# Refactoring List-Group 7

(We write tests for reinforce, startup and fortify phase before refactoring)

## Removal of duplicated showCommands methods

In build 1, showCommands such as showPlayer(), showAllPlayer(), showMap() exist in all controllers including gameController, startUpGameController, reinforceGameController and fortifyGameController.

* A screenshot of a cell phone

  Description automatically generated
* In build 2, we extract all showCommand methods and make them a static method in the MapDisplayUtils.class to avoid duplication and increase code readability and understandability.

A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generated

## MOVE PLAYER STATES AND CORRESPONDING LOGICS FROM GAMECONTROLLER TO A MODEL PlayerService.class

* In build 1, player states and corresponding logics are in gameController.
* A screenshot of a cell phone

  Description automatically generated
* In build 2, we move player state and corresponding logic in model named PlayerService.class and make that new model class Observable.
* A screenshot of a cell phone

  Description automatically generatedA screenshot of a cell phone

  Description automatically generatedA screenshot of a cell phone

  Description automatically generatedA screenshot of a cell phone

  Description automatically generated

Build 1: MapService only observable model. Consequently, too much overhead for controllers as they had to implement the validation, game logic as well as notify views of changes taking place in the game when players were concerned.

Build 2: PlayerService.class Observable as well. Consequently, many changes that occur to player states notified by model directly to views. Most probably a GUI (multiple views) will be implemented for build 3. Controllers will not have to notify each view. (Less task)

## REMOVAL OF GAME CONTROLLER

In Build 1, there were unnecessary dependencies on Game controller by other controllers as it contained all the player states. Having this additional controller layer also created additional overhead.

The game controller used to control game flow as well in other controllers and had many unnecessary values passed as parameters (Atomic Boolean) to other controllers.

This is not good design.

In Build 2, the Game controller has been removed and every controller is responsible to control its own game flow now. This has also been possible because Player states are now implemented in a model (PlayerService.class) and can be accessed by any controller without having to rely on others.

Unnecessary Boolean values SHARED between controllers and long sections of code to control gameflow has been removed consequently.

The game flow logic and code is now more simplified (readability + understandability)



A screenshot of a cell phone

Description automatically generated

* A screenshot of a cell phone

  Description automatically generated

## Reinforcement, Attack and Fortify game controllers used for validation only now. All logic implemented in model.

In build 1, controllers implemented validation of commands as well as Game logic.

In build 2, Game logic for reinforcement, attack and fortification has now been moved to model.

Controllers only validate commands being sent to them now and invoke different methods in the model.

Any information about state change is directly notified to the view by the model.

However, controllers still inform the view about validation outcomes (such as invalid commands).

(Build 1: reinforcement logic in controller)

* A screenshot of a cell phone

  Description automatically generated
* (Build 2: reinforcement logic in player model)
* A close up of a logo

  Description automatically generatedA screenshot of a cell phone

  Description automatically generatedA screenshot of a cell phone

  Description automatically generatedA screenshot of a cell phone

  Description automatically generatedA screenshot of a cell phone

  Description automatically generated

Build 1 (Fortification logic in fortification controller)

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

## Controller and GameView Interface added as well as 3 different views have been implemented

In build 1, we only had CommandPromptView to receive users’ commands, and send it to different controllers according to the game state.

All information sent by all controllers and by model was handled and displayed by commandPromptView.

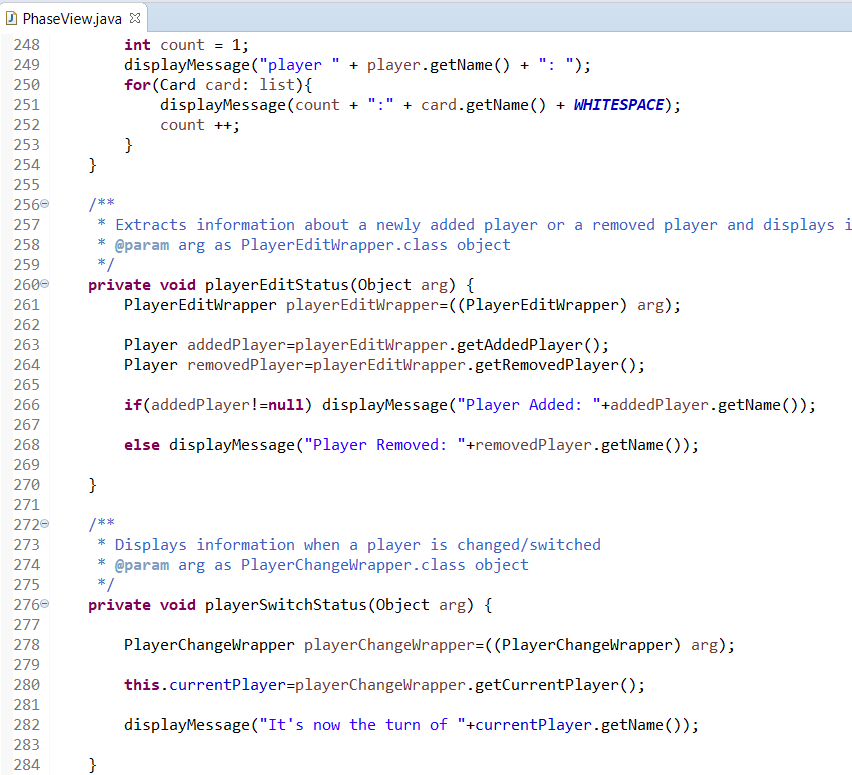
A screenshot of a cell phone

Description automatically generated

* In build 2, we have three different views, phaseView, dominationView and exchangeCardView. We make these three views implement GameViewInterface.
* This makes the system more flexible as we can replace these views in the future. The same case is for the controllers, we will be able to replace these controllers easily in the future based on changing requirements.
* We plan to implement a GUI in Build 3. Hence making the views more modular and making them able to interpret and display the same information differently is crucial here.
* A screenshot of a cell phone

  Description automatically generated
* A screenshot of a cell phone

  Description automatically generated
* A screenshot of a cell phone

  Description automatically generated
* 

## Refactored previous tests; and added more tests BEFORE ACTUAL REFACTORING.

In build 1, the mapLoader Test is hard to read, understand and maintain. Difficult to add more tests to the existing structure.

* A screenshot of a cell phone

  Description automatically generated

In build 2, we extract logics and parameters in different methods to make tests more readable and understandable.

We added more tests to test startup, reinforcement and fortification as well before any actual refactoring.

* A screenshot of a cell phone

  Description automatically generatedA screenshot of a cell phone

  Description automatically generated
* A screenshot of a cell phone

  Description automatically generated

Other Potential Refactoring (Yet to be done)

### Use streams everywhere and uniformly throughout the project instead of traditional for loops.

Streams improve performance and are usually shorter and more readable and hence understandable.

We think using streams will be beneficial to this project as many controllers and model classes implementing the main game logic are very long and hence tedious to understand.

(Overlooked because only 1 member had profound knowledge of streams which has a pretty steep learning curve and requires time to fully master.)

We however consider to use streams more in build 3, especially if we will be implementing a GUI part.

## Use Design Patterns such as Decorator

We have used a lot of additional classes to send information to observers.

These classes have many common fields (eg: PlayerEditWrapper, PlayerAttackWrapper, etc)

We planned to use Decorator patterns to dynamically change structure of objects sent to different views at runtime.