My design rationale includes the following aspects:

1. Design for code reuse: I used polymorphism when I designed the classes SpecialTile. SpecialTile has subclasses for each kind of special tile in this game. I also used delegation in many functionalities of classes like GameSystem and GameBoard.

2. Design patterns: I followed observer design pattern and composite design pattern for GUI; I followed polymorphism and factory method pattern in the core to get a certain type of SpecialTile.

3. Assigning responsibilities:

- I contained the responsibilities of each entity of the game within its own class so that I don't spread out the functionalities to other classes. For example, the SpecialTile class has an 'activateFunc' method to maintain low representational gap. No methods of other classes need to be called to complete SpecialTile's job. Similarly, I kept a list of Squares as a private field of GameBoard to stand for the squares on the board. I kept a list of LetterTiles as a private field of TileBag class to stand for the letter tiles contained in the tile bag. Moreover, on comparing object model to the domain model, there is no deviation with respect to relationships and associations between the classes.
- At the same time, I aim for high cohesion by keeping 'rack' and 'tileInventory' as private fields in Player class to keep each player's tiles stored in the same object; for the same reason, I kept players, dictionary, specialTiles, and tileBag as elements of GameSystem.
- Although the class GameSystem acts as the controller in my entire system, to maintain a low coupling I passed on some functionalities/ responsibilities such as "Calculate Score" to the GameBoard class to reduce the dependency on the GameSystem class, and also because GameBoard has by itself all the information needed to calculate the scores for the player.
- Aiming for good code readability of the core implementation, I created several private methods in those heavy-responsibility classes such as ScrabbleImpl and GameBoard. Also, for low conceptional weight, I tried to avoid putting too many responsibilities all to the ScrabbleImpl class by dividing/assigning them to the other classes which have the information needed to fulfill the responsibility. For the same reason, instead of creating different subclasses extending the abstract class Square, I designed the class Square to have instance variables to determine which kind of square it is.
- I employed the ScrabbleImpl as the controller and information expert throughout the implementation of the game, which coordinates and delegates work to other objects/classes that have the information needed.