Design Goal:

The goal of design is to realize a variant of Scrabble in this case. Scrabble is a word game involving 2~4 players. These players place certain number of letter tiles on the game-board which is divided into a 15×15 grid of squares to gain score points. Especially, this kind of Scrabble game includes special tiles.

Design Pattern:

1.Strategy pattern

To fulfill the design principle — design for change, I apply strategy pattern in my design. I build a interfaces SpecialTile, which can be implemented by different kind of special tiles. In the future, if I want to add a additional special tile, I will implement the same SpecialTile interface without influence on others, which make my design extensible. Also, I have a Timer interface which is change the letter or word score, like double letter score, double word score, in certain square on the board. I just implement the Timer interface with different change features. And one day, if the features change, I will add some features easily.

Design Heuristic:

GRASP pattern

Controller:
The ScrabbleGame class behaves like a controller. The ScrabbleGame class is like a coordinator, which does not do much work but delegates assignments to other special classes, like Board class, LetterBag class, Dictionary class, etc. It can also receive the message from the GUI and then notify the related Player class. In this way, it decrease the coupling between different classes. Most classes are coupled to controller, which is a smaller and stable class.

Information expert:
I apply this principle to reasonably assign responsibilities to different class who has the information necessary to fulfill a certain responsibility. It is like that I decide Board to check to validity of a move and calculate the move score of a player. Because the board has all information of all Squares, which is the key to check validity and calculate score. Besides, a Dictionary class designs to check the validity of a word and a Word class calculates its own value.

Design Principle:

1. Low representational gap

I create software class for each class in domain model and create corresponding relationships, which fulfill the low representational gap principle. In this way, we gap the different between software objects and real world objects, so that make a application easy to read.

2. Information hiding

I set every elements in field as “private”, and try to set some methods easily to change as “private” or use a stable interface.

3. Design for change

The strategy pattern make the design to fulfill this principle. I can add any special tiles by implementing the SpecialTile interface as I like.

4. Design for reuse

Every class has own methods and other class uses these methods by delegation, so that I can reuse the code.

5. Low coupling and high cohesion

Low coupling:
The ScrabbleGame is a controller, and all other classes couple to it. Cause ScrabbleGame is a smaller and stable class, this coupling is less harmful. At the same time, ScrabbleGame serves as a coordinator to mediate the communication among different classes. As a result, these classes have low coupling with each other.

High cohesion:
I break the program into classes. Every class has a set of related responsibility. For example, I separate Board from Game and Move from Player. In this way, Board can be delegated to complete some responsibilities within the scope of duty so that ScrabbleGame can maintain a controller role for communicating information. Besides, the Move class is used to store information which should belong to a Player. But I separate them in order to improve cohesion level. Move is used to store information and calculate score. A player own a Move.