Benjamin Bladow, Brandon Niblock, Eugene Krug

Gonzaga Men's Basketball Yahtzee
Design Alternative Analysis

Design Alternative Analysis 1

The issue: The first problem our group faced with the project is how to incorporate each group member's code from Programming Assignment 6 into a foundation for the group project. Each group member had a successful Programming Assignment 6, so we considered everyone's code. We mainly focused our decision on class design structure and studied each group member's UML diagram. We realized Eugene and Brandon had similar class design structures so for the sake of making a decision we thought of their assignments as one. We evaluated three alternatives.

Available alternatives

- Alternative 1: Keep all of Benjamin's code. There are many methods per class in his design.
- Alternative 2: Keep all of Brandon's code. Brandon's code has fewer methods and each method does more work in comparison to Benjamin's code. Eugene's class design structure is similar to Brandon's and will not be evaluated.
- Alternative 3: Combine the best aspects of Benjamin's and Brandon's class structure.

First alternative

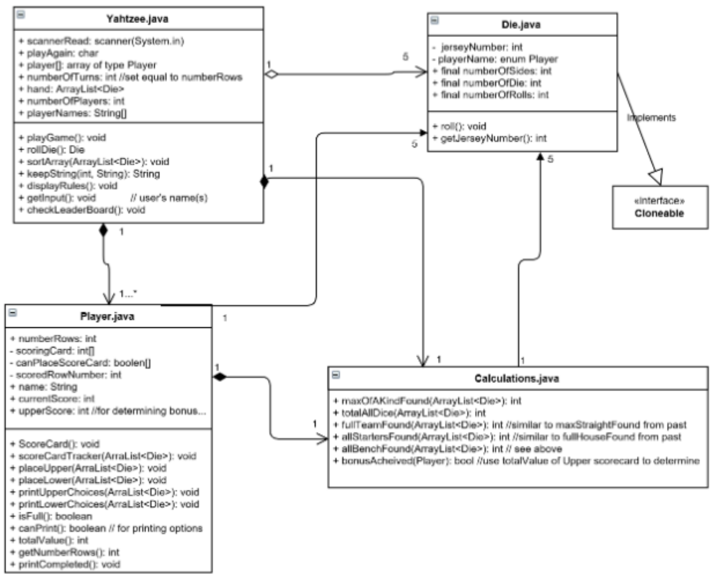
- In this design, we have four classes: Calculations, Player, Die, and Yahtzee. Each of the classes contains many methods.

Pros

- Since there are many methods in each of the classes the methods do less (in theory, one task). This modularized form of programming generally makes the code easier to read and debug.
- Code can be adapted for multiple users much easier.

Cons

- The design could be considered more complicated because there seems to be more "going on".
- The comments could be improved.


Yahtzee.java
+ scannerRead: scanner(System.in)
+ playAgain: char
+ player[]: array of type Player
+ numberOfTurns: int //set equal to numberRows
+ hand: ArrayList<Die>
+ numberOfPlayers: int
+ playerNames: String[]
+ playGame(): void
+ rollDie(): Die
+ sortArray(ArrayList<Die>): void
+ keepString(int, String): String
+ displayRules(): void
+ getInput(): void // user's name(s)
+ checkLeaderBoard(): void
Die.java
- jerseyNumber: int
- playerName: enum Player
+ final numberOfSides: int
+ final numberOfDie: int
+ final numberOfRolls: int
+ roll(): void
+ getJerseyNumber(): int
Implements
«interface»
Cloneable
Player.java
+ numberRows: int
- scoringCard: int[]
- canPlaceScoreCard: boolen[]
- scoredRowNumber: int
+ name: String
+ currentScore: int
+ upperScore: int //for determining bonus...
+ ScoreCard(): void
+ scoreCardTracker(ArraList<Die>): void
+ placeUpper(ArraList<Die>): void
+ placeLower(ArraList<Die>): void
+ printUpperChoices(ArraList<Die>): void
+ printLowerChoices(ArraList<Die>): void
+ isFull(): boolean
+ canPrint(): boolean // for printing options
+ totalValue(): int
+ getNumberRows(): int
+ printCompleted(): void
Calculations.java
+ maxOfAKindFound(ArrayList<Die>): int
+ totalAllDice(ArrayList<Die>): int
+ fullTeamFound(ArrayList<Die>): int //similar to maxStraightFound from past
+ allStartersFound(ArrayList<Die>): int //similar to fullHouseFound from past
+ allBenchFound(ArrayList<Die>): int // see above
+ bonusAcheived(Player): bool //use totalValue of Upper scorecard to determine
1
5
1
5
5
1
1...*
1
1
1
1
1

Second alternative

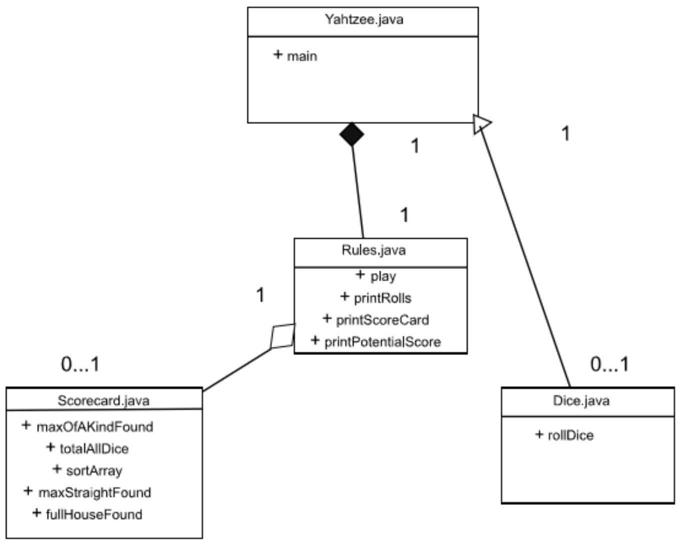
- In this class design, we have four classes: Dice, Scorecard, Rules, and Yahtzee.

Pros

- The sorting method is simpler because it does not involve cloneable.
- When looking at the UML Class Diagram, this design appears to be simpler than the first alternative.

Cons

- The code is more difficult to read and debug since each method does more than one task.
- Many of the methods are static.
- There are very few private variables with getter and setter functions.



Third alternative

- In this design, we combine Benjamin's and Brandon's Programming Assignment 6 to use as a foundation for the group project.

Pros

- Combines the best aspects of both assignments.

Cons

- No group member is an immediate expert of the code.

Decision: Our group elected to utilize the first alternative. The most significant benefit of the first alternative is that the code adapts with the most ease to a multi-user game. We believe the third alternative would be time-intensive to implement considering variable names, method names, etc. would not match up. Benjamin briefed the entire group on his Programming Assignment 6 code, so that all group members feel comfortable with it and can work with it moving forward.

Design Alternative Analysis 2

The issue: The problem that our group is encountering is multiuser. Our code has a specific player (user) class, which means that we can easily implement more than one user playing the game at a time. When playing a game of Yahtzee in real life, we usually play the game one round at a time, however when we were looking at our code, it works most efficiently when we play the game one user at a time, and then clearing the table, and playing again with the next user.

Available alternatives

- Alternative 1: Users play at the same time
- Alternative 2: Users play at different times

First alternative

- In this design, we have the users play at the same time. Figuratively, one user does their turn and then "hands over" the dice to the next user so that they can do their turn.

Pros

- This option allows the game to finish much quicker, as each user will be allowed to go one at a time.
- Users do not spend as much time waiting, since each user's turns will be a lot closer together rather than one whole game at a time.
- The game will feel less like a single user game and will cause users to be more competitive.

Cons

- This will take much longer to implement.
- This will cause more memory to be used and will make our code less efficient.
- We want the users to feel more competitive with the leaderboard than with the other users.

Second alternative

- In this design, we have users play one at a time, so that each user plays their own game, and then "passes" the dice on to the next user.

Pros

- This would be much simpler to implement.
- We could get rid of the memory at the end of the round, and then reuse the storage for the next play through of the game.
- The users will feel like they are playing against the leaderboard rather than each other.

Cons

- There is a longer wait time between turns since each user must wait until their turn to play the full game.
- The is not the traditional way a multiplayer Yahtzee game is played.

Decision: We chose to go with alternative one, because although it would be more difficult to implement, we want to have an emphasis on realism. Alternative one will look more similar to a real game of Yahtzee than the other alternative. We also do not want the non-playing users to get bored as they wait for their turn to play the game.

Design Alternative Analysis 3

The issue: The problem our group is facing is how to implement the GU Men's Basketball Player Numbers. Specifically, we are focusing on how to randomly generate a player and how to sort the players. We initially were planning to use their jersey numbers. However, the problem is that there is a wide range of jersey numbers (anywhere from 1 to 33) and the jersey numbers are not consecutive (i.e. there are players with jersey numbers 1, 2, 3 and 5, but no number 4). This presents a problem because we cannot simply use the random function in Java. We explored two alternatives for this problem.

Available alternatives

- Alternative 1: Utilize a random function with enum values
- Alternative 2: Assign consecutive integer values between 1 and 15 to each player and utilize the random function we are familiar with

First alternative

- In this design, we utilize a random function in Java for enum values. Each of the user's names is an enum value, so it would be possible with the function to select one.

Pros

- This option is rather straightforward and we found examples of how to implement it on the internet. One example we found was randomly selecting a card suit (i.e. Spades, Hearts, Diamonds, Clubs). Our case would be similar.

Cons

- We have never used the random function with enum values, so it is not something we are very familiar with.

An example found on the internet of implementing a random function with enum values is below. We would replace the card suits with the names of the players on the roster and modify the range to be 0 to 14.

```
public enum CardSuit
{
   Spades,
   Hearts,
   Diamonds,
   Clubs
}

void Start ()
{
   cardSuit = (CardSuit)Random.Range(0, 3);
}
```

Second alternative

- In this design, each player on the roster is assigned an integer value between 1 and 15. We utilize the "normal" and familiar random function to generate a random integer value between 1 and 15. Then, the number generated is matched back to the corresponding player on the roster and further operations are conducted with that player.

Pros

- We completely understand what is happening here.

Cons

- It will require many lines of code in comparison to the first alternative.

```
/**
* "Rolls" the die by finding a random double between 0.0 and 1.0.
* That result is multiplied by the number of sides + 1 and is
* casted to an integer. A series of conditions are checked, such that
* the result of the roll is matched with its corresponding player
* from the roster. In addition, the player is assigned his corresponding
* position and status.
* @param N/A
* @returns N/A
* @throws N/A
*/

public void roll(){
        jerseyNumber = (int)(Math.random() * numberOfSides + 1);

        if(jerseyNumber == 3 || jerseyNumber == 4 || jerseyNumber == 5 ||
          jerseyNumber == 7 || jerseyNumber == 13 || jerseyNumber == 15)
            position = "FORWARD";
        else
            position = "GUARD";

        if(jerseyNumber == 9 || jerseyNumber == 10 || jerseyNumber == 11 ||
                                 jerseyNumber == 13 || jerseyNumber == 15)
            status = "STARTER";
        else
            status = "BENCH";

        if(jerseyNumber == 1)
            name = "Joel Ayayi";
        else if(jerseyNumber == 2)
            name = "Jack Beach";
        else if(jerseyNumber == 3)
            name = "Brandon Clarke";
        else if(jerseyNumber == 4)
            name = "Rui Hachimura";
        else if(jerseyNumber == 5)
            name = "Jeremy Jones";
        else if(jerseyNumber == 6)
            name = "Corey Kispert";
        else if(jerseyNumber == 7)
            name = "Jacob Larson";
        else if(jerseyNumber == 8)
            name = "Alex Martin";
        else if(jerseyNumber == 9)
            name = "Silas Melson";
        else if(jerseyNumber == 10)
            name = "Zach Norvell Jr.";
        else if(jerseyNumber == 11)
            name = "Josh Perkins";
        else if(jerseyNumber == 12)
            name = "Brian Pete";
        else if(jerseyNumber == 13)
            name = "Killian Tillie";
        else if(jerseyNumber == 14)
            name = "Jesse Wade";
        else
            name = "Jonathan Williams";
}
```

Decision: We elected to go with the second alternative because we understood it the best. We know it will work and how it will work, so we were simply just most comfortable with it. The user will not know the difference from what they see and interact with.