***The changes to the API design and Test cases***

***Group 36***

* **Changes in API design**

1. We add one parameter “int turn” to the listen method in ManualKBL class. The reason for the change is that we want the manual page to return to the previous player. In that case, if one player wants to check the manual, when he/she returns to the game page, it will still be his/her turn. We add this extra parameter with the aim of not changing a lot of existing codes. Basically, if we do not modify this listen method in Manual class, we may have to change the whole game logic, which is time-consuming and complicated. There is another method to avoid such revision, we can change turn to a global variable so that we may do not have to pass the turn parameter.
2. We removed Warning class in the View part. The reason for the change is that we have found out that always importing Warning class to print out a user warning will make the code complicated and hard to understand. We inserted the warning information into GameKBL class, which helped us to simplify our code.
3. We removed setRank and setGroup method in Piece class. The reason for the change is that we found out that these methods will not be used outside of the Piece class. Therefore, we can just set the rank and group using “this” keyword, which makes our code simpler and more straightforward.

* **Changes in Test Cases**

1. We chang some assertSame methods into assertEquals methods. The reason for the change is that assertSame is used to check whether two objects refer to the same object and it is not suitable for our case.
2. We add more test cases in PieceTest. The reason is that when we implemented these cases one month ago, we only checked some simple cases. For example, we only compare the rank of each piece to decide whether a piece can be eaten. However, the situation can be complicated, we must include some cases such as piece in the trap, or piece in the river.

To avoid such revision, we think that we had better discuss the implementation detailly before we begin to write the code. In that case, we may face fewer conflicts. We have learned an important lesson that to avoid a huge amount of changes in the future, we must take every situation into account before we actually do it. When everything is clear, then the implementation part can go smoothly.