Monaco

A program to answer probability questions, especially from dice and card games.

A Motivational Document

Introduction

When playing games that include chance, questions often arise of the forms:

- How likely is that event?
- What is the average value of that result?
- What is the distribution of that result?

I[1] have a program called **Monaco** that can answer such questions if they can be suitably posed.

This short document describes some problems that the program can solve, based on real games and one puzzle. The objective of this document is purely motivational, to show you that the program can solve real problems. The program can also – and easily – solve simple problems too, such as "if I roll five dice, what is the probability that I have exactly two sixes?" and similar problems with cards. Some examples of this sort of problem are given in the section "More Simple Problems".

In addition to those simple problems, the problems for *Dungeons and Dragons*, *Yspahan* and bridge are real but simple, and probably represent the complexity level of most uses of the program. This program is for you even if you never want to go beyond that difficulty. The problems for *Monopoly*, poker, cribbage and *Dominion* are particularly complicated, and go well beyond what most users of the program are likely to ever consider. The other problems fall between those two extremes.

The main part of this document describes these problems and reports output from the program that solves them. However, it provides no description as to how the problems are solved by the program, or how that output is produced. A section at the end of this document provides some of that material, but without explanation, for which see the main documentation for the program.

However, one detail of the program does need to be summarised here to understand the examples. The program can in many cases produce exact answers to problems, and much of the output here is of that type. However, exact answers are not always possible, and so in some cases approximate output is produced based on simulating the problem many times, in most cases millions of times.

Which of the modes of the program, exact or approximate, is not explicitly said in each case, but a sign of approximate output is two comma separated values in brackets `[]` representing a range in which the corresponding true result will probably be found.[2] Results should be quoted to a precision consistent with this range. With a large number of simulated results, the accuracy should be good enough for all practical purposes, except that very rare results may not be produced. Exact results, which do not include such ranges, include a decimal representation and an exact value as a ratio.

Dungeons and Dragons

An original standard *Dungeons and Dragons*[3] (*D&D*) characteristic was produced by rolling three standard dice and adding them up. The distribution of such a characteristic is given by the output:

```
3 -  1 ~ 0.00462963 = 1/216
4 -  3 ~ 0.0138889  = 1/72
```

[1] Christopher Dearlove, christopher.dearlove@gmail.com. The program Monaco, current version 1.46, and all associated documentation, including this motivational document, are copyright Christopher Dearlove, 2008-2021, all rights reserved.

[2] Technically, and the output indicates this in some cases, these are estimated 95% confidence intervals. That does not mean that there is a 95% chance that the true result is in that interval, but the true result will usually be in that range, and if outside it will usually be not far outside it.

[3] Gary Gygax and Dave Arneson, TSR, 1974.

```
 5 -  6 ~ 0.0277778  = 1/36
 6 - 10 ~ 0.0462963  = 5/108
 7 - 15 ~ 0.0694444  = 5/72
 8 - 21 ~ 0.0972222  = 7/72
 9 - 25 ~ 0.115741   = 25/216
10 - 27 ~ 0.125      = 1/8
11 - 27 ~ 0.125      = 1/8
12 - 25 ~ 0.115741   = 25/216
13 - 21 ~ 0.0972222  = 7/72
14 - 15 ~ 0.0694444  = 5/72
15 - 10 ~ 0.0462963  = 5/108
16 -  6 ~ 0.0277778  = 1/36
17 -  3 ~ 0.0138889  = 1/72
18 -  1 ~ 0.00462963 = 1/216
```

This output is straightforward to understand, for example the last line means that a result of 18 occurred 1 time, with a probability of about 0.0046, or exactly 1 in 216 of the time. If you prefer you can think of this as about 0.46%; the program could show you it in that form, but I have not done that here – there is an example of this form of output for a later problem. This is a set of exact results, as shown by the presence of the final ratio and the absence of confidence intervals. The number of results was 216, you can get the program to tell you this, as some later examples show, but you do not need it, and so I have not shown that here.

Each line of that output allows you to answer a question of the form “How likely is that?”. For example, as described, from the last line you can answer the question “What is the probability of an 18?”. The set of results as a whole allows you to answer the question “What is the distribution of that result?”.

You may, however, want to answer more complicated “How likely is that?” questions such as “What is the probability of a 16 or greater?”. You could answer that question by adding up the results from the previous output for 16, 17 and 18, but the program can do that for you by producing the results:

```
>=  3 - 216 ~ 1
>=  4 - 215 ~ 0.99537    = 215/216
>=  5 - 212 ~ 0.981481   = 53/54
>=  6 - 206 ~ 0.953704   = 103/108
>=  7 - 196 ~ 0.907407   = 49/54
>=  8 - 181 ~ 0.837963   = 181/216
>=  9 - 160 ~ 0.740741   = 20/27
>= 10 - 135 ~ 0.625      = 5/8
>= 11 - 108 ~ 0.5        = 1/2
>= 12 -  81 ~ 0.375      = 3/8
>= 13 -  56 ~ 0.259259   = 7/27
>= 14 -  35 ~ 0.162037   = 35/216
>= 15 -  20 ~ 0.0925926  = 5/54
>= 16 -  10 ~ 0.0462963  = 5/108
>= 17 -   4 ~ 0.0185185  = 1/54
>= 18 -   1 ~ 0.00462963 = 1/216
```

The line starting `>= 16` gives you the answer, exactly 5 in 108, or about 0.046 or 4.6%.

You may also want to answer questions such as “What is the probability of an 8 or less?”, which can be answered, using the line starting `<=  8`, as exactly 7 in 27, or about 26%, from the output:

```
<=  3 -   1 ~ 0.00462963 = 1/216
<=  4 -   4 ~ 0.0185185  = 1/54
<=  5 -  10 ~ 0.0462963  = 5/108
<=  6 -  20 ~ 0.0925926  = 5/54
<=  7 -  35 ~ 0.162037   = 35/216
```

```
<=  8 -  56 ~ 0.259259   = 7/27
<=  9 -  81 ~ 0.375      = 3/8
<= 10 - 108 ~ 0.5        = 1/2
<= 11 - 135 ~ 0.625      = 5/8
<= 12 - 160 ~ 0.740741   = 20/27
<= 13 - 181 ~ 0.837963   = 181/216
<= 14 - 196 ~ 0.907407   = 49/54
<= 15 - 206 ~ 0.953704   = 103/108
<= 16 - 212 ~ 0.981481   = 53/54
<= 17 - 215 ~ 0.99537    = 215/216
<= 18 - 216 ~ 1
```

However playing *D&D*, most players wanted better characters than that method gives, and one approach that was devised, and is now standard in more recent editions of the game, was to create each *D&D* characteristic as the sum of the best three of four dice rolled. Results for this distribution – in a slightly different order than above – and preceded by overall statistics. are given by the output:

```
Number of results          = 1296
Mean                       = 12.2446 = 15869/1296
Standard deviation         = 2.84684
Minimum result             = 3
Maximum result             = 18

 3 -   1 ~ 0.000771605 = 1/1296
 4 -   4 ~ 0.00308642  = 1/324
 5 -  10 ~ 0.00771605  = 5/648
 6 -  21 ~ 0.0162037   = 7/432
 7 -  38 ~ 0.029321    = 19/648
 8 -  62 ~ 0.0478395   = 31/648
 9 -  91 ~ 0.070216    = 91/1296
10 - 122 ~ 0.0941358   = 61/648
11 - 148 ~ 0.114198    = 37/324
12 - 167 ~ 0.128858    = 167/1296
13 - 172 ~ 0.132716    = 43/324
14 - 160 ~ 0.123457    = 10/81
15 - 131 ~ 0.10108     = 131/1296
16 -  94 ~ 0.0725309   = 47/648
17 -  54 ~ 0.0416667   = 1/24
18 -  21 ~ 0.0162037   = 7/432

<=  3 -    1 ~ 0.000771605 = 1/1296
<=  4 -    5 ~ 0.00385802  = 5/1296
<=  5 -   15 ~ 0.0115741   = 5/432
<=  6 -   36 ~ 0.0277778   = 1/36
<=  7 -   74 ~ 0.0570988   = 37/648
<=  8 -  136 ~ 0.104938    = 17/162
<=  9 -  227 ~ 0.175154    = 227/1296
<= 10 -  349 ~ 0.26929     = 349/1296
<= 11 -  497 ~ 0.383488    = 497/1296
<= 12 -  664 ~ 0.512346    = 83/162
<= 13 -  836 ~ 0.645062    = 209/324
<= 14 -  996 ~ 0.768519    = 83/108
<= 15 - 1127 ~ 0.869599    = 1127/1296
<= 16 - 1221 ~ 0.94213     = 407/432
<= 17 - 1275 ~ 0.983796    = 425/432
<= 18 - 1296 ~ 1
```

```
>=  3 - 1296 ~ 1
>=  4 - 1295 ~ 0.999228  = 1295/1296
>=  5 - 1291 ~ 0.996142  = 1291/1296
>=  6 - 1281 ~ 0.988426  = 427/432
>=  7 - 1260 ~ 0.972222  = 35/36
>=  8 - 1222 ~ 0.942901  = 611/648
>=  9 - 1160 ~ 0.895062  = 145/162
>= 10 - 1069 ~ 0.824846  = 1069/1296
>= 11 -  947 ~ 0.73071   = 947/1296
>= 12 -  799 ~ 0.616512  = 799/1296
>= 13 -  632 ~ 0.487654  = 79/162
>= 14 -  460 ~ 0.354938  = 115/324
>= 15 -  300 ~ 0.231481  = 25/108
>= 16 -  169 ~ 0.130401  = 169/1296
>= 17 -   75 ~ 0.0578704 = 25/432
>= 18 -   21 ~ 0.0162037 = 7/432
```

The new answers to the previous questions are now: How likely is a characteristic to be 18? 7 in 432 or about 1.6%. How likely is a characteristic to be 16 or more? 169 in 1296 or about 13%. How likely is a characteristic to be 8 or less? 17 in 162, or about 10.5%. The mean characteristic is about 12.24, up from an average of 10.5 (not shown, but obvious) for just using three dice.

Yspahan

In the game *Yspahan*[4] in a normal turn you roll nine dice. You then group them by number rolled. One question of interest is how many different numbers you are likely to roll, because that affects which options are open to you that turn. The distribution of that result is given by the output:

```
1 -       6 ~ 5.95374e-07 = 1/1679616
2 -    7650 ~ 0.000759102 = 425/559872
3 -  363000 ~ 0.0360201   = 15125/419904
4 - 2797200 ~ 0.277563    = 6475/23328
5 - 5004720 ~ 0.496614    = 11585/23328
6 - 1905120 ~ 0.189043    = 245/1296
```

There is an option when playing the game of buying extra dice. When buying the maximum three extra dice, the distribution of the number of different numbers is now:

```
1 -         6 ~ 2.75636e-09 = 1/362797056
2 -     61410 ~ 2.82114e-05 = 10235/362797056
3 -  10383120 ~ 0.00476994  = 24035/5038848
4 - 220140360 ~ 0.101131    = 3057505/30233088
5 - 993168000 ~ 0.456255    = 287375/629856
6 - 953029440 ~ 0.437816    = 1654565/3779136
```

The chances of seeing all six numbers have gone up from about 18.9% to about 43.8%.

More Simple Problems

The previous two problems are the sorts of problem you are probably most likely to want to solve. The remainder of this document is mainly about more complicated problems, to show what the program is capable of. This section provides some balance, to show examples of some other simple problems that are easily handled by the program. Most are not from specific games, but problems like them may be found in many games. Two are dice problems, two are card problems.

[4] Sébastien Pauchon, Ystari Games, 2006.

These problems are:

- I draw four cards from a standard deck. What is the probability that I have one of each suit?
- I roll six standard dice. What is the probability I can put them all in, possibly repeated, pairs?
- I roll three dice numbered 0, 2, 3, 4, 5, 6. What is the distribution of the largest minus the smallest?[5]
- I draw three cards from a standard deck, and total their values, where ace to ten score 1 to 10 points and court cards score 10 points. What is the distribution of this score?

Here is some output that answers these questions, one block of output per question – the last one has a page break within its block of output, but that is just when included here. The first two outputs each answer a question of the form, "What is the probability of that?", other than from a distribution.

```
Number of results          = 270725
Number of false results    = 242164
Number of true results     = 28561
Probability                = 0.105498 = 2197/20825

Number of results          = 46656
Number of false results    = 44400
Number of true results     = 2256
Probability                = 0.0483539 = 47/972

0 -  6 ~ 0.0277778 = 1/36
1 - 24 ~ 0.111111  = 1/9
2 - 42 ~ 0.194444  = 7/36
3 - 48 ~ 0.222222  = 2/9
4 - 42 ~ 0.194444  = 7/36
5 - 24 ~ 0.111111  = 1/9
6 - 30 ~ 0.138889  = 5/36

 3 -    4 ~ 0.000180995 = 1/5525
 4 -   24 ~ 0.00108597  = 6/5525
 5 -   48 ~ 0.00217195  = 12/5525
 6 -   92 ~ 0.0041629   = 23/5525
 7 -  136 ~ 0.00615385  = 2/325
 8 -  200 ~ 0.00904977  = 2/221
 9 -  268 ~ 0.0121267   = 67/5525
10 -  352 ~ 0.0159276   = 88/5525
11 -  440 ~ 0.0199095   = 22/1105
12 -  620 ~ 0.0280543   = 31/1105
13 -  824 ~ 0.0372851   = 206/5525
14 -  960 ~ 0.0434389   = 48/1105
15 - 1124 ~ 0.0508597   = 281/5525
16 - 1216 ~ 0.0550226   = 304/5525
17 - 1336 ~ 0.0604525   = 334/5525
18 - 1388 ~ 0.0628054   = 347/5525
19 - 1464 ~ 0.0662443   = 366/5525
20 - 1472 ~ 0.0666063   = 368/5525
21 - 1772 ~ 0.080181    = 443/5525
22 - 1544 ~ 0.0698643   = 386/5525
23 - 1384 ~ 0.0626244   = 346/5525
24 - 1180 ~ 0.0533937   = 59/1105
```

[5] This is taken from the game *Onward to Venus* (Martin Wallace, Treefrog Games, 2014).

```
25 - 1040 ~ 0.0470588   = 4/85
26 -  856 ~ 0.038733    = 214/5525
27 -  740 ~ 0.0334842   = 37/1105
28 -  576 ~ 0.0260633   = 144/5525
29 -  480 ~ 0.0217195   = 24/1105
30 -  560 ~ 0.0253394   = 28/1105
```

From these you can see that the probability of one card of each suit is about 10.55%, that the probability that the dice can all be put in pairs is about 4.84%, the most likely dice difference is 3, which occurs 2/9 of the time, and that the most likely card total is 21, which occurs about 8.02% of the time. Other questions can be answered from the output in the last two cases. As in the *Dungeons and Dragons* example, in the third case if you want results such as how often is the dice difference 3 or less, or 3 or more, then the program run can be set up to also produce the output:

```
<= 0 -   6 ~ 0.0277778 = 1/36
<= 1 -  30 ~ 0.138889  = 5/36
<= 2 -  72 ~ 0.333333  = 1/3
<= 3 - 120 ~ 0.555556  = 5/9
<= 4 - 162 ~ 0.75      = 3/4
<= 5 - 186 ~ 0.861111  = 31/36
<= 6 - 216 ~ 1

>= 0 - 216 ~ 1
>= 1 - 210 ~ 0.972222 = 35/36
>= 2 - 186 ~ 0.861111 = 31/36
>= 3 - 144 ~ 0.666667 = 2/3
>= 4 -  96 ~ 0.444444 = 4/9
>= 5 -  54 ~ 0.25     = 1/4
>= 6 -  30 ~ 0.138889 = 5/36
```

You can also get output including other forms of average values:

```
Number of results          = 216
Mean                       = 3.33333 = 10/3
Standard deviation         = 1.63299
Minimum result             = 0
Maximum result             = 6
Median                     = 3
Mode                       = 3
```

Similar output in the last example is possible but is not provided here.

Can't Stop

In the game *Can't Stop*[6] for most of the game in your turn you set three target numbers – by a means that we do not consider here – and then each of your turns may have 0, 1 or 2 successes. We count successes by considering the best of the three ways in which the four dice can be split into two pairs and how many of those pairs match a target number. We ignore here that successes are not always entirely equal in value and that more complicated circumstances can occur later in the game.

The easiest target numbers are 6, 7 and 8. The average number of successes is given by the output:

```
Number of results          = 1296
Mean                       = 1.3179 = 427/324
Standard deviation         = 0.614275
```

[6] Sid Sackson, Parker Brothers, 1980.

```
Minimum result              = 0
Maximum result              = 2
```

More usefully, the distribution of the number of successes is given by the following output. In this case an additional piece of output has been added to the distribution, the odds. What is often of greatest interest playing the game is the probability of failing, the zero result, which is about 8.0%.

```
0 - 104 ~ 0.0802469 = 13/162   (1 in 12.4615)
1 - 676 ~ 0.521605  = 169/324  (1 in 1.91716)
2 - 516 ~ 0.398148  = 43/108   (1 in 2.51163)
```

For a harder but reasonable set of target numbers 4, 7 and 10, the output is:

```
0 - 160 ~ 0.123457 = 10/81  (1 in 8.1)
1 - 832 ~ 0.641975 = 52/81  (1 in 1.55769)
2 - 304 ~ 0.234568 = 19/81  (1 in 4.26316)
```

For the hardest set of target numbers, 2, 3 and 12 (and its mirror image 2, 11 and 12), the output is:

```
0 - 728 ~ 0.561728  = 91/162   (1 in 1.78022)
1 - 538 ~ 0.415123  = 269/648  (1 in 2.40892)
2 -  30 ~ 0.0231481 = 5/216    (1 in 43.2)
```

The probability of failing has gone up from about 8.0% to about 12.3% and then to about 56.2%.

Titan

From the game *Titan*,[7] we consider a single character versus single character combat continuing until at least one character is killed. Here we give results for a serpent versus a colossus. The serpent rolls 18 dice and needs to be hit 18 times to be killed, the colossus rolls 10 dice and needs to be hit 10 times to be killed. The serpent (skill 2) needs to roll 6 to hit on each die, the colossus (skill 4) needs to roll 2 or more to hit on each die. A serpent win is reported as a 1, a colossus win is reported as a -1, a draw (both are killed) is reported as a 0. Output is:

```
-1 - 70168191 ~ 0.701682   [0.701592, 0.701772]
 0 - 25429020 ~ 0.25429    [0.254205, 0.254376]
 1 -  4402789 ~ 0.0440279  [0.0439877, 0.0440681]
```

The serpent has only a 4.4% chance of winning, plus a 25.4% chance of drawing.

But the serpent might be sitting in bramble. Now the colossus needs 3 or more to hit. Output is:

```
-1 - 53236411 ~ 0.532364  [0.532266, 0.532462]
 0 - 34545708 ~ 0.345457  [0.345364, 0.34555]
 1 - 12217881 ~ 0.122179  [0.122115, 0.122243]
```

The serpent is still losing, but its chance of winning is now 12.2%, with a 24.5% chance of drawing.

An unwise colossus could also be sitting in bramble and now needs 4 or more to hit. Output is:

```
-1 - 24911620 ~ 0.249116  [0.249031, 0.249201]
 0 - 31933725 ~ 0.319337  [0.319246, 0.319429]
 1 - 43154655 ~ 0.431547  [0.431449, 0.431644]
```

Now it is advantage serpent, with a 43.2% chance of winning, plus a 31.9% chance of drawing.

[7] Jason McAllister and David Trampier, Avalon Hill, 1980.

Monopoly

A problem related to the game *Monopoly*[8] is to ask what are the probabilities of each space being visited after each die roll – not after each turn – and any actions resulting from that die roll, such as Go to Jail! or following Chance and Community Chest cards.

The following output is of the probabilities of ending on each space playing the standard American version of the game. Spaces are numbered from 0 (Go) to 39 (Boardwalk). We assume that a player always pays immediately to leave Jail, and thus there is no significant difference whether a player is in Jail or is Just Visiting. However, we will show Jail as space -1, with only Just Visiting as space 10:

```
-1 - 3949427 ~ 0.0394943  [0.0394561, 0.0395325]
 0 - 3095945 ~ 0.0309594  [0.0309255, 0.0309934]
 1 - 2130424 ~ 0.0213042  [0.021276, 0.0213326]
 2 - 1885883 ~ 0.0188588  [0.0188322, 0.0188855]
 3 - 2162311 ~ 0.0216231  [0.0215946, 0.0216516]
 4 - 2330091 ~ 0.0233009  [0.0232714, 0.0233305]
 5 - 2961620 ~ 0.0296162  [0.029583, 0.0296494]
 6 - 2262978 ~ 0.0226298  [0.0226006, 0.0226589]
 7 -  865810 ~ 0.0086581  [0.00863996, 0.00867628]
 8 - 2318373 ~ 0.0231837  [0.0231543, 0.0232132]
 9 - 2302024 ~ 0.0230202  [0.0229909, 0.0230497]
10 - 2270618 ~ 0.0227062  [0.022677, 0.0227354]
11 - 2701725 ~ 0.0270172  [0.0269855, 0.027049]
12 - 2601493 ~ 0.0260149  [0.0259837, 0.0260461]
13 - 2372096 ~ 0.023721   [0.0236912, 0.0237508]
14 - 2465523 ~ 0.0246552  [0.0246249, 0.0246856]
15 - 2918841 ~ 0.0291884  [0.0291554, 0.0292214]
16 - 2792326 ~ 0.0279233  [0.027891, 0.0279556]
17 - 2592582 ~ 0.0259258  [0.0258947, 0.025957]
18 - 2941119 ~ 0.0294112  [0.0293781, 0.0294443]
19 - 3084903 ~ 0.030849   [0.0308152, 0.0308829]
20 - 2881167 ~ 0.0288117  [0.0287789, 0.0288445]
21 - 2836698 ~ 0.028367   [0.0283345, 0.0283995]
22 - 1048063 ~ 0.0104806  [0.0104607, 0.0105006]
23 - 2736556 ~ 0.0273656  [0.0273336, 0.0273976]
24 - 3180937 ~ 0.0318094  [0.031775, 0.0318438]
25 - 3066087 ~ 0.0306609  [0.0306271, 0.0306947]
26 - 2708581 ~ 0.0270858  [0.027054, 0.0271176]
27 - 2678223 ~ 0.0267822  [0.0267506, 0.0268139]
28 - 2807509 ~ 0.0280751  [0.0280427, 0.0281075]
29 - 2585856 ~ 0.0258586  [0.0258275, 0.0258897]
30 -       0 ~ 0
31 - 2676545 ~ 0.0267654  [0.0267338, 0.0267971]
32 - 2626627 ~ 0.0262663  [0.0262349, 0.0262976]
33 - 2364517 ~ 0.0236452  [0.0236154, 0.023675]
34 - 2502655 ~ 0.0250266  [0.024996, 0.0250572]
35 - 2433569 ~ 0.0243357  [0.0243055, 0.0243659]
36 -  867700 ~ 0.008677   [0.00865884, 0.0086952]
37 - 2187114 ~ 0.0218711  [0.0218425, 0.0218998]
38 - 2179494 ~ 0.0217949  [0.0217663, 0.0218236]
39 - 2625990 ~ 0.0262599  [0.0262286, 0.0262913]
```

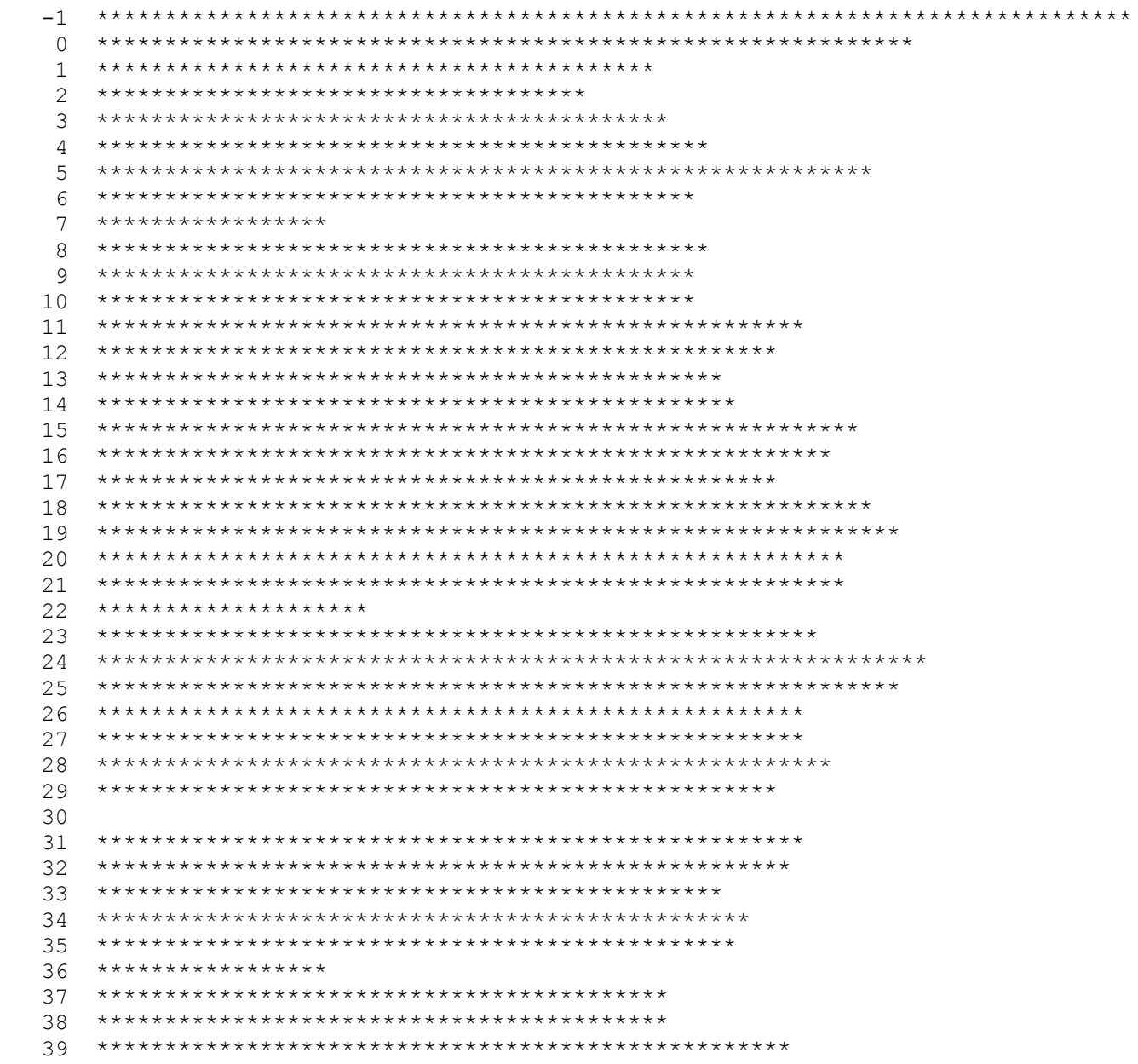
These are approximate results, over a chosen number of a hundred million die rolls – so they are quite accurate versions of approximate. You can, as previously mentioned, tell they are approximate

[8] Parker Brothers, 1935, based on Elizabeth Magie, *The Landlord's Game*, 1902.

from the final confidence interval replacing the exact ratio. The output has been forced to show the result 30 (Go to Jail!) where a player cannot finish a turn. However, because a confidence interval would be meaningless in this case one is not shown.

The most obvious other results are that the most common space is -1 (Jail, excluding Just Visiting). The least common spaces, other than Go to Jail!, are the Chance spaces, numbers 7, 22 and 36 – because more than half of all Chance cards move you somewhere else – and the most common spaces other than Jail are 24 (Illinois Avenue), 0 (Go) and 25 (B & O Railroad).

You might also find the following useful in seeing the variation.



Dice Rerolling Games

There are many games in which you have some dice, roll them, and then you can reroll some or all of those dice again a, not necessarily fixed, number of times. This section considers some examples. Again, answers are approximate, each over ten million sets of rolls.

The simplest case, requiring no intelligence, is one where you have a fixed target number and reroll those dice that are not showing the target number. A typical number of dice is five, with one roll and

two rerolls, or three rolls in total. This might be a turn late in the game *Yahtzee*[9] if – not very cleverly – you have left a single target number to the end.

The distribution of the number of successes in this case is given by the output:

```
0 -  648967 ~ 0.0648967  [0.0647442, 0.0650495]
1 - 2362523 ~ 0.236252   [0.235989, 0.236516]
2 - 3439379 ~ 0.343938   [0.343644, 0.344232]
3 - 2503204 ~ 0.25032    [0.250052, 0.250589]
4 -  913316 ~ 0.0913316  [0.0911532, 0.0915103]
5 -  132611 ~ 0.0132611  [0.0131904, 0.0133322]
```

Next, we consider a case where instead of a fixed target number, we can choose the target number as the most common number in our first roll. But having chosen that number we are stuck with it. The distribution of the number of successes in this case is given by the output:

```
1 -  215014 ~ 0.0215014  [0.0214117, 0.0215915]
2 - 2705924 ~ 0.270592   [0.270317, 0.270868]
3 - 4250570 ~ 0.425057   [0.424751, 0.425363]
4 - 2376093 ~ 0.237609   [0.237346, 0.237873]
5 -  452399 ~ 0.0452399  [0.0451113, 0.0453689]
```

But if we only care about how many equal valued dice we roll, why should we fix our target number? We now allow ourselves to choose a new target number each roll. The distribution of the number of successes is now given by the output:

```
1 -    8150 ~ 0.000815   [0.000797503, 0.000832881]
2 - 2560810 ~ 0.256081   [0.255811, 0.256352]
3 - 4523859 ~ 0.452386   [0.452077, 0.452694]
4 - 2446456 ~ 0.244646   [0.244379, 0.244912]
5 -  460725 ~ 0.0460725  [0.0459427, 0.0462026]
```

This can clearly be seen to be an improvement, as expected.

The above rules allow you to reroll any dice you choose. However, some games require that you "freeze" dice that you cannot reroll. As an example we change the rules in two ways. First, we are not limited to three rolls. But second, we must freeze at least one die each roll – if there are no dice that we want to freeze, then we have to freeze one die that we do not want.

First, we revert to a fixed target, set after the first roll, and thus we do not care which unwanted die we freeze. We stop when all dice are frozen. The distribution is now given by the output:

```
1 -  149159 ~ 0.0149159  [0.014841, 0.0149912]
2 - 2623375 ~ 0.262338   [0.262065, 0.26261]
3 - 4214101 ~ 0.42141    [0.421104, 0.421716]
4 - 2513520 ~ 0.251352   [0.251083, 0.251621]
5 -  499845 ~ 0.0499845  [0.0498496, 0.0501197]
```

Second, we allow changing target number while having unlimited rolls, but having to freeze at least one die per turn. This is a real case, found in the games *To Court the King*[10] and its derivative *Favor of the Pharaoh*[11]. Note that in those games you do not always have five dice, you have a variable number, starting with three. However for this example we continue with five dice. You also might have other options in those games that are not included in this example.

[9] Unknown designer, E.S. Lowe Company, 1956.

[10] Thomas Lehman, Rio Grande Games, 2006.

[11] Thomas Lehman, Bézier Games, 2015.

Unfortunately, a perfect answer to this problem is not possible, because as things stand, it is not well defined. Suppose we make a first roll of 1,2,2,4,4, and freeze 4,4. Then suppose that our three new dice are 5,5,5. Should we freeze all of these three dice, improving our result from 2 to 3 matching dice, but making a result of 4 matching dice impossible, or should we freeze just one die – because we must – and risk a result of only 2 matching dice, but leaving a result of 4 matching dice possible? That depends on how much we value results of 2, 3, and 4 matching dice.

What we can do is to come up with two possible approaches that make those two different choices, and see how their results compare. Full details of the two approaches are not presented here, only summaries of them; they include making intelligent choices when freezing mismatching dice.

The first approach freezes as many dice as it can of a single target value, and only ever freezes one die of any other number; it only changes target value if another number "catches up", which can only be one die at a time. The second approach freezes all of any die that we currently have a maximum number of, frozen and unfrozen, but freezes as few dice as possible when doing that.

The distributions in these two cases are given, respectively, by the output:

```
1 -   11063 ~ 0.0011063  [0.00108589, 0.0011271]
2 - 2414225 ~ 0.241423   [0.241157, 0.241688]
3 - 4546403 ~ 0.45464    [0.454332, 0.454949]
4 - 2528430 ~ 0.252843   [0.252574, 0.253112]
5 -  499879 ~ 0.0499879  [0.049853, 0.0501231]
```

and the output:

```
1 -     540 ~ 5.4e-05    [4.96295e-05, 5.87546e-05]
2 - 2640903 ~ 0.26409    [0.263817, 0.264364]
3 - 4483939 ~ 0.448394   [0.448086, 0.448702]
4 - 2374822 ~ 0.237482   [0.237219, 0.237746]
5 -  499796 ~ 0.0499796  [0.0498447, 0.0501148]
```

Comparing the two outputs, which approach to prefer depends on your objective. For example, the second approach greatly reduces the probability of a result of 1, but also reduces the probability of a result of 4 – the probability of a result of 5 is, necessarily, the same in both cases.

Bridge

What is the distribution of the length of the longest suit in a *bridge*[12] hand? A bridge hand is 13 cards from a standard 52 card deck containing 13 cards of each of 4 suits. The exact answer is:

```
 4 - 222766089260 ~ 0.350805    = 11138304463/31750677980
 5 - 281562853572 ~ 0.443397    = 10055816199/22679055700
 6 - 105080049360 ~ 0.165477    = 1313500617/7937669495
 7 -  22394644272 ~ 0.0352664   = 11761893/333515525
 8 -   2963997036 ~ 0.00466761  = 105857037/22679055700
 9 -    235237860 ~ 0.000370445 = 11761893/31750677980
10 -     10455016 ~ 1.64642e-05 = 1306877/79376694950
11 -       231192 ~ 3.64074e-07 = 28899/79376694950
12 -         2028 ~ 3.19363e-09 = 507/158753389900
13 -            4 ~ 6.29908e-12 = 1/158753389900
```

I did not make the program deal 635013559600 hands, that would take impossibly long. Instead, that output used a more intelligent feature of the program to produce those results. However the next results have to revert to being approximate, at least with the default version of the program.

[12]Contract bridge, Harold Stirling Vanderbilt and others, 1925, based on earlier whist and bridge games for which the same analysis applies.

Now deal 13 cards to all four players; what is the distribution of the length of the longest suit in any player's hand? Output, including one set of accumulated output, is:

```
 4 -  292777 ~ 0.0292777  [0.0291734, 0.0293824]
 5 - 4020711 ~ 0.402071   [0.401767, 0.402375]
 6 - 4225035 ~ 0.422503   [0.422197, 0.42281]
 7 - 1262658 ~ 0.126266   [0.12606, 0.126472]
 8 -  183236 ~ 0.0183236  [0.0182407, 0.0184069]
 9 -   14882 ~ 0.0014882  [0.0014645, 0.00151229]
10 -     684 ~ 6.84e-05   [6.34591e-05, 7.3725e-05]
11 -      17 ~ 1.7e-06    [1.03953e-06, 2.74462e-06]

>=  4 - 10000000 ~ 1          [1, 1]
>=  5 -  9707223 ~ 0.970722   [0.970618, 0.970827]
>=  6 -  5686512 ~ 0.568651   [0.568344, 0.568958]
>=  7 -  1461477 ~ 0.146148   [0.145929, 0.146367]
>=  8 -   198819 ~ 0.0198819  [0.0197956, 0.0199686]
>=  9 -    15583 ~ 0.0015583  [0.00153404, 0.00158294]
>= 10 -      701 ~ 7.01e-05   [6.50958e-05, 7.54882e-05]
>= 11 -       17 ~ 1.7e-06    [1.03953e-06, 2.74462e-06]
```

About 14.6% of the time, or slightly more than one in seven sets of hands, someone should have a seven or more card suit. 12 and 13 card suits are possible but have not been seen in these results.

The main documentation has some other examples drawn from bridge hands.

Poker

We evaluate a random five card *poker*[13] hand as a high card hand, one pair, two pairs, three of a kind, a straight, a flush, a full house, four of a kind or a straight flush.

The distribution of these types of hands is given by the output:

```
High card       - 1302540 ~ 0.501177    = 1277/2548
One pair        - 1098240 ~ 0.422569    = 352/833
Two pairs       -  123552 ~ 0.047539    = 198/4165
Three of a kind -   54912 ~ 0.0211285   = 88/4165
Straight        -   10200 ~ 0.00392465  = 5/1274
Flush           -    5108 ~ 0.0019654   = 1277/649740
Full house      -    3744 ~ 0.00144058  = 6/4165
Four of a kind  -     624 ~ 0.000240096 = 1/4165
Straight flush  -      40 ~ 1.53908e-05 = 1/64974
```

The main documentation has some other poker examples relating to the poker game *Texas hold'em*.

Cribbage

We evaluate a random four card *cribbage*[14] hand plus random turnup card. The average value of the hand is given by the exact output:

```
Number of results          = 12994800
Mean                       = 4.76915 = 60759/12740
Standard deviation         = 3.12544
Minimum result             = 0
Maximum result             = 29
```

[13]Origins uncertain, developed in the 18th and 19th centuries.

[14]Credited to Sir John Suckling, 16th century, as a development of an earlier game *noddy*.

```
Median                      = 4
Mode                        = 4
```

with distribution given by the exact output:

```
 0 - 1009008 ~ 0.0776471   = 33/425
 1 -   99792 ~ 0.00767938  = 297/38675
 2 - 2813796 ~ 0.216532    = 234483/1082900
 3 -  505008 ~ 0.0388623   = 1503/38675
 4 - 2855676 ~ 0.219755    = 237973/1082900
 5 -  697508 ~ 0.0536759   = 24911/464100
 6 - 1800268 ~ 0.138538    = 450067/3248700
 7 -  751324 ~ 0.0578173   = 26833/464100
 8 - 1137236 ~ 0.0875147   = 284309/3248700
 9 -  361224 ~ 0.0277976   = 15051/541450
10 -  388740 ~ 0.029915    = 6479/216580
11 -   51680 ~ 0.00397698  = 38/9555
12 -  317340 ~ 0.0244205   = 5289/216580
13 -   19656 ~ 0.00151261  = 9/5950
14 -   90100 ~ 0.00693354  = 53/7644
15 -    9168 ~ 0.000705513 = 191/270725
16 -   58248 ~ 0.00448241  = 2427/541450
17 -   11196 ~ 0.000861575 = 933/1082900
18 -    2708 ~ 0.000208391 = 677/3248700
20 -    8068 ~ 0.000620864 = 2017/3248700
21 -    2496 ~ 0.000192077 = 4/20825
22 -     444 ~ 3.41675e-05 = 37/1082900
23 -     356 ~ 2.73956e-05 = 89/3248700
24 -    3680 ~ 0.00028319  = 46/162435
28 -      76 ~ 5.84849e-06 = 19/3248700
29 -       4 ~ 3.07815e-07 = 1/3248700
```

A score of 19 is impossible, and thus in cribbage slang, a “nineteen hand” is one scoring zero.

Of course real cribbage scores are much higher. That is because you do not just get four cards, you get six cards and keep four of your choice. A fully intelligent algorithm to decide which cards to discard would include considering the effects of those discards (which go into a box or crib owned by one player) and what to keep for pegging, another part of the game, and might also depend on the state of the game, especially if close to finishing. These are all beyond the scope of this program.

However, playing to simply maximise your own average score in hand can be analysed. This is not the best strategy, but a reasonable starting point – and no strategy can get higher average values in hand. Average values in this case are given by the approximate output:

```
Number of results           = 100000
Mean                        = 8.28819
Standard deviation          = 2.62886
Standard deviation of mean = 0.00831318
95% confidence interval     = [8.27189, 8.30448]
Minimum result              = 2.13043
Maximum result              = 22.7826
```

There is quite a lot of work in evaluating this, as each hand needs to consider the 15 possible discards and the 46 possible turnup cards, so these results are for just a hundred thousand hands, although a million hands would be possible while still requiring a practical time to have to wait.

The mean score has gone up from about 4.77 to about 8.3.

The approximate distribution of hand scores using this approach is given by the output:

```
 0 -   339 ~ 0.339%   [0.304782%, 0.377033%]
 1 -    37 ~ 0.037%   [0.0266946%, 0.0511439%]
 2 -  3904 ~ 3.904%   [3.7857%, 4.02584%]
 3 -  1205 ~ 1.205%   [1.1392%, 1.27455%]
 4 - 11340 ~ 11.34%   [11.145%, 11.538%]
 5 -  3281 ~ 3.281%   [3.17236%, 3.39323%]
 6 - 15201 ~ 15.201%  [14.9798%, 15.4249%]
 7 -  6751 ~ 6.751%   [6.59714%, 6.90818%]
 8 - 18942 ~ 18.942%  [18.7003%, 19.1861%]
 9 -  7232 ~ 7.232%   [7.07309%, 7.39419%]
10 -  9133 ~ 9.133%   [8.95601%, 9.31313%]
11 -  1674 ~ 1.674%   [1.5963%, 1.75542%]
12 - 10524 ~ 10.524%  [10.3353%, 10.7157%]
13 -   838 ~ 0.838%   [0.783327%, 0.89645%]
14 -  3707 ~ 3.707%   [3.59165%, 3.8259%]
15 -   564 ~ 0.564%   [0.519407%, 0.612391%]
16 -  3279 ~ 3.279%   [3.17039%, 3.3912%]
17 -   704 ~ 0.704%   [0.654005%, 0.757782%]
18 -   206 ~ 0.206%   [0.179682%, 0.236144%]
20 -   585 ~ 0.585%   [0.539557%, 0.63424%]
21 -   177 ~ 0.177%   [0.152722%, 0.205106%]
22 -    44 ~ 0.044%   [0.0326409%, 0.0591971%]
23 -    29 ~ 0.029%   [0.020023%, 0.0418161%]
24 -   299 ~ 0.299%   [0.266962%, 0.334856%]
28 -     5 ~ 0.005%   [0.00176471%, 0.0120762%]
```

A score of 29 is possible, but a hundred thousand hands is not enough for it have occurred.

And They're Off

The horse racing game *And They're Off*[15] is a variant of a simpler traditional game.

We have four horses, represented by the four suits of a deck of cards, with thirteen cards in each suit. We start by revealing seven cards at random, with no more than five in any suit. But here we do not reveal those seven cards at random, rather each different pattern of those seven cards requires a separate analysis. Here we just analyse one pattern, that where horse 0 has 0 cards revealed, horse 1 has 1 card revealed, horse 2 has 2 cards revealed and horse 3 has 4 cards revealed.

We draw cards, in each case moving the matching horse one space forward, until we have moved one horse eight spaces forward. If this were all there were to the game the winning probabilities of the four horses would be as given by the following ten million game output.

This output also shows the program producing percentage results and another form of odds output:

```
0 - 4660237 ~ 46.6024%  (1.14581 : 1)  [46.5715%, 46.6333%]
1 - 3095065 ~ 30.9507%  (2.23095 : 1)  [30.922%, 30.9793%]
2 - 1859851 ~ 18.5985%  (4.37677 : 1)  [18.5744%, 18.6226%]
3 -  384847 ~ 3.84847%  (24.9844 : 1)  [3.83657%, 3.86041%]
```

However there is an added feature of *And They're Off* that is not in the traditional game. There six *Horse Hampered* cards added to the deck. When we draw one of these cards, we push the last moved horse back one space, but not before the start line.

[15]Charlie Brooks, Andrew Pye-Jeary and Andrew Lloyd Webber, The Really Useful Games Company, 1993.

With the six Horse Hampered cards added to the deck, the winning probabilities of the four horses are given by the output:

```
0 - 4622909 ~ 46.2291%  (1.16314 : 1)  [46.1982%, 46.26%]
1 - 3095043 ~ 30.9504%  (2.23097 : 1)  [30.9218%, 30.9791%]
2 - 1881779 ~ 18.8178%  (4.31412 : 1)  [18.7936%, 18.842%]
3 -  400269 ~ 4.00269%  (23.9832 : 1)  [3.99056%, 4.01486%]
```

Adding these cards has made a change, but not much, other than adding to the tension in the game.

Why are the horses not equally likely? Because the game as played proceeds by first revealing the seven cards, then players bet on horses, then the race is run and those players that backed the winning horse collect from their bets. The odds at which they collect depend on the number of revealed cards for that horse. Note that the actual odds also depend on the distribution of the other cards, but the game does not take that into account.

The odds in the game are for 0 cards evens (1:1), for 1 card 2:1, for 2 cards 3:1, and for 4 cards 16:1. (Also there are 3 cards 6:1 and 5 cards 50:1 that do not appear in this example.) These are actually all bad bets if you were playing for money, to make a profit. Played to see who has the most money at the end of the game after several races they might be more reasonable.

Dominion

Dominion[16] is an example of a deck building game in which a player starts with some cards, and adds to his or her cards as the game progresses. Many mechanisms in the real game are not included in this simple example. Here we just derive a result from the mindless – and hence easily implemented – strategy known as “Big Money”, assuming it is not interfered with.

There are five types of basic cards that we consider here: Estate, Province, Copper, Silver and Gold. A player starts with three Estates and seven Copper. Copper, Silver and Gold have purchasing value of 1, 2 and 3, the other cards have a purchasing value of 0.

Using the five cards in our hand – which we keep – we buy the first card on this list that we can afford: Province (cost 8), Gold (cost 5) and Silver (cost 3). Bought cards and all of our hand go in our discard pile, then we draw five new cards from our draw deck, reshuffling our discard pile as a new draw deck when necessary. For convenience for these results we stop when we have purchased four Provinces.

The average value (about 16.8) and distribution of the number of turns that the games takes is given by the following approximate output over a million games:

```
Number of results          = 1000000
Mean                       = 16.8006
Standard deviation         = 1.83614
Standard deviation of mean = 0.00183614
95% confidence interval    = [16.797, 16.8042]
Minimum result             = 10
Maximum result             = 26

10 -     30 ~ 3e-05     [2.08473e-05, 4.29939e-05]
11 -    924 ~ 0.000924  [0.000866305, 0.000985529]
12 -   3347 ~ 0.003347  [0.00323568, 0.00346214]
13 -  14512 ~ 0.014512  [0.0142795, 0.0147483]
14 -  87740 ~ 0.08774   [0.0871871, 0.0882961]
15 - 143410 ~ 0.14341   [0.142724, 0.144098]
16 - 194722 ~ 0.194722  [0.193947, 0.195499]
17 - 219991 ~ 0.219991  [0.21918, 0.220804]
```

[16]Donald X. Vaccarino, Rio Grande Games, 2008.

```
18 - 154877 ~ 0.154877  [0.154169, 0.155587]
19 - 100634 ~ 0.100634  [0.100046, 0.101225]
20 -  54703 ~ 0.054703  [0.054259, 0.0551504]
21 -  21434 ~ 0.021434  [0.021152, 0.0217197]
22 -   2546 ~ 0.002546  [0.0024491, 0.00264672]
23 -    789 ~ 0.000789  [0.000735819, 0.000846016]
24 -    275 ~ 0.000275  [0.000244309, 0.000309531]
25 -     59 ~ 5.9e-05   [4.56232e-05, 7.62178e-05]
26 -      7 ~ 7e-06     [3.0668e-06, 1.47746e-05]
```

The main documentation takes this further, giving an example that improves on Big Money.

Monty Hall

In the infamous *Monty Hall*[17] problem, a game show contestant is offered a choice of three doors, behind which there are a car and two goats, the position of the car being randomly chosen and unknown to the contestant. In the problem – the game show may have varied this – the contestant, who would rather win a car than a goat, is allowed to pick one door, but does not open it. The host – Monty Hall – who does know where the car is, then always opens one of the doors that the contestant has not chosen to reveal a goat, but reveals no other information to the contestant. If Monty could open either of the two unchosen doors then he opens one at random. The contestant is then offered a choice as to stick with his or her original door or to change doors. What should the contestant do?

The answer is to change doors. This can be shown by a clear argument, but another way is to simulate it. The contestant's probability of success when he changes door is shown by the following approximate, but very accurate, output from ten million playings of the game:

```
Number of results          = 10000000
Number of false results    = 3335142
Number of true results     = 6664858
Probability                = 0.666486
95% confidence interval    = [0.666194, 0.666778]
```

The probability of success is clearly consistent with the expected result of winning two thirds of the time, and not with the often erroneously believed answer of a half.

Parameters Used

Explanation

The following sections, one for each problem section above, include one or more instances of a form of output that the program can produce reporting the parameters provided to the program, introduced by the text `Parameters:`. These parameters are used on a single command line following whatever is needed to be provided on the system that you use to invoke the program.

Where possible the output lines, which need to be joined together, are split between parameters, but in many cases one exceptionally long parameter, known as the main expression, is itself split across lines. The main expression can be recognised by that (in these cases, there can be exceptions) it does not start with a `+` or `-` sign, it does not follow a parameter such as `-u0` or `-v9` and it is either last (if preceded by `-exact`) or is followed only by a number (otherwise).

None of the parameters reported below, including the main expression, contain any spaces (although that is possible). On many systems, some parameters, in particular main expressions, will need to be enclosed in some form of quotes or otherwise modified to be accepted.

[17] Named for the original host of the game show *Let's Make a Deal*, NBC, 1963.

Dungeons and Dragons

```
Parameters: -exact -histogram 3d6
```

```
Parameters: -exact -rcumulative 3d6
```

```
Parameters: -exact -cumulative 3d6
```

```
Parameters: -exact -all sum_range(1,3,sorted4d6)
```

Yspahan

```
Parameters: -exact -histogram count_diff(sorted9d6)
```

```
Parameters: -exact -histogram count_diff(sorted12d6)
```

More Simple Problems

```
Parameters: -exact -probability -statistics
            different(combin4(copy13(sequence4)))
```

```
Parameters: -exact -probability -statistics all_eq(counts(6d6)%2,0)
```

```
Parameters: -exact all v0:=3d{0,2,3,4,5,6};max(v0)-min(v0)
```

```
Parameters: -exact -histogram sum(combin3(copy4(lower(sequence13+1,10))))
```

Can't Stop

```
Parameters: -exact -histogram -percent -odds -u0 {6,7,8}
            s0:=s1:=2;max_xvcombin01(d6,sum_eq(u0,sumv01))
```

```
Parameters: -exact -histogram -percent -odds -u0 {4,7,10}
            s0:=s1:=2;max_xvcombin01(d6,sum_eq(u0,sumv01))
```

```
Parameters: -exact -histogram -percent -odds -u0 {2,3,12}
            s0:=s1:=2;max_xvcombin01(d6,sum_eq(u0,sumv01))
```

Titan

```
Parameters: -histogram -c0 18 -c1 10 -c2 2 -c3 6
            r0:=c0;r1:=c1;until(r0-=count_ge([c1]d6,c2);r1-=count_ge([c0]d6,c3),
            r0<=0|r1<=0);(r0>0)<=>(r1>0) 100000000
```

```
Parameters: -histogram -c0 18 -c1 10 -c2 3 -c3 6
            r0:=c0;r1:=c1;until(r0-=count_ge([c1]d6,c2);r1-=count_ge([c0]d6,c3),
            r0<=0|r1<=0);(r0>0)<=>(r1>0) 100000000
```

```
Parameters: -histogram -c0 18 -c1 10 -c2 4 -c3 6
            r0:=c0;r1:=c1;until(r0-=count_ge([c1]d6,c2);r1-=count_ge([c0]d6,c3),
            r0<=0|r1<=0);(r0>0)<=>(r1>0) 100000000
```

Monopoly

```
Parameters: -noreset -histogram +graph +nogaps -v8 sequence16 -v9 sequence16
            f0[r1:=0;r0:=-1];f1[((r0+5)/10*10+5)%40];f2[r0<12|r0>=29?12:28];f3[
            r0:=get(wdraw8,{0,f0(),11,24,39,5,f1(),f1(),f2(),(r0-3)%40}#r0[6])];
            f4[r0:=get(wdraw9,{0,f0()}#r0[14])];f5[r0==30?f0():(r0==7|r0==22|r0=
            =36)&f3();(r0==2|r0==17|r0==33)?f4():r0];f6[r0+=r2+r3;r0%=40;f5()];
            r0<0&(r0:=10);r1|(r8:=r9:=0);r2:=d6;r3:=d6;(r2==r3?incr1:r1:=0);(r1=
            =3?f0():f6()) 100000000
```

Dice Rerolling Games

```
Parameters: -histogram do3(v0:=upper(v0,5d6));count_eq(v0,6) 10000000
```

```
Parameters: -histogram
            v0:=-1;do3(v0:=(v0==r0?r0:5d6);r0?=mode_max(v0));count_eq(v0,r0)
            10000000
```

```
Parameters: -histogram
            v0:=-1;do3(v0:=(v0==r0?r0:5d6);r0:=mode_max(v0));count_eq(v0,r0)
            10000000

Parameters: -histogram
            until(v0:=(v1?v0:5d6);r0?=mode_max(v0);any(!v1&v0==r0)?v1|=v0==r0:(
            r1:=find_eq(v1,0);e1:=1),all(v1));count_eq(v0,r0) 10000000

Parameters: -histogram
            s0:=5;f0[r1:=find_max(count_eq(v1?v0:0,v1?-1:v0)-v1);e1:=1];f1[r0:=
            find_max(v2);v1|=v0==e0];v0:=d6;v1:=v0==[mode_max(v0)];while(!all(v1
            ),v0:=v1?v0:d6;v2:=v1?0:get(v0-1,count_seq6(v1?v0-1:-1));any(v2)?f1(
            ):f0());count_mode(v0) 10000000

Parameters: -histogram
            s0:=5;f0[r1:=find_max(count_eq(v1?v0:0,v1?-1:v0)-v1);e1:=1];f1[r0:=
            find_min(v2?counts(v2):s0);v1|=v0==e0];v0:=d6;v1:=v0==[mode_max(v0)]
            ;while(!all(v1),v0:=v1?v0:d6;v2:=(v1|counts(v0)<[count_mode(v0)]?0:
            v0);any(v2)?f1():f0());count_mode(v0) 10000000
```

Bridge

```
Parameters: -exact -histogram count_mode(take13from[sequence52%4])

Parameters: -histogram -rcumulative
            u0:=sequence52/13;count_mode(shuffle(u0)+4*u0) 10000000
```

Poker

```
Parameters: -exact -histogram -names
            High\_card,One\_pair,Two\_pairs,Three\_of\_a\_kind,Straight,Flush,
            Full\_house,Four\_of\_a\_kind,Straight\_flush
            v0:=combin5(sequence52);v1:=v0/4;v2:=v0%4;r0:=last(runs(v1))|list_eq
            (v1,{0,1,2,3,12});r1:=same(v2);v3:=groups(v1);r0&r1?8:get(3,v3)?7:!
            first(v3)?6:r1?5:r0?4:get(2,v3)?3:get(1,v3)
```

Cribbage

```
Parameters: -exact +statistics -histogram
            v0:=combin5(sequence52);veswap0(random(5),4);v1:=v0/4;v2:=v0%4;v3:=
            lower(v1+1,10);do31(vincr4;dot(v3,v4)==15&(r0+=2));r1:=dot(groups(v1
            ),{0,2,6,12,0});r2:=dot(runs(v1),{0,0,3,4,5});r3:=find_ne(v2,first(
            v2));r3<4&(r3:=0);r4:=any_eq(head4(v0),last(v2)+40);r0+r1+r2+r3+r4

Parameters: -statistics -scale 1 46
            f0[v9:=q0/4;v8:=q0%4;r9:=find_ne(v8,first(v8));r9<4&(r9:=0);r9+=any_
            eq(head4(q0),last(v8)+40);v8:=lower(v9+1,10);v7:=0;do31(vincr7;dot(
            v7,v8)==15&(r9+=2));r9+=dot(groups(v9),{0,2,6,12,0});r9+=dot(runs(v9
            ),{0,0,3,4,5})];v0:=combin6(sequence52);v1:=skip_seq46(v0);r0:=4;
            wcombin0(r2:=max{r2,vsum1(f0(head4(v0)#{e1}))}) 100000

Parameters: -histogram
            f0[v9:=q0/4;v8:=q0%4;r9:=find_ne(v8,first(v8));r9<4&(r9:=0);r9+=any_
            eq(head4(q0),last(v8)+40);v8:=lower(v9+1,10);v7:=0;do31(vincr7;dot(
            v7,v8)==15&(r9+=2));r9+=dot(groups(v9),{0,2,6,12,0});r9+=dot(runs(v9
            ),{0,0,3,4,5})];v0:=combin6(sequence52);v1:=skip_seq46(v0);r0:=4;r0:
            =4;wcombin0((r3:=vsum1(f0(head4(v0)#{e1})))>=r2&(v2:=head4(v0);r2:=
            r3));v3:=v2#{select(v1)};f0(v3) 100000
```

And They're Off

```
Parameters: -histogram -percent +odds
            u0:={0,1,2,4};s1:=k0;v0:=make_seq(13-u0);until(r1:=wdraw0,ince1==8)
            10000000
```

```
Parameters: -histogram -percent +odds
            u0:={0,1,2,4};s1:=k0;v0:=make_seq(13-u0)#[-1][6];until((r2:=wdraw0)<
            0?e1&dece1:(r1:=r2;ince1),e1==8);r1 10000000
```

Dominion

```
Parameters: -statistics -histogram
            f0[v1:=v0-v2;r4:=r3-r5];f1[r1:=r2:=distribute(v1);dece1;ince2;decr4;
            incr5];f2[r4|f0();f1()];f3[v2:=0;r5:=0;do5(f2())];f4[r0:=p0;ince0;
            incr3];f5[(r8:=p0)<0|f4(r8)];f6[r9:=dot(v2,{0,0,0,1,2,3});r9>=8?2:r9
            >=6?5:r9>=3?4:-1];f7[f5(f6())];v0:=v1:={3,0,0,7,0,0};r3:=r4:=10;
            until(f3();f7();incr6,r0:=2;e0==4) 1000000
```

Monty Hall

```
Parameters: -probability -statistics
            u0:=sequence3;r0:=select(u0);r1:=select(u0);r2:=skip_select(u0,r01);
            r3:=skip_first(u0,r12);r3==r0 10000000
```