David Giacobbi
CPSC 224: HW4 Summary

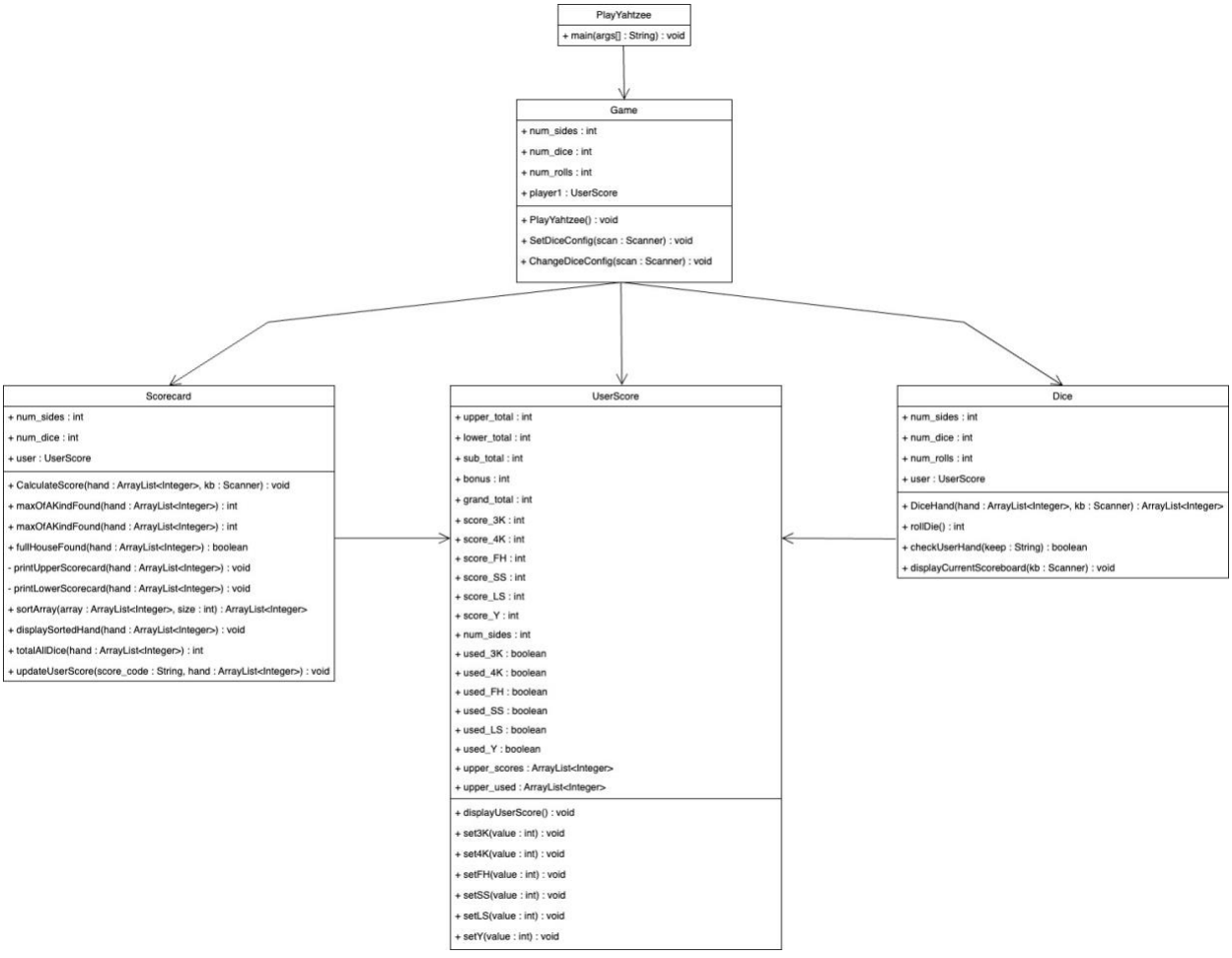
Summary:

The overall purpose of this development in the Yahtzee program was to generate a dynamic scorecard for the single player so that they can keep track of their scores as rounds progress. Moreover, the introduction of a user score display would imply that the player needs to pick a certain scoring line after each round. Once this scoring line is chosen, however, the line must be removed from the possibilities offered at the end of every turn.

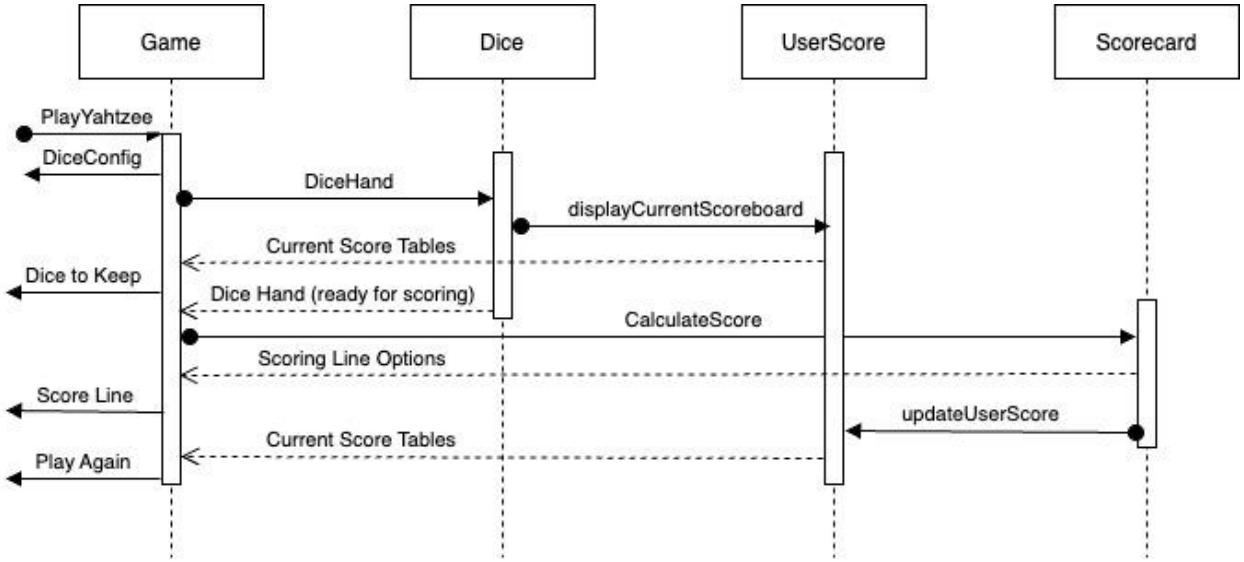
Overview of General Design:

The overall design of this addition was to create another object called UserScore. In this class, all attributes related to the user's scoring lines would be stored in this object. Scorecard and Dice could then call methods from this class to set the scores as they are picked. Moreover, these objects could check to see if the scoring line has been used already or not. Lastly, UserScore would contain the method that would print the dynamic scorecard to the screen at request.

UML Class Diagram:



UML Sequence Diagram:



Design/Programming Issues:

One major programming issue was trying to decide how to deal with the upper scores. Given that they are based on a dice attribute that can change at the beginning of any game, it was hard to determine how to allocate enough space to not only track the scores but also check if they have been used or not. The solution was to use ArrayLists to keep track of both these parameters. Because of this, the UserScore needed to take in the number of dice sides into its constructor. With that, it was quite easy to resolve any upper scoring issues. Another issue that I ran into was making sure that used scoring lines were checked before displaying as well as the value of that score. If-statement checking was used to resolve this matter, and the various scoring check methods in Scorecard aided me in fixing this issue.

Retrospective:

Looking back, if I had more time, I would have loved to add more input and error checking. At the moment, user-input methods are very fragile and can exit the program if the wrong code or character is entered. Next project, I intend to do a deep examine into the validity checks that are necessary to make this program more robust and useful. Secondly, I would like to continue to add unit tests as this program continues to expand in detail and volume, ensuring that I do not have to back track far when a method is not performing the way I would like it to.