한다. ② 시스템은 동일 모델/컨트롤러를 재사용해 실행한다.
Post-condition	두 UI 모두 정상 동작(문서에 변경-지점 기록)

⑪ Use Case Diagram

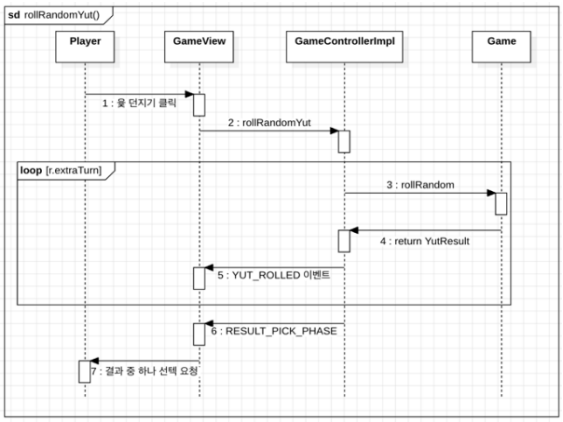


<그림 2>

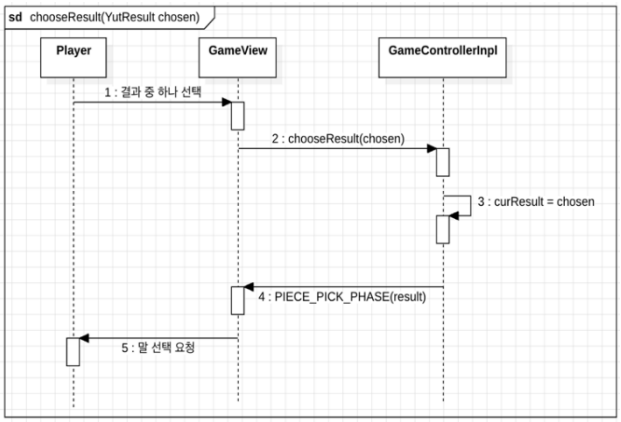
4. 기타 요구사항 분석 (Operation Contract, SSD)

① rollRandomYut()

- Responsibility: 현재 턴의 플레이어가 무작위 윷 결과를 얻고, 필요한 경우 재굴림을 수행
- Precondition: 게임이 시작된 상태이며, 현재 턴이 존재하고 진행 중이어야 함
- Postcondition
  - 결과가 pendingResults 에 추가됨
  - 결과가 extraTurn 일 경우 자동 재굴림 수행
  - 마지막 결과 이후 RESULT_PICK_PHASE 이벤트를 전송하여 플레이어가 결과 중 하나를 선택하도록 함



<그림 3> rollRandomYut()



<그림 4> chooseResult(YutResult chosen)

② chooseResult(YutResult chosen)

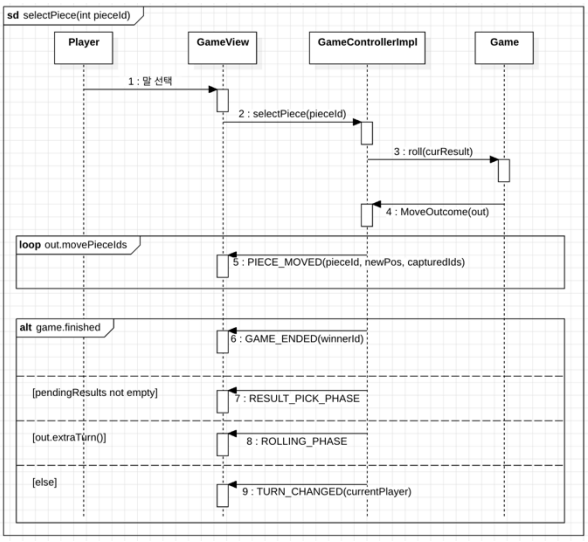
- Responsibility: 플레이어가 다중 윷 결과 중 하나를 선택하면, 해당 결과를 이동에 사용할 수 있도록 설정하고 말 선택 단계로 넘긴다.
- Precondition
  - pendingResults 가 비어 있지 않음

- chosen 값이 pendingResults 에 포함되어 있어야 함

- Postcondition
  - curResult <- chosen
  - chosen 이 pendingResults 에서 제거되지 않음(selectPiece 후 제거)
  - PIECE_PICK_PHASE 이벤트 전송

③ selectPiece(int pieceId)

- Responsibility: 플레이어가 이동할 말을 선택하면, 시스템은 해당 결과를 적용하여 말을 이동시키고, 게임 상태(잡기, 업기, 게임 종료 등)를 처리한다.
- Precondition
  - curResult 가 null 이 아니어야 함
  - 선택된 pieceId 가 현재 플레이어의 말이어야 함
- Postcondition
  - 말이 이동됨 (Game.move(pieceId) 수행)
  - PIECE_MOVED 이벤트 전송됨 (여러 개일 수 있음)
  - 게임 종료 / 결과 선택 / 추가턴 / 턴 종료 중 하나가 상태로 이어짐



<그림 5> selectPiece(int pieceId)

5. 설계 및 구현 리포트

i. 전체 아키텍처 (MVC 구조 기반)

YutGame	app	UIMode(Enum), YutGameLauncherApp
	controller	GameController(Interface), GameControllerImpl, GameEvent, GameEventType(Enum)
	model	Board(Interface), BoardShape(Enum), Game, HexagonBoard, MoveOutcome, PentagonBoard, Piece, Player, SquareBoard, YutResult(Enum)
	ui	GameLauncher, GameLauncherFX

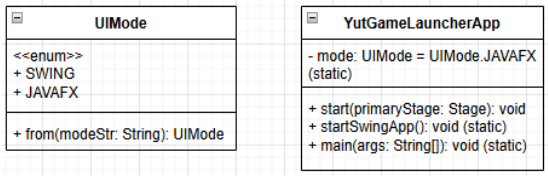
	view	① GameView(Interface) ② javafx: BoardPanelFX, GameBoardViewFX, GameSetUpViewFX, GameCompleteViewFX, PieceViewFX, JavaFXGameViewAdapter ③ swing: BoardPanel, GameBoardView, GameSetUpView, GameCompleteView, PieceView, SwingGameViewAdapter

ii. 클래스 다이어그램

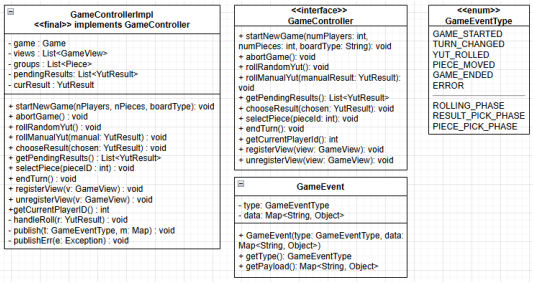
① YutGameLauncherApp: 게임 실행 시점에서 UIMode를 기반으로 적절한 UI 및 컨트롤러를 초기화하는 런처 역할을 수행한다. 이때 UIMode는 Enum 기반으로 정의되어 있어 Swing 또는 JavaFX UI 선택이 가능하며, UI 이외의 컨트롤러/모델 로직은 동일하게 재사용된다.

② GameController(인터페이스): View가 호출하는 모든 게임 기능의 진입점으로서, 게임 시작, 윷 던지기, 결과 선택, 말 선택, 턴 종료 등의 메서드를 가진다.

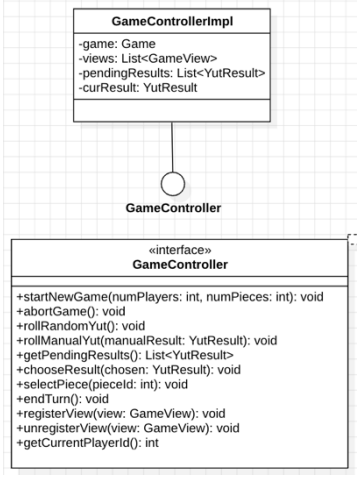
③ GameControllerImpl: GameController 인터페이스를 구현한다. Game 객체를 통해 게임 상태를 조작하며, 여러 GameView를 views로 관리하고 pendingResults, curResults 등의 턴 기반 상태를 관리한다.



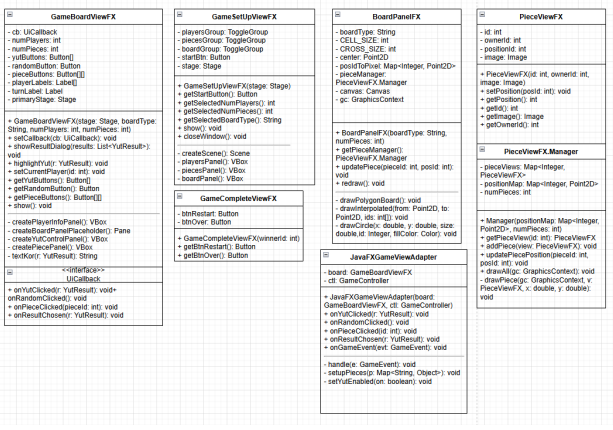
<그림 6> App package



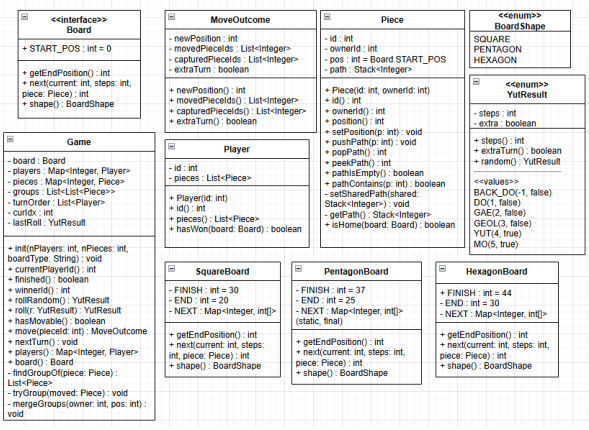
<그림 7> Controller package



<그림 7-1> Controller Class Diagram



<그림 8> View-FX package

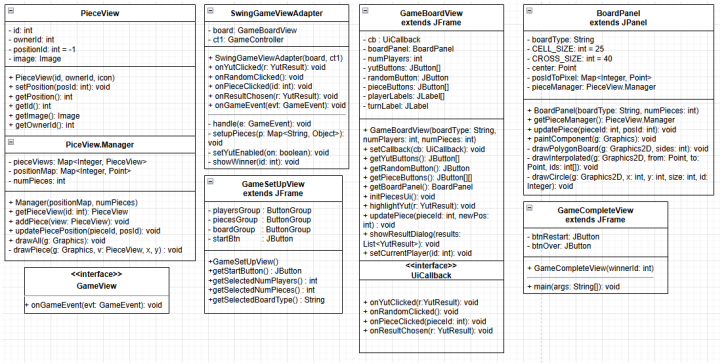


<그림 9> Model package

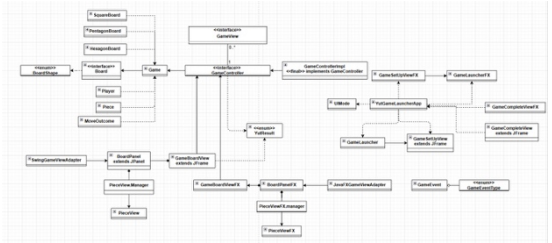
④ Game: 게임의 핵심 로직을 담당한다. 플레이어 리스트와 보드 정보를 보유하며 윷 던지기, 말 이동, 턴 전환, 승리 판단 등의 로직을 수행한다.

⑤ Board (추상 클래스): 게임판의 기본 구조를 정의한다. 모양에 따라 3 개의 서브 클래스(SquareBoard, PentagonBoard, HexagonBoard)가 존재한다. 오버라이팅 메서드 getNextPosition()을 통해 분기 경로를 정리한다.

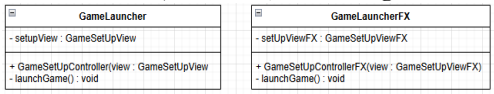
⑥ GameView(인터페이스): UI 로부터 게임 이벤트를 전달받는 관찰자(Observer) 인터페이스이다. 다양한 UI 기술(Swing, JavaFX)에 따라 게임 뷰를 교체할 수 있도록 추상화되어 있으며, 게임의 상태를 시각적으로 표현하는 역할을 담당한다.
초기에는 SwingGameViewAdapter 와 JavaFXGameViewAdapter 가 이 인터페이스를 구현하였으나, **최종 구조에서는 리팩토링 과정을 통해 MVC 패턴에 맞게 기능을 분리하였으며**, 현재는 각 UI 클래스(GameBoardView, GameBoardViewFX)가 GameView 를 직접 구현하여 View 역할을 수행하도록 구조가 단순화되었다. 이를 통해 UI 와 게임 로직 사이의 결합도를 낮추고, 유지보수성과 테스트 용이성이 향상되었다.



<그림 10> View-Swing package



<그림 11> 전체 Class Diagram



<그림 12> UI package

iii. 멀티 UI (Swing, JavaFX) 구현

두 개의 UI 도구를 모두 지원하기 위해 GameView 인터페이스를 중심으로 추상화된 View 구조를 채택하였다.

- **초기 구성에서는**, Swing 기반 UI 클래스(GameSetUpView, PieceView 등)를 중심으로 JavaFX 버전은 이를 재사용하거나 어댑터(JavaFXGameViewAdapter, SwingGameViewAdapter)를 통해 기능을 전달받는 방식으로 설계되었다.
- **최종 구조에서는**, 기존에 분리되어 있던 YutGameLauncher(Swing)와 YutGameLauncherFX(JavaFX)를 하나의 통합 런처 클래스인 YutGameLauncherApp 으로 통합하여 JavaFX 및 Swing UI 각각이 GameView 인터페이스를 **직접 구현**하는 구조로 리팩토링되었다. 이에 따라 각 UI 는 독립적으로 게임 이벤트를 처리하며, 공통 인터페이스를 기반으로 동일한 게임 로직을 공유할 수 있게 되었다.

- 통합된 런처는 프로그램 실행 시 매개변수를 통해 Swing 또는 JavaFX 중 하나를 선택하여 실행할 수 있도록 구성되었으며, 실행 진입점을 일원화하여 배포의 효율성을 향상시켰다.

iv. 게임판 커스터마이즈 구조

게임판은 사용자의 선택에 따라 사각형, 오각형, 육각형 형태로 구성될 수 있도록 설계되었다. 다음은 구조에 대한 설명이다.

- Board 는 추상 클래스로 정의되며, 이를 상속한 SquareBoard, PentagonBoard, HexagonBoard 가 각각 구체 구현체로 존재한다.
- 사용자는 게임 시작 시 UI 를 통해 원하는 BoardShap Enum 을 선택할 수 있으며, GameLauncher(FX)에서 이를 받아 보드를 초기화한다.
- 각 보드는 getNextPosition()을 오버라이딩하여 중심점 분기 처리 로직을 다르게 구현한다. 예를 들어, 오각형 및 육각형 보드에서는 말이 중심점에 멈출 경우와 통과할 경우를 구분하여 종료 지점과 가까운 경로를 자동 선택하게 된다.

이러한 구조는 OCP(개방 폐쇄 원칙)을 잘 따르고 있으며, 새로운 판 형태를 추가하고 싶을 때도 기존 코드 수정을 최소화하고 확장이 가능하다.

v. 핵심 컨트롤러 흐름

- 게임 시작 시 참여자 수와 말 개수를 설정할 수 있음
  → GameSetUpView 와 GameLauncher(FX)를 통해 처리됨
- 모든 플레이어는 각기 다른 말 색상을 부여받고, 개인전 형식으로 플레이함
- <랜덤 윷 던지기> 및 <지정 윷 던지기> 버튼을 제공하여 테스트 및 실제 진행이 가능함 → GamePlayController 에서 버튼 이벤트 처리
- 윷 결과 적용 시, 플레이어는 말 선택 후 자동 진행되며, 다중 결과 시 순차 선택 가능하도록 사용자에게 선택 기회 제공
  → pendingResults, chooseResult(), selectPiece() 등으로 구현
- 업기 기능: 동일한 칸에 같은 플레이어의 말이 도착하면 그룹핑
  → Piece 및 Game 에서 그룹 정보 유지
- 잡기 기능: 다른 플레이어의 말이 위치한 칸에 도착 시 상대 말을 초기 위치로 돌려보냄 → MoveOutcome 및 capturedPieceIds 를 통해 구현
- 말이 모두 골인한 플레이어가 승리하며, GameCompleteView 를 통해 승리 알림 및 재시작/종료 버튼 제공
- 오각형, 육각형 판 커스터마이즈 기능 구현
  → SquareBoard, PentagonBoard, HexagonBoard 클래스 및 BoardShape(Enum)으로 설계
- 오각형/육각형 시 중심점 경로 선택 로직도 반영됨
  → Board.getNextPosition() 오버라이딩으로 구현

6. Junit Test Report

1) GameControllerImplTest

① rollYut_addsPendingResult()
- 목적: 수동 윷 던지기를 호출했을 때 pendingResults 리스트가 1 개 증가하는지 확인
- 내용: 수행 전과 후의 리스트 크기를 비교하고, 결과가 리스트에 포함되었는지 확인
- 결과: 성공적으로 pendingResults 에 결과가 추가됨.

② chooseResult_thenSelectPiece_consumesResultAndMoves()
- 목적: 결과 선택 이후 말을 선택했을 때 말이 이동하고 결과가 제거되는지 테스트
- 내용: 선택한 말의 위치 변경 여부와 결과 리스트에서 제거 여부 확인
- 결과: 말이 전진했고 결과 리스트에서도 제거됨.

③ endTurn_switchesCurrentPlayer()
- 목적: 턴 종료 시 플레이어가 바뀌는지 확인
- 내용: 현재 플레이어 ID 변경 여부를 검증
- 결과: 현재 플레이어가 정상적으로 변경됨

④ selectPiece_withoutChoose_doesNothing()
- 목적: chooseResult 없이 말 선택 시 아무 변화가 없어야 함을 테스트
- 내용: 말의 위치와 pending 리스트의 변동 없음 확인
- 결과: 실제로 말은 이동하지 않고 리스트도 변경되지 않음

2) GameEventTest

① createEvent_withPayload_storesTypeAndData()
- 목적: GameEvent 객체가 타입과 페이러드 데이터를 정상적으로 저장하는지 확인
- 내용: Map 으로 넘긴 값과 객체 내 값이 일치하는지 테스트
- 결과: 정상적으로 저장되었고 get 메서드로 동일하게 반환됨

② createEvent_withoutPayload_allowsNull()
- 목적: payload 가 null 일 때도 객체가 생성되는지 확인
- 내용: null 값 처리 확인
- 결과: 타입은 저장되며 payload 는 null 로 허용됨

3) BoardTest

① startPos_isZero()
- 목적: 시작 위치 상수(Board.START_POS)가 0 인지 확인
- 내용: START_POS == 0 인지 테스트
- 결과; 정상적으로 0 으로 정의됨

② endPosition_isPositive()
- 목적: 도착 지점이 시작보다 큰 양수인지 확인
- 내용: getEndPosition()의 반환값이 START_POS 보다 큰지 확인

- 결과: 음수가 아닌 정상 범위의 위치임이 확인됨

③ next_advancesForward()
- 목적: 말이 두 칸 이동 시 위치 증가 확인(빽도 제외)
- 내용: next() 호출 결과 이전 위치보다 커졌는지 확인
- 결과: 두 칸 전진하여 위치 증가함

4) GameTest

① init_createsPlayersAndPieces()
- 목적: 게임 초기화 시 플레이어 수와 말 수가 정확히 생성되는지 검증
- 내용: 플레이어 수와 각 플레이어의 말 수를 2 개씩으로 설정 후 각각의 개수 확인
- 결과: 2 명 플레이어 생성, 각 플레이어에 2 개 말이 생성됨

② rollRandom_setsLastRollInternally()
- 목적: rollRandom()이 적절한 YutResult 를 반환하는지 확인
- 내용: 반환된 결과가 유효한 범위 내에 있는지 확인
- 결과: 범위 내에 있음을 확인함

③ move_updatesPiecePosition()
- 목적: move() 메서드가 말을 실제로 이동시키는지 확인
- 내용: 이동 전후 말의 위치 비교, 반환된 MoveOutcome 의 이동 ID 목록 확인
- 결과: ID 포함 말 위치 변경됨

④ move_withoutRoll_throws()
- 목적: 결과를 선택하지 않고 말을 이동시키려 할 때 예외가 발생하는지 확인
- 내용: roll() 호출 없이 move() 시도 후 예외 발생 확인
- 결과: IllegalStateException 예외 발생

5) MoveOutcomeTest

① record_fields_areStored()
- 목적: MoveOutcome 의 모든 필드가 생성자 입력값대로 정확히 저장되는지 검증
- 내용: newPosition, movedPieceIds, capturedPieceIds, extraTurn 값을 확인
- 결과: 모든 필드 값이 정확히 저장됨

6) PieceTest

① initial_state()
- 목적: 초기 상태에서 말의 위치가 시작점이고 경로가 비어있는지 확인
- 내용: position 값과 path 상태 확인
- 결과: 시작 위치 0, path 비어있음

② path_push_pop_peek()
- 목적: 말의 이동 경로 stack 이 올바르게 작동하는지 확인
- 내용: push, pop, peek 조작 후 순서 검증
- 결과: 정상 동작

③ isHome_true_afterEnd()
- 목적: 말이 도착점 이후로 이동했을 때 isHome()이 true 를 반환하는지 확인
- 내용: 말 위치를 finish+1 로 설정 후 확인
- 결과: true 반환됨

7) PlayerTest

① hasWon_true_whenAllPiecesHome()
- 목적: 모든 말이 도착점 이후에 있을 때 승리 조건이 충족되는지 확인
- 내용: 각 말의 위치 설정 후 hasWon() 호출
- 결과: true 반환됨

8) SquareBoardTest

① next_oneStep_fromStart()
- 목적: 시작 지점에서 1 칸 이동 시 정상 위치로 가는지 확인
- 내용: next(0, 1)의 반환값 확인
- 결과: 위치 1 로 이동됨

② next_movesToFinish()
- 목적: 끝 직전에서 한 칸 이동 시 FINISH 로 이동하는지 확인
- 내용: 20 → FINISH 위치 확인
- 결과: 도착 지점으로 이동 확인됨

③ next_neverExceedsFinishByMoreThanSteps()
- 목적: 여러 칸을 이동해도 도착 지점을 넘지 않는지 확인
- 내용: 20 회 반복 이동 후 위치 검증
- 결과: 도착 지점 초과 없음

9) YutResultTest

① steps_and_extraTurn_flags()
- 목적: 윷 결과(YutResult)의 이동 칸 수(steps())와 추가 턴 여부(extraTurn()) 속성이 올바르게 동작하는지 검증하기 위함
- 내용:
  - DO 결과의 steps()가 1 인지 확인
  - GAE 는 추가 턴을 주지 않아야 하므로 extraTurn()이 false 인지 확인
  - YUT 은 추가 턴을 주므로 extraTurn()이 true 인지 확인
- 결과: 모든 테스트가 통과하였으며, 각각의 YutResult 상수가 정의한 로직대로 정확히 동작함이 확인됨

10) BoardPanelTest

① testConstructor_initializesCorrectly()
- 목적: BoardPanel 이 정상적으로 채워지는지, piecemanager 가 null 이 아닌지 테스트
- 내용: BoardPanel("사각형", 4)의 생성자 확인, getPieceManager()의 동작 확인

- 결과: 정상 동작

11) GameBoardViewTest

① testInitComponents()
- 목적: GameBoardView 생성 확인, getYutButton 의 null 상태 확인, 윷 버튼 정상 활성화 확인, 버튼이 각 플레이어수, 말 수로 초기화 되었나 확인
- 내용: GameBoardView("사각형", 2, 2)의 생성자 확인, getYutButtons 의 null 상태 확인, 지정 윷 던지기 버튼 6 개(도~백도)와 랜덤 윷 던지기 버튼(getRandomYutButton) 활성화 상태 확인, 생성자에서 지정한 플레이어와 말의 수 2, 2 확인
- 결과: 정상 동작

② testFrameOpen()
- 목적: GameBoardView 호출 후 오류없이 화면 출력 되는지 확인
- 내용: GameBoardView 객체 생성 후 setVisible()이용하여 동작 확인
- 결과: 정상 동작

12) GameCompleteViewTest()

① setUp(), ② tearDown() (Swing Test)
- 목적: swing view 테스트를 위한 더미 view 생성

③ testButtonsAreCreated() (Swing Test)
- 목적: GameCompleteView 에서 생성된 게임 재시작 및 종료 버튼의 null 상태, 각 버튼에 해당하는 문구 표시 확인
- 내용: 각 버튼의 NotNull 상태 확인, "게임 재시작", "게임 종료" 문구 정상 출력 확인
- 결과: 정상 동작

④ initToolkit(), ⑤ FXsetUp() (JavaFX Test)
- 목적: JavaFX view 테스트를 위한 환경 초기화 및 더미 view 생성

⑥ testButtonsAreCreatedAndLabeledCorrectly() (JavaFX Test)
- 목적: ③과 동일한 목적을 JavaFX 에서도 확인
- 내용: ③과 동일한 내용을 JavaFX 에서도 확인
- 결과: 정상 동작

13) GameSetUpViewTest

① testComponentInitialization() (Swing Test)
- 목적: GameSetUpView 의 startButton 의 null 상태, 출력 문구 확인
- 내용: startButton 의 null 상태, 문구 "시작하기" 정상 출력 확인
- 결과: 정상 동작

② testSwingStartButtonActionListener() (Swing Test)
- 목적: GameSetUpView 의 starButton 의 eventListener 의 정상 동작 확인
- 내용: ActionListner 를 동작해 버튼 클릭시 동작을 확인
- 결과: 정상 동작

③ testSwingDefaultSelections() (Swing Test)

- 목적: GameSetUpView 의 기본값 확인
- 내용: GameSetUpView 의 기본값이 2(플레이어 수), 2(말의 수), 사각형(보드)로 설정되어있는지 확인
- 결과: 정상 동작

④ testFXStartButtonInitialization() (JavaFX Test)
- 목적: ①과 동일한 목적을 JavaFX 에서도 확인
- 내용: ①과 동일한 내용을 JavaFX 에서도 확인
- 결과: 정상 동작

⑤ testFXDefaultSelections() (JavaFX Test)
- 목적: ③과 동일한 목적을 JavaFX 에서도 확인
- 내용: ③과 동일한 내용을 JavaFX 에서도 확인
- 결과: 정상 동작

14) PieceViewTest

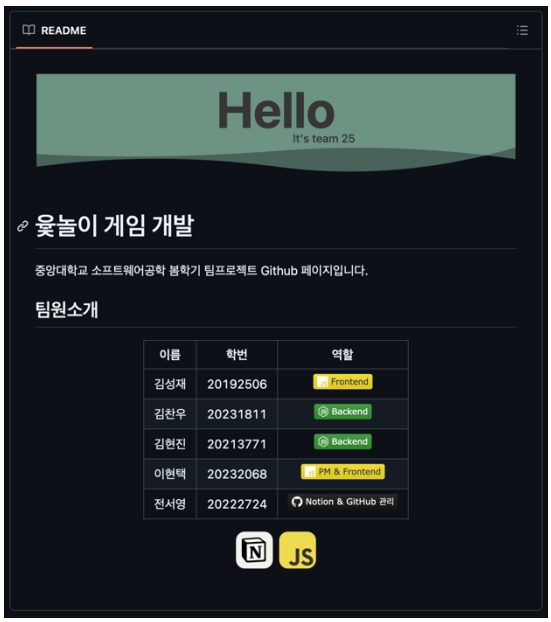
① testSwingPieceViewInitialization() (Swing Test)
- 목적: piece 의 기본 설정 확인
- 내용: id(1), ownerId(2), img, getPosition()으로 piece 의 초기위치 -1 확인
- 결과: 정상 동작

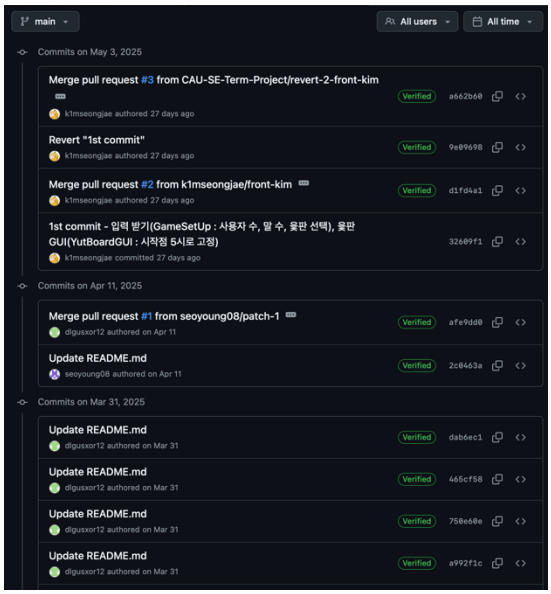
② testSwingSetAndGetPosition() (Swing Test)
- 목적: piece 객체의 동작 확인
- 내용: piece(1, 2, new ImageIcon) 객체의 setPosition(5)과 getPosition()으로 5 를 반환하는지 확인
- 결과: 정상 동작

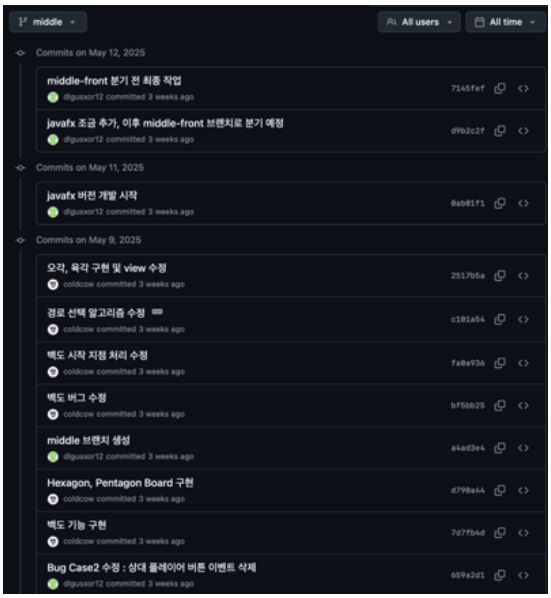
③ testFXPieceViewInitialization() (JavaFX Test)
- 목적: ①과 동일한 목적을 JavaFX 에서도 확인
- 내용: ①과 동일한 내용을 JavaFX 에서도 확인
- 결과: 정상 동작

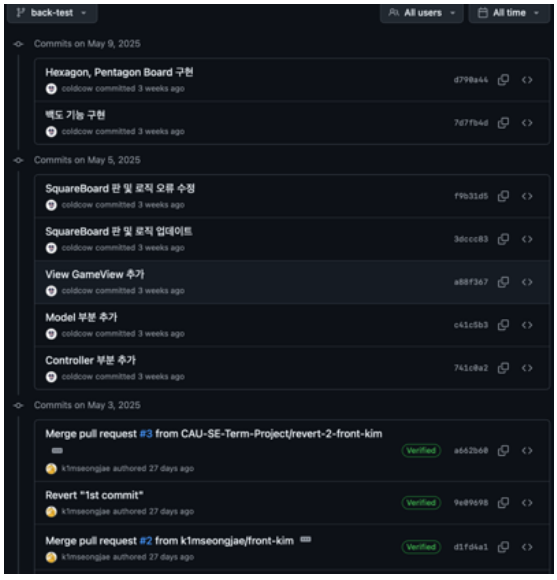
④ testFXSetAndGetPosition() (JavaFX Test)
- 목적: ②과 동일한 목적을 JavaFX 에서도 확인
- 내용: ②과 동일한 내용을 JavaFX 에서도 확인
- 결과: 정상 동작

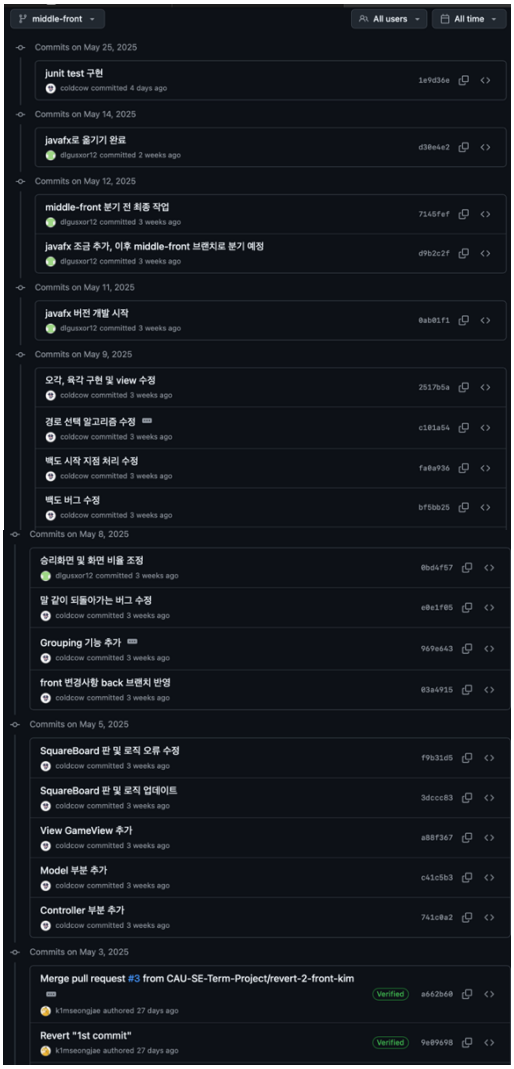
7. GitHub Project Report
   - Github link: https://github.com/CAU-SE-Term-Project/SE_Term_project
   - Project Progress History

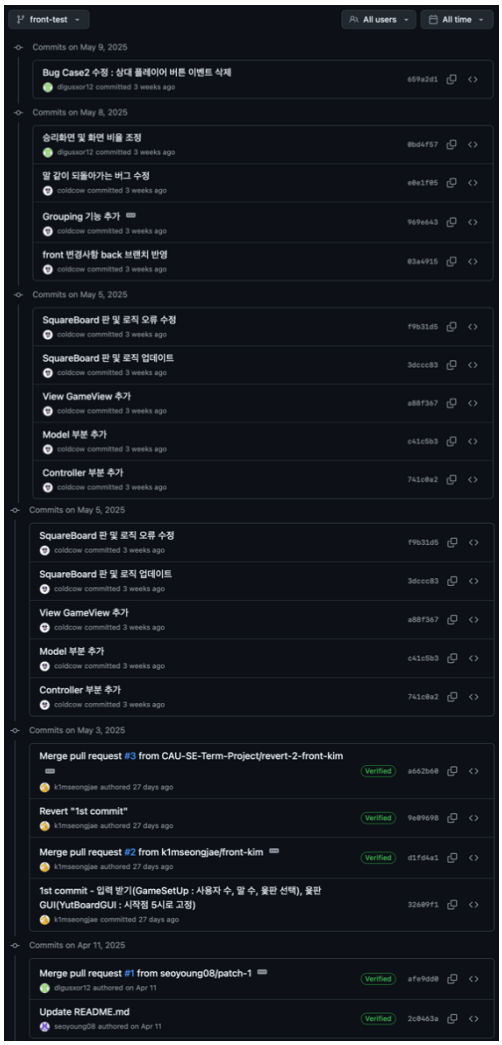












- 팀원 별 역할:

김성재(20192506): 프론트엔드
김찬우(20231811): 백엔드
김현진(20213771): 백엔드
이현택(20232068): PM & 프론트엔드
전서영(20222724): 노션 & 깃허브 관리 및 보고서, 발표자료 준비