***DnD Ability Score Generation Methods***

*An analysis*

Dungeons and Dragons is a great tabletop roleplaying game, which requires a lot of dice. In DnD, you use dice for a lot of things, making an attack, determining strength, telling how perceptive you are, and even seducing a dragon. This adds a lot of randomness and chance to the game.

When making a roll, you usually have a modifier, something to add or subtract from your dice roll. This can be up to +5 or down to -4 (without any other bonuses). These modifiers are determined by something called ability scores.

There are six main ability scores:

* Strength – how strong your character is
* Dexterity – how dextrous your character is
* Constitution – how healthy your character is
* Intelligence – how intelligent your character is, book smarts
* Wisdom – how wise your character is, street smarts
* Charisma – how charismatic your character is

There are several more abilities in DnD, but those scores are determined by these 6 main scores, and also a few proficiencies you may get from race or class bonuses.

When generating these 6 scores, you have a few different options on how to do so. Since they are the main scores, players generally want them to be as high as possible. The ways of generating ability scores are usually decided by the DM.

In this essay, I will go through a few generation methods, and show the statistics on them. If they are skewed to the high or low side, how effective they are, and other interesting quirks that I found.

1. Standard

The standard method for DnD ability score generation is rolling 4d6, dropping the lowest roll, and then totalling the remaining dice. For example, if you rolled

3, 4, 6, 2

You would drop the 2 and get 13 as your ability score (3 + 4 + 6).

I initially wanted to write each possible roll for 4d6, remove the lowest, and get the total that way, but I quickly realised how long and tedious that would get. So, I instead did 4d3 as an example, and wrote a program to generate the rolls for 4d6.

This was the monstrous table of rolls for 4d3:

Chart, table

Description automatically generated

The red is the lowest roll, to be removed, and the green are the totals.

As you can see, even with 4d3, there is a lot of possible combinations, a total of 81 possible rolls.

This then allowed me to make a table and graph for the rolls:

Chart

Description automatically generated

From this, you can see that the frequency of getting very low numbers is low, but it increases for the higher numbers. The maximum is 9, and the minimum is 3, and you can see the disparity between the frequency in each roll.

The most common roll is a seven, which is also, incidentally, the most common roll for 2d6.

From here, I decided to write a program to generate the rolls for 4d6, find and remove the lowest, and get the totals. Then it counts how many of each total occur (for example, how many 3s, 4s, etc.)

Here are the results:

Chart, bar chart, histogram

Description automatically generated

As you can see, the number of rolls very rapidly increases since 4d6 has 1296 possible combinations.

This time, 13 is the most common roll, with 12 with a close second. 13 had a frequency of 172, and 12 has a frequency of 167. We can also again see that it is more difficult to roll 3s and 4s, but the frequencies increase as we get towards 13, then decrease again, but still stay quite high.

The median of these rolls is 13, and the mean is 12.2. This means the data is negatively skewed since mean < median.

For this method, the chance of generating the lowest number possible (3) is 0.077%

The chance of generating the highest number possible (18) is 1.62%

2. Classic

Another very common method for generating ability scores is the classic method. In the classic method, you roll 3d6, and take that as your ability score.

I again wrote a program to generate all the possible rolls and find the totals, instead of writing them by hand.

Here is the data:

Chart, histogram

Description automatically generated

This is a fully symmetric graph, with both 10 and 11 being the peaks. The graph increases and decreases uniformly, producing a perfect bell curve. It makes sense why it is a perfect curve, since it is just pure rolls, nothing to vary or change the results.

For this method, the chance of generating the lowest possible number (3) is 0.46%

This is also the percentage chance for generating the highest possible number (18)

3. Heroic

This generation is a little more uncommon since it doesn’t work well for balanced and generic campaigns. This is more for one-shots or similar if you want more powerful characters.

For this method, you roll 2d6 and then add 6 to get your final score. As you may be able to tell, this boosts the scores quite a bit. The maximum is 18 and the minimum is 8 with this method.

Here is the data:

Chart, bar chart

Description automatically generated

This is also perfectly symmetrical, with 13 being the peak. Again, it is clear why the graph is symmetric since there is nothing to vary the totals. There are 36 data possible rolls for this generation type.

The chance of generating the lowest possible number (8) using this method is 2.7%

This is also the percentage chance for generating the highest possible number (18)

4. Other methods

There are two other methods which are sometimes used, but I haven’t calculated data for.

The first is called dice pool.

Each player has a pool of 24d6 to assign to their scores and decides how many d6 to use for each score, with a minimum of 3d6. After deciding how many dice to roll for each score, the player rolls them and takes the total of the 3 highest dice as their ability score.

For example, if 5d6 were assigned to Strength, and

2, 4, 1, 5, 6

Was rolled, you would take 15 as your ability score (4 + 5 + 6)

The second method is called purchase.

Each player gets a number of points to spend on increasing their ability scores. Each score starts at 8. It costs 1 point to increase the score by one and the player gains 1 point if they decrease their score by one.

The reason I have not calculated the data and made graphs for these two methods is because there are too many factors, too many variables that can change things. I feel like there would be too many things to handle, although I may do it at a later time.

Conclusion

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Min chance | Max chance | Most common | Most common chance |
| Standard | 0.077% | 1.62% | 13 | 13.27% |
| Classic | 0.46% | 0.46% | 10/11 | 12.5% |
| Heroic | 2.7% | 2.7% | 13 | 16.67% |

From this, we can see that the best method for generating the highest scores is, obviously, heroic. It has a chance of 2.7% of generating an 18, which is the highest out of the three methods. It also has a 2.7% chance of generating the lowest and, while that may seem high, remember that the minimum for the heroic method is 8, as opposed to 3 for the other two methods.

Heroic also has the most chance of generating the most common value of 13.

Classic is the worst method of generating ability scores, out of the three since it has the lowest chance of generating an 18 at 0.46%. That’s less than half a percent, albeit there is the same chance for generating the lowest possible score of 3, so maybe it has its advantages.

Classic also has the lowest chance of generating the most common value of 10/11 at 12.5%. It has a 9.72% chance of generating a 13, which is the most common for the other two methods, which is even worse.

Standard method does seem to be the most balanced, which explains why it is the most widely used and called the “standard” method.

Evaluation

Now that I am looking back, the generation method for getting all possible rolls could be skewed or incorrect. I’ll explain how the code works to generate it.

Using the 4d6 example, I start with 4 variables which all have a value of 1.

I add that list to a big list, then check if the last variable is less than 6. If it is, increment it by one.

If not however, I then check if the second last variable is less than 6. If it is, increment it by one and also set the last variable to 1.

If it is not, check if the second variable is less than 6. If it is, increment it by one and also set the second last *and* last variable to 1.

If it is not, check if the first variable is less than 6. If it is, increment it by one and also set all other variables to 1.

If it is not, it means that all variables should be at 6, which means all combos have been accounted for. This then makes the code set a variable to True.

Then loop back to the start, where it appends the four variables as a list to the big list. It then repeats this process until the variable is set to True.

At first, I made the mistake of only setting the next variable to 1 instead of all of the ones after. For example, if the code incremented the second variable, it would only set the third to 1, and leave the fourth alone. This caused me to not have the correct amount of data and so I had to redo parts of it.

The reason why this could be an incorrect method of generating rolls is because it treats the roll

1, 2, 3, 4

The same as

1, 3, 2, 4

And also the same as

3, 1, 4, 2

And so on

I don’t know if that is correct or not, since its technically the same roll but also each die is independent of each roll and every other die.

Thank you very much for reading. Please let me know of any changes and criticisms. I am also not sure about this method of sharing on discord, so let me know how it is.