|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| LEARNING PROFILE FOR Chapter04Exercise04 | | | | | |
| *Name* | *:* | *Tyler Lucas* | *Submission Date* | *:* | *N/A* |
| *Student ID* | *:* | *3305203* | *Date Created* | *:* | *2017-05-23* |

# Problem Statement

Chapter 4, Exercise 4

This exercise builds on Exercise 4.3. Every time you roll the dice repeatedly, trying to get a given total, the number of rolls it takes can be different. The question naturally arises, what’s the average number of rolls to get a given total? Write a function that performs the experiment of rolling to get a given total 10,000 times. The desired total is a parameter to the subroutine. The average number of rolls is the return value. Each individual experiment should be done by calling the function you wrote for Exercise 4.3. Now, write a main program that will call your function once for each of the possible totals (2, 3, …, 12). It should make a table of the results.

# Description of the Code

Main method iterates through every possible permutation of numbers of dice and total to look for, averaging each result over 10,000 rolls. The table format is relatively complex, so formatting Strings (and introductory text) were implemented in their own methods to keep the main method clutter-free.

# Errors and Warnings

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Errors / Warnings** | **Details** | **How I solved them** |
| 1 | java.lang.ArrayIndexOutOfBoundsException: 5 | I swapped the initialized order of two array dimensions but did not change their referenced order when printing their values. | Matched the initialized array dimension order with referenced order when printing. |

# Sample Input and Output

## Exercise Solution Version (Version 1.0)

Let's roll some dice an unnecessarily

gargantuan number of times in order to

figure out, on average, how many rolls

it takes a given number of dice to get

a certain total.

|----------------------------------------------------|

| Average Number of Rolls Taken To Get Total |

| Total |--------------------------------------------|

| on | Number of Dice |

| Dice | 1 | 2 | 3 | 4 | 5 |

|-------|--------|--------|--------|--------|--------|

| | | | | | |

| 001 | 5.0 | 0.0 | 0.0 | 0.0 | 0.0 |

| | | | | | |

| 002 | 5.0 | 36.0 | 0.0 | 0.0 | 0.0 |

| | | | | | |

| 003 | 6.0 | 18.0 | 213.0 | 0.0 | 0.0 |

| | | | | | |

| 004 | 6.0 | 12.0 | 71.0 | 1291.0 | 0.0 |

| | | | | | |

| 005 | 6.0 | 8.0 | 36.0 | 322.0 | 7753.0 |

| | | | | | |

| 006 | 5.0 | 7.0 | 21.0 | 129.0 | 1546.0 |

| | | | | | |

| 007 | 0.0 | 5.0 | 14.0 | 63.0 | 523.0 |

| | | | | | |

| 008 | 0.0 | 7.0 | 10.0 | 36.0 | 216.0 |

| | | | | | |

| 009 | 0.0 | 9.0 | 8.0 | 22.0 | 108.0 |

| | | | | | |

| 010 | 0.0 | 12.0 | 7.0 | 16.0 | 61.0 |

| | | | | | |

| 011 | 0.0 | 18.0 | 8.0 | 12.0 | 38.0 |

| | | | | | |

| 012 | 0.0 | 35.0 | 8.0 | 10.0 | 25.0 |

| | | | | | |

| 013 | 0.0 | 0.0 | 10.0 | 9.0 | 18.0 |

| | | | | | |

| 014 | 0.0 | 0.0 | 14.0 | 8.0 | 14.0 |

| | | | | | |

| 015 | 0.0 | 0.0 | 21.0 | 9.0 | 12.0 |

| | | | | | |

| 016 | 0.0 | 0.0 | 36.0 | 10.0 | 10.0 |

| | | | | | |

| 017 | 0.0 | 0.0 | 72.0 | 12.0 | 9.0 |

| | | | | | |

| 018 | 0.0 | 0.0 | 215.0 | 16.0 | 10.0 |

| | | | | | |

| 019 | 0.0 | 0.0 | 0.0 | 23.0 | 10.0 |

| | | | | | |

| 020 | 0.0 | 0.0 | 0.0 | 36.0 | 12.0 |

| | | | | | |

| 021 | 0.0 | 0.0 | 0.0 | 65.0 | 14.0 |

| | | | | | |

| 022 | 0.0 | 0.0 | 0.0 | 128.0 | 18.0 |

| | | | | | |

| 023 | 0.0 | 0.0 | 0.0 | 321.0 | 25.0 |

| | | | | | |

| 024 | 0.0 | 0.0 | 0.0 | 1287.0 | 37.0 |

| | | | | | |

| 025 | 0.0 | 0.0 | 0.0 | 0.0 | 60.0 |

| | | | | | |

| 026 | 0.0 | 0.0 | 0.0 | 0.0 | 112.0 |

| | | | | | |

| 027 | 0.0 | 0.0 | 0.0 | 0.0 | 219.0 |

| | | | | | |

| 028 | 0.0 | 0.0 | 0.0 | 0.0 | 516.0 |

| | | | | | |

| 029 | 0.0 | 0.0 | 0.0 | 0.0 | 1553.0 |

| | | | | | |

| 030 | 0.0 | 0.0 | 0.0 | 0.0 | 7867.0 |

| | | | | | |

|-------|--------|--------|--------|--------|--------|

## Method Testing and Verification Version (Version 1.1)

See Discussion.

# Discussion

The algorithm is slow and can be improved upon by many orders of magnitude, but the program (script) completes in a reasonable amount of time, about 30 seconds with 5 dice and averaging over 10,000 iterations, so I’ll not be improving on it[[1]](#footnote-1).

## Integer Cut-Off Errors

I did have one beef with the results that I couldn’t let be, however: the results have integer cut-off errors. They’re supposed to be precise doubles, but you can see that a few of the results are clearly off. (The simplest one to spot is the average number of rolls it takes a single die to get a total of 1,2, and 6, at 5.0 – it’s supposed to be 6, or at least close to 6.) This error comes from improperly typecasting an int to double, with the following line of code:

avgRollsToTotal[total-1][dice-1] =

sumNumberRollsToTotal / dataPoints;

avgRollsToTotal is a double, and sumNumberRollsToTotal and dataPoints are int. I’ve always thought of inline typecasting to be a quick and dirty hack, but, after searching online for other opinions and methods, the consensus appears to support this solution. Here’s the fixed code:

avgRollsToTotal[total-1][dice-1] =

sumNumberRollsToTotal / (double)dataPoints;

This gives in the proper output, the results for a single die all being within 0.1 of 6, and didn’t appear to change the run time.

## Integer Overflow Errors

I experimented with setting dataPoints to numbers larger than 10,000, and noticed int overflow errors when set to 1,000,000 (and, I assume, any larger number). I didn’t expect that, as I made a mistake when estimating what values would be used in my algorithm. (Dice.countDiceRollsUntil(int,int) will return an average maximum of 6^dice, not 6\*dice.) I put in a comment not to set dataPoints any larger than .

1. “Premature optimization is the root of all evil.” -- Donald Knuth (Also see [xkcd: Optimization](https://www.xkcd.com/1691/).) [↑](#footnote-ref-1)