Firstly, I would like to explain my functional distribution of different classes. Basically, the Game class contains functions that directly interact with users(external players) and other main classes. I t works like a project manager which assign work to different departments.

And the Board class is closely linked with the game class, which contain all the location information for the game board(which is a matrix of Location) and a bunch of boolean functions. So each time when player pick and place a tile onto the board, it will automatically check the validation of those feature.

Then the Player class contain the information of each player's name and id, name attribute is used for GUI and terminal to display and id is for deciding the play order. And also the Player class will check if the user input is valid by taking a list of tile.

So the Tile class doesn't manipulate anything(so far), it takes character and their corresponding scores as attributes and it has been store in various classes, for example, Player, Location.

Dictionary class is a part of Game, and it simply check if the challenge location(tiles) ever in the dictionary.

So Location class is the base unit of Board class, and it stores all the location information. It also contain Tile and SpecialTile object.

Each time a player finish his/her move, Word class will store the location and tile he/she made and pass it to checkRow(Location) and checkCol(Location) method to check if this move trigger a special tile or not.

And additionally, for the Square class, I am using the decorator pattern. Because square class will have two properties(double and triple) and since it associate with Location class which contain information of specialTile and I will add more properties for the special tile, it would be more flexible to handle.

And SpecialTile is an abstract class that represent different properties of their subclass. And for the customized specialTile, I was planning to implement function called "addtionOrder", which means when the specialTile has been triggered, the owner of it will get an additional chance to play move and after that, the order is the same.

Then, in terms of the design decisions, basically, I started thinking as a MVC model. Game works as controller, Board is the model and further implements of GUI is the viewer. Each time users can directly interact with function in game and when they need get certain locations, the board will invoke method in location class. So as I mentioned in the Board class before, the reason why I separated board and location into two classes is because, in my mind, board handle more general issues such as check invalid movement and modified scores, but when users need return location or more special information, location be invoked. So location class works more like a storage and give access to other classes.

Finally for the challenge scenario, for my design version, each player can only challenge the last move. When a player click the “challenge” button, the system will automatically get the location of that word and pass it to the Dictionary. Then if it contains, the player who made that challenge will be pass as a parameter of the skipPlayer method, i not, the system will modify the scores by invoke corresponding method in Board and Square and then store in the Player scope.