Liar’s dice is a great game. Easy to learn, difficult to master. A good balance of luck and strategy. If you don’t know the game, I encourage you to check it out, it’s great for a group.

In our last session, we encountered an interesting situation when we got down to the final two. One player had all 6 of his remaining dice while his opponent was down to one. Of course this led to much discussion and side betting. Mainly, we set out *(read: argued)* to answer the following questions:

> Can the player with all six dice devise an optimal strategy to guarantee a victory?

> If not, what are the odds for the player with only one dice?

And just like that, a bet was born. A bet not even involving the two players actually playing the game.

The Strategy

Before we get to the bet, we needed to agree on what the strategy of the player with all 6 dice would do. We pretty quickly arrived that, in order to avoid ever “bluffing,” the person with 6 should just call the highest thing they can possibly call and hope that the opponent with 1 die won’t be able to beat it. So, given dice `1, 1, 3, 4, 4, 5`, he should call four `4`s (`1`s are wild).

The Bettors

After taking minimal game theory courses in college, I tend to assume there is always some kind of strange optimal strategy. I was unable to find said strategy but I of course immediately started trying. My goal was to find some consistent strategy (one that did not allow for variable bluffing, despite it being a key component of the game) and then calculate the odds of how often it would win. A statistical (and thus, sure fire, right?) approach to the problem.

On the other side of this bet was my friend Kurt. Kurt, a long time card-player, doesn’t think about the math right away. Instead, he rapidly starts running through thousands of iterations in his head of how the game might play out. His intuition about odds and seeing the game so many times allows him to determine how often each player will win.

The Bet

My quick math showed me that the person with all 6 dice would only lose if the opponent had the same number they called, or had a `1` (wild card). Given `1, 1, 3, 4, 4, 5` and a call of four `4`s, the underdog only wins when he has a `4` or a `1`, thus allowing him to call five `4`s and win. Rolling a `1` and another specific roll will happen 1 in 3 times and it needs to happen 6 times in a row in order to lose. In other words, only 1 in 729 times will the underdog win.

Kurt did some other kind of gambler's voodoo magic in his head and boldly exclaimed "if you give me 1 dice against your 6 15 times in a row, I bet you I win one." Me having just calculated the odds with 100% certainty (maybe not…) saw an opportunity to strike. I replied "you're on" so quickly that Kurt second-guessed himself based on my confidence. I balked a bit at his request for 2:1 odds (he's already getting odds by running it 15 times) but eventually gave in, thinking I was still a statistical lock.

6 dice vs 1 die. 15 times in a row.

Time to play.

The Result

The first few turns went exactly according to plan. I'd call something like three `5`s, Kurt has a `3`, game over.

For the most part, the first 10 games went great. Kurt rolled a couple `1`s in there but would pretty much lose on the next turn every time. But then, somewhere around the 10th game, something happened.

I rolled `1 1 3 4 4 5` and, per the strategy, called four `4`s, the most I could safely call. Kurt then comes back with four `5`s, having rolled his own `5`.

Oh, crap.

I lost, and he didn’t roll a duplicate or a `1`. I ended up winning that game anyways, but he did get me down to three dice. I went through the next few games no problem and won the bet. But my interest had been piqued. My assumptions, and thus my odds, weren’t correct. I must find out what they actually are.

Simulation

Now realizing the problem is possibly more complicated than simple statistics can solve, I set out to write a simple simulation script that could run many many trials to get an estimate of the odds. It also seemed to be a good opportunity to test out Jupyter Notebooks, something I had been meaning to do for a while. All of the code and iPython notebooks for this project is available on GitHub.

Building out the dice randomization, general gameplay flow, and my simple rule-based-no-bluffing strategy wasn’t too tricky. The real variable in this situation is the strategy that Kurt should use.

Run 1 — The One-Upper

As a sanity check, I wanted to check my original odds and give Kurt a basic strategy. Whatever I bid, Kurt bids one more of that and hopes that he has a `1` or whatever number I called. Not smart, but a good starting point.

```python
def get_kurts_bid(my_dice, num_dice_matt_has, matts_bid=None):
   if matts_bid is None:
       # If Matt has not bid yet start off with 1 of my dice
       return (1, my_dice)
   else:
       # Return one more of what Matt said
       return (matts_bid[0] + 1, matts_bid[1])
```

Results:

```
Simulation Complete
-------------------
Num Simulations: 10,000,000
Took:            247.9 seconds
Kurt Wins:       13,742
Win Percent:     0.14%
Wins 1 in 727.70 times
```

Sure enough, 10 million simulations show that my odds of 1 in 729 were right. It also, perhaps more importantly, shows that the code is working. Time to start optimizing.

Run 2 — The Good Guesser

Maybe we can do better by trying to trap Matt on the initial bid.

Rather than letting Matt drive the action, Kurt can try and guess the number that Matt would have called anyway. An initial guess of three of whatever die Kurt rolls seems like a good guess.

```
def get_kurts_bid(my_dice, num_dice_matt_has, matts_bid=None):
   if my_dice == 1:
       if matts_bid is None:
           return (1, 2)
       else:
           return (matts_bid[0] + 1, matts_bid[1])
   else:
       if matts_bid is None:
           return (3, my_dice)
       else:
           return (matts_bid[0] + 1, matts_bid[1])
```

Results:

```
Simulation Complete
-------------------
Num Simulations: 10,000,000
Took:            243.0 seconds
Kurt Wins:       14,365
Win Percent:     0.14%
Wins 1 in 696.14 times
```

Not much improvement, dropped to 1 in 696 times for Kurt to win.

This actually makes sense, mostly because of gameplay. Once Kurt wins the first turn, Matt gets to go first in subsequent turns. So really this strategy only affects the very first turn of the game.

Run 3 — The End Gamer

Since the previous strategy only affected the first turn, let’s see if we can be smarter towards the end of the game, if we get there.

When it gets down to Matt having two or less dice Kurt wants to guess two of whatever his dice is. The hope is Matt rolls `1, 3` and Kurt rolls `4`. Matt guesses two `3`s, Kurt comes back with two `4`s and wins.

```
def get_kurts_bid(my_dice, num_dice_matt_has, matts_bid=None):
    # When Matt has 2 or less dice, guess 2 of whatever I have if I can, otherwise guess 3 of them
    if num_dice_matt_has <= 2:
        if (2, my_dice) > matts_bid:
            return (2, my_dice)
        return (3, my_dice)

    if my_dice == 1:
        if matts_bid is None:
            return (1, 2)
        else:
            return (matts_bid[0] + 1, matts_bid[1])
    else:
        if matts_bid is None:
            return (3, my_dice)
        else:
            return (matts_bid[0] + 1, matts_bid[1])
```

Results:

```
Simulation Complete
-------------------
Num Simulations: 10,000,000
Took:            243.5 seconds
Kurt Wins:       14,831
Win Percent:     0.15%
Wins 1 in 674.26 times
```

A little better, but we're still not doing that much damage. Kurt is only getting to the final two dice about 1 in 80 times anyway. Any strategy we do there doesn't do too much to help overall.

Run 4 — The Finger Crosser

Let's apply the situation that happened during the real game. As a refresher, Matt rolled `1 1 3 4 4 5` and called four `4`s. Then Kurt then came back with four `5`s, having rolled his own `5`. So Kurt's strategy is to guess the same quantity that Matt bids, but of the number he has instead. The hope is that Matt has only one natural roll of that number and the rest `1`s.

```
def get_kurts_bid(my_dice, num_dice_matt_has, matts_bid=None):
    # When Matt has 2 or less dice, guess 2 of whatever I have if
I can, otherwise guess 3 of them
    if num_dice_matt_has <= 2:
        if (2, my_dice) > matts_bid:
            return (2, my_dice)
        return (3, my_dice)




    if my_dice == 1:
        if matts_bid is None:
            return (1, 2)
        else:
            return (matts_bid[0] + 1, matts_bid[1])
    else:
        if matts_bid is None:
            return (3, my_dice)  # Might as well try and trap him
        elif (matts_bid[0], my_dice) > matts_bid:
            return (matts_bid[0], my_dice)  # Hoping for [1, 1,
3] against [4]
        else:
            return (matts_bid[0] + 1, my_dice) # Hope he has all
1s?
```

Results:

```
Simulation Complete
-------------------
Num Simulations: 10,000,000
Took:            294.1 seconds
Kurt Wins:       89,389
Win Percent:     0.89%
Wins 1 in 111.87 times
```

Ooooh, progress. Kurt wins 1 in 112 times now. This situation of having all `1`s and one natural actually happens a lot, especially towards the end of the game when there are fewer dice.

In the last case, if Matt calls three `4`s and I have a `3`, for example, Kurt would call four `3`s. From what I can tell, this only happens when Matt rolls all `1`s, so probably not a whole lot.

Run 5 — The Adjuster

Upon thinking more about the last strategy, I realize it's essentially the same as the original end game strategy. Actually, it's an even better end game strategy! The previous end game strategy of guessing 2 of whatever Kurt has (if possible) doesn't account for a Matt roll of `3, 4` and a Kurt roll of `5`. Matt guesses one 4, Kurt should guess one 5 but would previously guess two `5`s.

Let's apply the general game strategy to the end game also.

```
def get_kurts_bid(my_dice, num_dice_matt_has, matts_bid=None):
   if my_dice == 1:
       if matts_bid is None:
           return (1, 2)
       else:
           return (matts_bid[0] + 1, matts_bid[1])
   else:
       if matts_bid is None:
           return (3, my_dice)  # Might as well try and trap him
       elif (matts_bid[0], my_dice) > matts_bid:
           return (matts_bid[0], my_dice)  # Hoping for [1, 1,
3] against [4]
       else:
           return (matts_bid[0] + 1, my_dice) # Hope he has all
1s
```

Results:

```
Simulation Complete
-------------------
Num Simulations: 10,000,000
Took:            301.5 seconds
Kurt Wins:       354,633
Win Percent:     3.55%
Wins 1 in 28.20 times
```

Whaaa!! Kurt wins this way 1 in 28 times!

It kind of makes sense maybe. Kurt just needs to get through a couple turns by rolling a `1` or the same as me and then has to count on me rolling `1`s. It definitely happens, and it turns out it happens about 3.5% of the time.

Outcome

So, 15 runs at 2:1 odds are actually looking pretty good right now.

Even more importantly, it shows that Kurt's intuition was nearly spot on. This is truly fascinating to me, just by drawing on past experiences and seeing so many dice rolls, Kurt was able to almost exactly predict the odds of this complex scenario.

At least I still won the bet.

https://codeburst.io/a-statistical-look-at-liars-dice-the-gambler-vs-the-mathematician-42d3a96a88d1