- Changes from Milestone A to Milestone B:

1) Instead of representing the tile bag as member field using a List<LetterTile> in GameSystem, I created the class TileBag with the constructor method to create and initialize the 98 LetterTiles contained in the tile bag. The reason for this is to reduce the conceptional weight of the class GameSystem and improve readability. Also, by initializing and keeping the LetterTiles in the TileBag rather than the GameSystem, I aim for low representational gap and high cohesion.
2) For SpecialTile interface, I changed the parameters of the method activeFunc() from (GameBoard board, GameSystem game) to (Integer index, GameSystem game) to facilitate the implementation of Boom and NegativePoints, because the function activation of these two tiles will require the information of the triggering location of the special tile. Also, since the GameSystem is the controller class which coordinates and delegates work to other objects/classes, it has access to other classes like GameBoard and Player, I removed the other argument GameBoard board for the activeFunc().
3) Instead of creating different subclasses extending the abstract class Square, I changed the Square to be an ordinary class which has several instance variables to determine which kind of square(like premium or not?) it is. This will largely reduce the conceptional weight of the core.
4) I created some private method in main classes like GameSystem, GameBoard and Player to facilitate the implementation of the game.

- Changes from Milestone B to Milestone C:

1) I created the GameChangeListener interface and added addGameChangeListener(…) and removeGameChangeListener(…) and several “notifyGameChanges()” methods to the class GameSystem(now renamed as ScrabbleImpl), following the observer design pattern, to make the core independent of GUI while GUI dependent on the core, so that once the core changes, all its dependents (different GameChangeListeners or GUIs) are updated correspondingly and automatically. This is also consistent with the ‘view’ part of the MVC model.
2) I created a new class SpecialTileFactory to represent the special tile store of the game following the factory method pattern. The method getSpecialTile(String tileType) enables the user to create SpecialTile objects without specifying the exact subclass of the SpecialTile that will be created. Meanwhile, for the sake of object reuse, I just created five objects representing different kinds of SpecialTiles. Whenever the user want to get a SpecialTile of the same type, the same object/ reference is returned.
3) I changed the method getScoreForMainWord(…) in the GameBoard class to getScoreForNegativeWords(…) due to my renewed understanding of the function of NegativePoints special tile.
4) I added the overridden toString() methods to a lot of substantial classes to facilitate the use of the string representation of these classes in GUI.

5) I added a few private methods to the core classes to enhance readability and to reduce the conceptional weight of the public methods.

- The effort required for the additional special tile is just creating another class which implements the SpecialTile interface and overriding the activateFunc(…), getPrice() and toString() methods.