Assignment 1 - Yatzee
Milestone 1 – Die Object

Dean Zeller, Dr. Krishna Kambhampaty
CS201 Fall 2022
Deadline: Fri, Sep 16th, 2022

Objective The students will design and test a simple die object for use in future assignments involving *Yahtzee* and other dice-games.

Background
Many board games employ some sort of random determination mechanic. The most common version for gameplay is a set of six-sided dice (singular, die). *Yahtzee* uses five dice, whereas *Monopoly, Parcheesi* and *Backgammon* use two. Other games such as *Dungeons and Dragons* use dice of different sizes, denoted by a D. D4 refers to a 4-sided die, with an equal chance of any number between 1 and 4, whereas a D20 would have a range between 1 and 20. The Die object you create within this assignment will be used for future assignments.

Nomenclature
In Object Oriented Programming, it is very, very important to name classes correctly, to reflect what is contained within. The class created in this assignment is the representation for a single "die" object. Do not call the class "Dice" as that is a plural form of "die", and is incorrect. It is far more appropriate to use "dice" as a variable name for an ArrayList of Die objects, but not the object itself.

Instructions
1. With a new folder named `Yahtzee` (or something similar), create the Java files and `Die.java` and `DieTester.java`
2. In both files, enter the appropriate file documentation to identify you as the primary programmer.
3. In `Die.java`, implement a class for the die object described below, including documentation, attributes, and methods. Use the `DieTester.java` file to test your object in development. You do not need to implement user input; simply hardcode the use of the die in the tester, using parameters for the values.
4. Within DieTester, write a function called `singleRollTest`. It should take a die as a parameter and run a single test of rolling the die. All of the printing and rolling should take place in the function.
5. Within DieTester, write a function called `multipleRollTest`. It should take a die and a number of rolls as parameters. It will then roll the die the specified number of times, outputting the results and the sum of the rolls.
6. Upon completion, your program should execute exactly like the program test run, but with different random numbers. Test your program several times with different integer values, to make sure your program works correctly. In the tester, complete the following:
    a. Call the `printSummary` method to give a general summary of the die.
    b. Call the single-roll test function to generate a single random number using the die object.
    c. Call the multiple-roll test function to generate a series of numbers using the die object. Print the numbers all on the same line, and calculate the sum. Use a different number of rolls for the tests.
7. Create five additional tests with other dice, similar to those given. Remember *encapsulation* – only functionality dealing with the Die object itself should be within the Die class. Any testing should be done in the tester.
8. Take screenshots of your output. It should fit on a single screenshot, but use as many as you need to show all of the tests.
9. Take screenshots of your code. Only one screenshot is necessary for each file. Include as much code as reasonably possible in the screenshot for each file.

10. Create a Zip file that contains the following files:
    a. `Die.java`
    b. `DieTester.java`
    c. Output screenshot(s)
    d. Code screenshots (2)
11. Submit your zip file to the appropriate Canvas dropbox by the due date.

Layout:

Object: `Die` or `DieObject`
(Note: do not call it dice, that is plural and denotes multiple die in a set)

Attributes:

`name` – String, describes name of die (such as D20)
`numSides` – integer, number of sides on the die
`currentValue` – integer, the current value of the die roll

Methods:

Method	*Purpose*
Primary Constructor (`name, numSides`)	Create a die object, using a String parameter for the name and an integer parameter for the number of sides.
Constructor (`name`)	Create a die object, using a String parameter for the name, and with a default parameter of 6 sides.
Constructor (`numSides`)	Create a die object, using an integer parameter for the number of sides. Since the name is not specified, automatically generate the name based on the number of sides (such as D6 or D20).
Constructor (no parameters)	Create a 6-sided die, with the name of D6.
`getNumSides`	Return the number of sides.
`setNumSides(numSides)`	Set the number of sides to the parameter given. Ensure that the number of sides is 1 or more. If 0 or less is entered, print a warning message and do not change the attribute.
`getCurrentValue`	Return the current value of the die as an integer.
`toString`	Return the current value of the die as a String. This is useful for easy printing of the object.
`printSummary`	Print a 3-line summary of the die attributes. (See example)
`roll`	Set the current value to a generated random value between 1 and the number of sides, inclusive. (Essentially, "rolling" the die.)
`setCurrentValue` (or `cheat`)	Set the current value to the specified parameter, with error checking to ensure the new value is greater than 0 and not greater than the number of sides.
`setCurrentValueOverride` (or `reallyCheat`)	Set the current value to the specified parameter. Do not provide any error checking, and allow the die roll to be any integer value, regardless of common sense. (large numbers, negative)

Test Run:

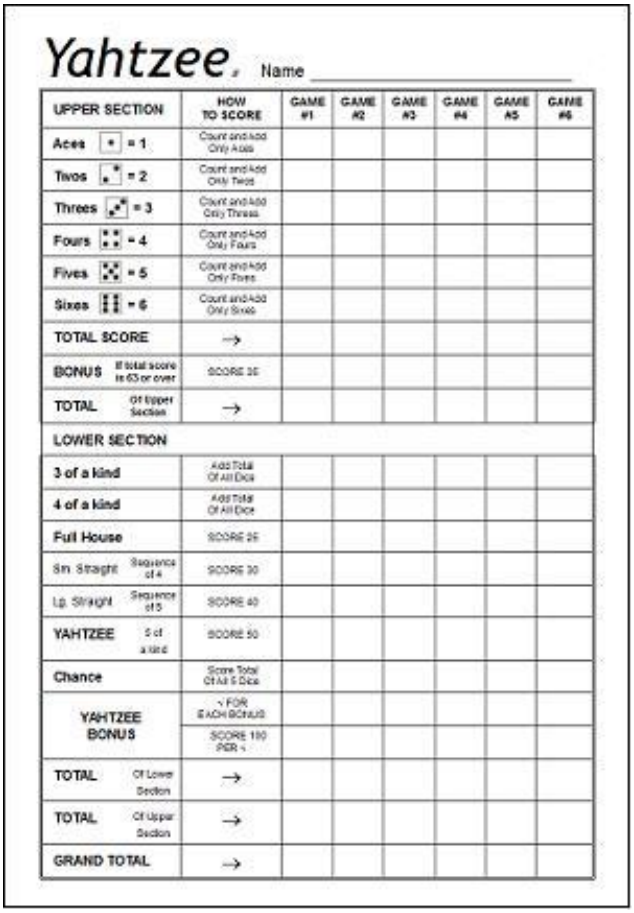
```
Summary:
   Name:   D6
   Range:  1 to 6

Single number test: 5

Multiple number test (10 rolls)
3 1 6 4 5 1 2 3 1 1
Sum:  27



Summary:
   Name:   Percentile die
   Range:  1 to 100

Single number test: 17

Multiple number test (20 rolls)
31 41 59 26 5 35 89 79 32 36 4 82 64 33 83 27 9 50 28 84
Sum:  897



Summary:
   Name:  12-sided die
   Range:  1 to 12

Single number test: 4

Multiple number test (6 rolls)
8 2 7 11 5 7
Sum:  40
```

Milestone 2 – *Yahtzee* Hand

Dean Zeller, Dr. Krishna Kambhampaty
CS201 Fall 2022
Deadline: Sun, Sep 25th, 2022

Objective The student will use Java to program the first part of a solo game of *Yahtzee*, making use of the Die object previously created in Milestone 1.

Reference Yahtzee rules
https://www.wikihow.com/Play-Yahtzee


Yahtzee. Name _____

UPPER SECTION	HOW TO SCORE	GAME #1	GAME #2	GAME #3	GAME #4	GAME #5	GAME #6
Aces • = 1	Count and Add Only Aces						
Twos = 2	Count and Add Only Twos						
Threes = 3	Count and Add Only Threes						
Fours = 4	Count and Add Only Fours						
Fives = 5	Count and Add Only Fives						
Sixes = 6	Count and Add Only Sixes						
TOTAL SCORE	→						
BONUS If total score is 63 or over	SCORE 35						
TOTAL Of Upper Section	→						
LOWER SECTION							
3 of a kind	Add Total Of All Dice						
4 of a kind	Add Total Of All Dice						
Full House	SCORE 25						
Sm Straight Sequence of 4	SCORE 30						
Lg Straight Sequence of 5	SCORE 40						
YAHTZEE 5 of a kind	SCORE 50						
Chance	Score Total Of All 5 Dice						
YAHTZEE BONUS	√ FOR EACH BONUS						
	SCORE 100 PER √						
TOTAL Of Lower Section	→						
TOTAL Of Upper Section	→						
GRAND TOTAL	→						

Background – Divide and Conquer Programming
Milestone 2 builds onto Milestone 1 to eventually produce a solo game of Yahtzee. Big problems tend to be easier if they are broken into smaller, more understandable parts. Creating the `Die` object was just the first part. This milestone involves creating an `ArrayList` of `Die` objects in a new object named `YahtzeeHand`. In Milestone 3, you will use the YahtzeeHand object to write a *Yahtzee* solo game.

Background – Front-End and Back-End Programming
Software engineering courses will go into great detail over what is considered front-end vs. back-end programming. In this case, the Die and YahtzeeHand objects are considered back-end programming, as they deal with the overall functionality. When we create YahtzeeSoloGame in Milestone 3, you will build an interface around the YahtzeeHand object, creating the front-end of the project.

Instructions

1. Learn the dice game, *Yahtzee*. It isn't the most strategy-heavy game in the world. It is a good start to game programming because it is easy to learn and fun for all ages, yet still challenging to program.
2. Within the same folder, create a Java file for a `YahtzeeHand` object.
3. Define `dice` as an ArrayList of `Die` objects as an attribute of `YahtzeeHand`. This time, use the plural form of die, "dice" as a single variable will refer to a series of Die object instances.
4. Create three constructors for `YahtzeeHand`, as follows:

Two integer parameters specified	Number of dice, and number of sides.
One integer parameter specified	Number of dice are specified. Use 6 sides as a default.
No parameters	Use 6 sides and 5 dice as defaults.

5. Create a `YahtzeeHandTester`, to test the methods of `YahtzeeHand`.
6. Create and test the following support methods. You may create other methods as necessary.

`void rollDice()`	Roll all of the dice, to generate new values.
`String toString()`	Return a String of the values of all dice separated by spaces.
`int countDice(int n)`	Return the number of dice matching the parameter `n`. This method will be used by many of the scoring methods.

`void setDice()` Accept an ArrayList of integers as a parameter, setting all dice.

7. Modify the `YahtzeeHand` object to include the methods for scoring listed below. Use the tester to illustrate they work correctly, both in recognizing correct and incorrect hands. Note that all functions return an integer value, representing the score of the dice.

   `int faceValue(int n)`
   Return the number of `n`'s in the Yahtzee Hand, multiplied by `n`. This function will represent the top six rows of the scoresheet. The same test be completed without a parameter, but with six different functions (`score ones`, `score twos`, etc..)

   `int chanceValue()`
   Sum the dice, and return the result. There are no dice requirements on Chance, so a sum of the dice will always be returned. Even though this is the last row in the Yahtzee scoresheet, implement it first. Once complete, the `chanceValue` function can be used to calculate the sum in the `threeKindValue` and `fourKindValue` functions, making the code more modular overall.

   `int threeKindValue()`
   If at least three of the dice have the same value, return the sum of all dice by calling the `chanceValue` function. Otherwise, return 0.

   `int fourKindValue()`
   If at least four of the dice have the same value, return the sum of all dice by calling the `chanceValue` function. Otherwise, return 0.

   `int fullHouseValue()`
   If three of the dice are equal, and the other two dice are also equal, return 25. Otherwise, return 0. All five dice equal also qualifies for a full house, even though it is typically put into the higher-scoring Yahtzee.

   `int smallStraightValue()`
   If four of the five dice are in a consecutive order, return 30. Otherwise, return 0. You may need to implement some kind of `sortDice` method to sort them in increasing order, but there are also solutions in which sorting is not necessary. The only combinations of a small straight are 1 - 2 - 3 – 4 , 2 – 3 – 4 – 5, or 3 – 4 – 5 – 6, so checking for them manually is an easy alternative to sorting the cards.

   `int largeStraightValue()`
   If all five dice are in a consecutive order, return 40. Otherwise, return 0. See the comment for small straight and sorting cards, as it applies to large straight as well, with only two combinations to check (1 – 2 – 3 – 4 – 5 or 2 – 3 – 4 – 5 – 6).

   `int yahtzeeValue()`
   If all five dice are equal, return 50. Otherwise, return 0.

8. Once all of the Yahzee scoring methods are complete, write the method necessary to display a report. Create a `void reportLine (int lineNum)` method to print one row of the report found below. Display the `lineNum` parameter, the current values of the dice, and the result for each of the Yahtzee score categories.

9. Write a corresponding method, `void reportHeader()` that simply prints the header of the report, with the same spacing as the `reportLine` data.
10. In the tester, recreate the first report found below. For the first seven tests, use the dice given on the assignment writeup. Create at least three additional tests that show your program works correctly.
11. Take screenshot of your code files. Include as much of the code you can reasonably fit in a single screenshot.
12. Take a screenshot of the output table.
13. Create a Zip file that contains the following files:
    a. `Die.java`
    b. `Yahtzeehand.java`
    c. `YahtzeehandTester.java`
    d. Screenshots of program (3)
    e. Screenshot of output (1)
14. Create a zip file with your program file and screenshots. Make sure all necessary files are in the submission, and no unnecessary files. Submit the zip file to the appropriate dropbox on Canvas by the due date.

Extra Credit

- Create additional score categories. For example, the German version of Yahtzee has two extra slots for one-pair and two-pair, that work similar to 3-kind and 4-kind. Another possibility is to include the mini-straight, similar to the small-straight but with only three-in-a-row (1-2-3, 2-3-4, 3-4-5, 4-5-6).
- This assignment assumes Yahtzee is a game with 5 dice. Write the program to use any number of dice, up to 20. You will need to modify some of the rules to reflect this change.
- This assignment assumes Yahtzee is a game with six-sided dice. Write the program have dice of any number of sides, up to 20. You will need to modify some of the rules to reflect this change.
- Display the dice graphically in text, or with any graphics package.

Report

```
YahtzeeHand Report

Creating 10 manual YahzteeHand examples

    Dice        1s   2s   3s   4s   5s   6s   3k   4k   FH   Sm   Lg   Yt   Ch
1.  2 1 3 6 1    2    2    3    0    0    6    0    0    0    0    0    0   13
2.  6 2 5 3 4    0    2    3    4    5    6    0    0    0   30   40    0   20
3.  5 5 2 1 5    1    2    0    0   15    0   18    0    0    0    0    0   18
4.  2 2 4 2 4    0    6    0    8    0    0   14    0   25    0    0    0   14
5.  4 4 4 4 4    0    0    0   20    0    0   20   20   25    0    0   50   20
6.  2 3 4 5 4    0    2    3    8    5    0    0    0    0   30    0    0   18
7.  6 2 6 6 6    0    2    0    0    0   24   26   26    0    0    0    0   26
8.
9.
```

10.

Milestone 3 – *Yahtzee* Solo Game

Dean Zeller, Dr. Krishna Kambhampaty
CS201 Fall 2022
Deadline: Fri, Oct 7th, 2022

Objective The student will create a solo game of *Yahtzee*, making use of the Die and YahtzeeHand objects previously created.

Background – Appropriate Code Location

One of the major points of Object Oriented Design is the concept of appropriate placement of functionality. Methods should be written as low on the object hierarchy as possible, but still appropriate for the problem. The example of appropriate code placement in this assignment is in the calculation of the scores. The calculation of the hand results is done in `YahtzeeHand`, because they can be done at that level. However, the Bonus and Total Score attributes must be completed in the `YahtzeeScoreCard` object, as their value depends on the other scores completed, and not what is shown on the dice. In long-term code development, proper code placement is vital in the design stages, and misplaced code can result in difficult maintenance issues the software development lifecycle.

Instructions

The instructions below only represent a possible way to organize the structure of this project. If you believe to have an alternate organizational scheme for this project, feel free to use it.

1. Create a new object, named `YahtzeeScoreCard`, in a file named `YahtzeeScoreCard.java`. Also create a `YahtzeeScoreCardTester` file.
2. Within the `YahtzeeScoreCard` object, create integer attributes to record the following scores. This may be individual variables or an array. The descriptions indicate what the attributes will eventually hold when the game is played. Most of the scores are methods called from the `YahtzeeHand` object, but two of them are based on other scores within the scoresheet.

`onesScore`	Result of `faceValue(1)`
`twosScore`	Result of `faceValue(2)`
`threesScore`	Result of `faceValue(3)`
`foursScore`	Result of `faceValue(4)`
`fivesScore`	Result of `faceValue(5)`
`sixesScore`	Result of `faceValue(6)`
`bonusScore`	If the sum of `onesScore` through `sixesScore` is 63 or more, score 35. Otherwise, score 0.
`threeKindScore`	Result of `threeKindValue()`
`fourKindScore`	Result of `fourKindValue()`
`fullHouseScore`	Result of `fullHouseValue()`
`smallStraightScore`	Result of `smallStraightValue()`
`largeStraightScore`	Result of `largeStraightValue()`
`yahtzeeScore`	Result of `yahtzeeValue()`
`chanceScore`	Result of `chanceValue()`
`totalScore`	Sum of all scores above

3. You will need some way to denote the values are empty, as opposed to having a score of 0. One way is to declare scrores as `int` variables and use -1 to denote empty. Another method is to use `Integer` variables, and use `null` to denote empty. A third strategy is to have an array of Booleans, each representing whether a row has been taken. In any case, you will need some way to accomplish this task.

4. In the `YahtzeeScoreCard` object, create the following methods:

```
void rollDice()  // rolls all dice
void rollDice(boolean d1, boolean d2, boolean d3, boolean d4,
              boolean d5)    // rolls only true parameters
```

Roll the dice corresponding to the true Boolean parameters. This function parameter may also be done with an ArrayList of Booleans.

```
void scoreOnes()
void scoreTwos()
void scoreThrees()
void scoreFours()
void scoreFives()
void scoreSixes()
```

Calculate the score, and store the result in the appropriate location in the scoresheet. Also recalculate the `bonus` and `totalScore` attributes.

```
void scoreThreeKind()
void scoreFourKind()
void scoreFullHouse()
void scoreSmallStraight()
void scoreLargeStraight()
void scoreYatzee()
void scoreChance()
```

Calculate the score, and store the result in the appropriate location in the scoresheet. Also recalculate the `totalScore` attribute.

```
void displayScoresheet()
```

Display all scores with labels in a neatly formatted table.

5. Within the same folder, create a `YahtzeeSoloGame` object and tester, to use the methods of `YahtzeeScoreCard` in a solo game of *Yahtzee*. Create a menu-driven interface to test the calls interactively to simulate a solo game, similar to the test-run, allowing users to select any of the five dice for rerolling, for up to three total rolls. You may alter the formatting of the output a little, but the content and the feel should be the same.
6. Use text-formatting to ensure the Yahtzee scores are aligned on the right. See the example print format statement below. That seems to be the only place that requires formatting, but feel free to use it elsewhere.
7. Double-check the game interface to make sure:
    a. Scores cannot be overwritten by the player.
    b. Shows a score of blank until it is taken, and then shows the score.
    c. If a hand does not meet the requirements of the selected score category, score 0.

8. Use screencastomatic.com to record a screen capture video demonstrating the solutions to your questions. Include the following requirements, in this order:
    a. When the video starts, it should show a simple title page indicating your name, the class, and the assignment. Introduce yourself, greet the teacher and evaluating students, and indicate that this is your video submission for Assignment 1 in CS201. (30 seconds max)
    b. Scroll through your code while describing what the code is doing and how it is completing the assigned task. (120-240 seconds)
    c. Demonstrate that your program works by playing a complete game. Also demonstrate the error-checking features. (120-240 seconds)
    d. Thank your audience and indicate that the video is complete. You may have some sort of fun ending to entertain the viewer. (30 seconds max)
    e. In the end, the total time of the video should be 5 to 7 minutes.
9. Take screenshot of your code files. Include the block documentation at the top and as much of the code you can reasonably fit in a single screenshot.
10. Take screenshots of turns 2, 10, and 13 during gameplay.
11. Create a Zip file that contains the following files:
    a. `Die.java`
    b. `YahtzeeHand.java`
    c. `YahtzeeScoreSheet.java`
    d. `YahtzeeScoreSheetTester.java`
    e. `YahtzeeSoloGame.java`
    f. `YahtzeeSoloGameTester.java`
    g. Screenshots of programs (6)
    h. Screenshot of output, turns 2, 10, and 13 (3)
    i. The demo screencapture video
12. Submit your zip file to the appropriate Canvas dropbox by the due date.

Grading

You will be graded on the following criteria:

Effort	Creating the class and tester for the `YahtzeeHand` and `YahtzeeScoreCard`, and `YahtzeeSoloGame` objects.
Output Readability	Organization and flow of the output

Extra credit will be given for adding features to the game. Examples include:

a. Graphic display of dice using StdDraw or another graphics library.
b. Sound played while rolling dice or scoring a Yahtzee.
c. Multi-player mode, where turns rotate among players. Create `Player` and `Game` classes to implement rotating turns.
d. Algorithms to play a game automatically. Start with a basic random algorithm that will do very poorly, and then come up with progressively better algorithms.
e. Allow for any number of dice, along with modified rules.
f. Allow for the dice to be any number of sides, along with modified rules.

Print Formatting Example

The code example prints the variable onesScore in two spaces, followed by a newline.

```
System.out.format("1.  Ones:        %2d %n", onesScore);
```

```
Turn #1 of 13

Current Scoresheet:

 1. Ones:
 2. Twos:
 3. Threes:
 4. Fours:
 5. Fives:
 6. Sixes:
BONUS:                0
 7. 3-Kind:
 8. 4-Kind:
 9. Full House:
10. Small Straight:
11. Large Straight:
12. Yahtzee:
13. Chance:
TOTAL:                0

Dice Throw #1:  2 1 6 2 3
Dice to throw:  1 6 3

Dice Throw #2:  2 1 4 2 2
Dice to throw:  1 4

Dice Throw #3:  2 2 4 2 2

Use in row:  2

Score of 8 saved in
row 2 (Twos)
```

```
Turn #2 of 13

Current Scoresheet:

 1. Ones:
 2. Twos:              8
 3. Threes:
 4. Fours:
 5. Fives:
 6. Sixes:
BONUS:                0
 7. 3-Kind:
 8. 4-Kind:
 9. Full House:
10. Small Straight:
11. Large Straight:
12. Yahtzee:
13. Chance:
TOTAL:                8

Dice Throw #1:  1 4 2 5 6
Dice to throw:  1

Dice Throw #2:  6 4 2 5 6
Dice to throw:  4 2 5

Dice Throw #3:  6 1 6 6 6

Use in row:  8

Score of 25 saved in
row 8 (4-Kind)
```

```
Turn #3 of 13

Current Scoresheet:

 1. Ones:
 2. Twos:              8
 3. Threes:
 4. Fours:
 5. Fives:
 6. Sixes:
BONUS:                0
 7. 3-Kind:
 8. 4-Kind:           25
 9. Full House:
10. Small Straight:
11. Large Straight:
12. Yahtzee:
13. Chance:
TOTAL:               33

Dice Throw #1:  5 1 3 4 2
Dice to throw:

Dice Throw #2:  5 1 3 4 2
Dice to throw:

Dice Throw #3:  5 1 3 4 2

Use in row:  11

Score of 40 saved in
row 11 (Large Straight)
```

```
Turn #4 of 13

Current Scoresheet:

 1. Ones:
 2. Twos:              8
 3. Threes:
 4. Fours:
 5. Fives:
 6. Sixes:
BONUS:                0
 7. 3-Kind:
 8. 4-Kind:           25
 9. Full House:
10. Small Straight:
11. Large Straight:   40
12. Yahtzee:
13. Chance:
TOTAL:               73

Dice Throw #1:  5 5 2 5 6
Dice to throw:  2 6

Dice Throw #2:  5 5 1 5 2
Dice to throw:  1 2

Dice Throw #3:  5 5 4 5 4

Use in row:  5

Score of 15 saved in
row 5 (Fives)
```

Turn #5 of 13

Current Scoresheet:

1. Ones:	
2. Twos:	8
3. Threes:	
4. Fours:	
5. Fives:	15
6. Sixes:	
BONUS:	0
7. 3-Kind:	
8. 4-Kind:	25
9. Full House:	
10. Small Straight:	
11. Large Straight:	40
12. Yahtzee:	
13. Chance:	
TOTAL:	88

Dice Throw #1: 3 4 3 6 2
Dice to throw: **4 5 2**

Dice Throw #2: 3 3 3 1 4
Dice to throw: **1 4**

Dice Throw #3: 3 3 3 3 3

Use in row: **12**

Score of 50 saved
in row 12 (Yahtzee)

Turn #6 of 13

Current Scoresheet:

1. Ones:	
2. Twos:	8
3. Threes:	
4. Fours:	
5. Fives:	15
6. Sixes:	
BONUS:	0
7. 3-Kind:	
8. 4-Kind:	25
9. Full House:	
10. Small Straight:	
11. Large Straight:	40
12. Yahtzee:	50
13. Chance:	
TOTAL:	138

Dice Throw #1: 1 4 2 5 5
Dice to throw: **1 4 2**

Dice Throw #2: 2 4 1 5 5
Dice to throw: **2 4 1**

Dice Throw #3: 1 2 4 5 5

Use in row: **13**

Score of 17 saved
in row 13 (Chance)

Turn #7 of 13

Current Scoresheet:

1. Ones:	
2. Twos:	8
3. Threes:	
4. Fours:	
5. Fives:	15
6. Sixes:	
BONUS:	0
7. 3-Kind:	
8. 4-Kind:	25
9. Full House:	
10. Small Straight:	
11. Large Straight:	40
12. Yahtzee:	50
13. Chance:	13
TOTAL:	151

Dice Throw #1: 6 2 6 6 3
Dice to throw: **2 3**

Dice Throw #2: 6 6 6 6 1
Dice to throw: **1**

Dice Throw #3: 6 6 6 6 5

Use in row: **6**

Score of 24 saved
in row 6 (Sixes)

Turn #8 of 13

Current Scoresheet:

1. Ones:	
2. Twos:	8
3. Threes:	
4. Fours:	
5. Fives:	15
6. Sixes:	24
BONUS:	0
7. 3-Kind:	
8. 4-Kind:	25
9. Full House:	
10. Small Straight:	
11. Large Straight:	40
12. Yahtzee:	50
13. Chance:	13
TOTAL:	151

Dice Throw #1: 1 5 3 4 4
Dice to throw: **1 4**

Dice Throw #2: 2 5 3 4 4
Dice to throw:

Dice Throw #3: 2 5 3 4 4

Use in row: **10**

Score of 30 saved
in row 10 (Small Straight)

Turn #9 of 13

Current Scoresheet:

1. Ones:	
2. Twos:	8
3. Threes:	
4. Fours:	
5. Fives:	15
6. Sixes:	24
BONUS:	0
7. 3-Kind:	
8. 4-Kind:	25
9. Full House:	
10. Small Straight:	30
11. Large Straight:	40
12. Yahtzee:	50
13. Chance:	13
TOTAL:	181

Dice Throw #1: 1 3 1 3 4
Dice to throw: **4**

Dice Throw #2: 1 3 1 3 5
Dice to throw: **5**

Dice Throw #3: 1 3 1 3 2

Use in row: **1**

Score of 2 saved
in row 1 (Ones)

Turn #10 of 13

Current Scoresheet:

1. Ones:	2
2. Twos:	8
3. Threes:	
4. Fours:	
5. Fives:	15
6. Sixes:	24
BONUS:	0
7. 3-Kind:	
8. 4-Kind:	25
9. Full House:	
10. Small Straight:	30
11. Large Straight:	40
12. Yahtzee:	50
13. Chance:	13
TOTAL:	183

Dice Throw #1: 3 3 1 4 6
Dice to throw: **1 4 6**

Dice Throw #2: 3 3 5 6 3
Dice to throw: **5 6**

Dice Throw #3: 3 3 3 2 3

Use in row: **3**

Score of 12 saved
in row 3 (Threes)

Turn #11 of 13

Current Scoresheet:

1. Ones:	2
2. Twos:	8
3. Threes:	12
4. Fours:	
5. Fives:	15
6. Sixes:	24
BONUS:	0
7. 3-Kind:	
8. 4-Kind:	25
9. Full House:	
10. Small Straight:	30
11. Large Straight:	40
12. Yahtzee:	50
13. Chance:	13
TOTAL:	195

Dice Throw #1: 6 2 4 5 1
Dice to throw: **5 6 2 1**

Dice Throw #2: 3 1 4 6 2
Dice to throw: **3 1 6 2**

Dice Throw #3: 3 3 4 2 3

Use in row: **4**

Score of 4 saved
in row 4 (Fours)

Turn #12 of 13

Current Scoresheet:

1. Ones:	2
2. Twos:	8
3. Threes:	12
4. Fours:	4
5. Fives:	15
6. Sixes:	24
BONUS:	35
7. 3-Kind:	
8. 4-Kind:	25
9. Full House:	0
10. Small Straight:	30
11. Large Straight:	40
12. Yahtzee:	50
13. Chance:	13
TOTAL:	234

Dice Throw #1: 2 3 4 4 1
Dice to throw: **2 3 1**

Dice Throw #2: 5 4 5 4 6
Dice to throw: **6**

Dice Throw #3: 5 4 5 4 1

Use in row: **9**
Not a Full House, score 0

Score of 0 saved
in row 9 (Full House)

```
Turn #13 of 13

Current Scoresheet:

 1. Ones:             2
 2. Twos:             8
 3. Threes:          12
 4. Fours:            4
 5. Fives:           15
 6. Sixes:           24
BONUS:               35
 7. 3-Kind:
 8. 4-Kind:          25
 9. Full House:       0
10. Small Straight:  30
11. Large Straight:  40
12. Yahtzee:         50
13. Chance:          13
TOTAL:              234

Dice Throw #1:  6 5 2 3 5
Dice to throw:  6 2 3

Dice Throw #2:  2 5 3 1 5
Dice to throw:  2 3 1

Dice Throw #3:  5 5 5 2 5

Use in row:  7

Score of 22 saved
in row 7 (3-Kind)
```

```
Final Scoresheet:

 1. Ones:             2
 2. Twos:             8
 3. Threes:          12
 4. Fours:            4
 5. Fives:           15
 6. Sixes:           24
BONUS:               35
 7. 3-Kind:          22
 8. 4-Kind:          25
 9. Full House:       0
10. Small Straight:  30
11. Large Straight:  40
12. Yahtzee:         50
13. Chance:          13
TOTAL:              234

234.  Not bad.
```