My design rationale includes several aspects:

1. Design for code reuse: I used polymorphism and inheritance when I designed the classes SpecialTile and Square. 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 template method pattern and polymorphism design for code reuse. For GUI, I will use observer pattern.
3. Assigning responsibilities:
4. 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 'Square' as a private field of GameBoard to stand for the squares on the board. Moreover, on comparing object model to the domain model, there is no deviation with respect to relationships and associations between the classes
5. 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.
6. 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.