Omar Syed
15-214
Section
October 9, 2013

Homework 4 Rationale

The basic design choice for this implementation of Scrabble was to have one major class called Game that would interact with the GUI and facilitate the processing of the data of all the other classes. This way each class would have control over the respective information of its domain without much knowledge about what the game actually does. That information is what the Game class is responsible for. This also facilitates code reuse because the other components besides the class Game are not Scrabble specific. If some other type of game needs to hold information the way scrabble does, then many of the classes like Board, Player, BoardSpace, Disctionary, and Hand can be reused.

In the UML diagram, underneath Game are four classes, CurrentMove, Dictionary, BoardSpace, TileBag, and Player. If Scrabble was stripped to ts main components, it would be those classes. The Player has a score, name, and hand. It can remove tiles from its hand, add tiles to its hand and update its score. The Board is an array of BoardSpaces and it adds Tiles to those BoardSpaces. BoardSpace actually has two fields for holding tiles, they are viewableTile and actualTile. The reason for this is so that it can be easy to hide SpecialTiles from players who aren't the owners of them. Even if there is a SpecialTile in the actualTile, the viewableTile can be null for a player so that they naively play tiles on top only to find out that when they actually submit, there is a SpecialTile underneath. Tiles are bought from the TileBag class, which creates the tiles and makes sure the number of each type of letter is adhered to. When Tiles are exchanged, the TileBag updates the count of how many of that kind of letter it can create and return.

When players add tiles onto the board, the tile is also added to the currentMove, so that tiles can be added and taken back within the same turn and kept track of to submit a move. When the submit move button is clicked, the Game class gives the currentMove to the Board so it can process a word out of it. When

the word, a string of letters is returned, the Game checks if it is in the dictionary and if it is then it deactivates the modifiers on the tile and adds the score to the player. If the word was not valid then nothing in the game changes but a popup comes up saying that the word needs to be changed. When the currentMove is valid, then the currentWord is cleared and the modifiers on those letters are deactivated and the currentPlayer changes to the next person in the game.

When a player changes, the hand shown by the GUI would also change. The hand is an ArrayList of tiles so that a player can only have seven letters but can have as many specialTiles that they buy. Since both SpecialTiles and LetterTiles can go in a Hand or BoardSpace, they are put under the same abstract class, Tile. The Board class can call SpecialTile's effects when a move is submitted and it is transferring the viewableTile to the actualTile. If there is a specialTile in the actualTile field at this point, then the effect will be called. The doEffect method takes in a board and game so that it can act upon them. Modifiers like a TileModifier or Word Modifier can also be added to a BoardSpace and they take effect when a Word is being processed. They are added to a Word as a whole if they are WordModifiers, or they take effect on a LetterTiles score if they are TileModifiers. During this time, the modifiers are also added to a list of used modifiers in the CurrentMove class which then deactivates them all at the end of a turn.

During the gameplay, if a player cannot make a move, he or she can exchange the tiles in the hand or simply skip a turn. If there are no tiles left in the TileBag, players cannot exchange and they are forced to skip their turn if they can't play anything. If all players skip their turn then the game ends because no one could do anything and the board did not change.