monty_hall_proof.md

Monty Hall Problem - Proof Using Python

The Monty Hall Problem is a relatively famous brain teaser and probability puzzle based on an American game show called *Let's Make a Deal* and named after it's host, Monty Hall. The problem was first sent in as a letter to the *American Statistician* in 1975 and its solution received notable backlash. This was the question that was submitted:

- Suppose you're on a game show, and you're given the choice of three doors. Behind one door is a car and behind the two others is a goat. You pick a door, say door A, and the host, who knows what's behind the doors, opens another door, say door C, which has a goat. He then says to you, "Do you want to switch your pick to door B?" Is it to your advantage to switch your choice to door B?

At first, you may think the obvious - that there is no better chance if you switch your door than if you remained with your first choice. I created this python simulation to help explain why this is not quite the case. In fact, your odds are twice as likely if you were to switch to the other door than if you were to remain with your original choice!

The Rules of the Game

It is important to remember three very important rules to this brain teaser:

1. The host must always open a door that was not picked by the contestant
2. The host must always open a door that reveals a goat and never the car (so that he does not simply give away the location of the car)
3. The host must always offer the chance to switch between the originally chosen door and the remaining closed door

The Proof

Just to be clear, this "proof" is more of an explanation and walk-through of the problem than a formal mathematical proof. It is meant to explain the problem through logical steps, if's and else's, that sort of thing. Hopefully this approach will make sense as we go along.

Generating a Set of Games

The first step in the process is to generate our sample size: 100,000 games. In each game scenario, we have put our new car prize behind one of three doors: A, B, or C. Thus, the probability of the car being behind each door is:

```
P(A) = P(B) = P(C) = 1/3
```

Conversely, the probability that a goat is behind each door is:

```
P(A') = P(B') = P(C') = 2/3
```

The 'games' list will contain a list of binary responses for where the door is located - a '1' will indicate that the car is behind that door and a '0' will indicate that a goat is behind that door.

```
doors = ["A", "B", "C"]
games = []
for i in range(100000):
    # Randomly create a winning door, either A, B, or C
    winning_door = random.choice(doors)
    if winning_door == "A":
        games.append([1, 0, 0])
    elif winning_door == "B":
        games.append([0, 1, 0])
    else:
        # winning_door is "C"
        games.append([0, 0, 1])
```

For simplicity of the proof, let's assume that the contestant always chooses door A first. The door that is chosen is irrelevant, since we know that there is a 1/3 chance that the car is behind each door, but this makes the proof a bit easier to follow.

Scenario 1: Contestant Does Not Switch Doors

As stated above, we assume that the contestant chooses door A. The game show host must then open a different door (B or C) to reveal a goat (Rules 1 and 2). The contestant is offered to switch to the remaining door (Rule 3), **but they refuse and remain with door A**.

Thus, in order to find the success rate of door A, we sum the binary values located in the representation of door A for each game simulation in our 'games' list. In other words, we add the 0 index of each game in the 'games' list because the 0 index pertains to door A.

If the value at index 0 is 1, then the contestant was correct in sticking with their door and they won the car! If the value is 0, however, then the contestant was incorrect and should have switched their door.

The total of adding all index 0 binary values determines our success rate out of 100,000 game scenarios.

```
total_door_A_wins = 0
for i in range(100000):
    total_door_A_wins += games[i][0]

print(f"Door 'A' Success Rate: {total_door_A_wins/100000}")
```

If our probability of the car being behind door A is correct, then the success rate should be roughly 33.3%, or 1/3.

Scenario 2: Contestant Switches Doors

Next, we test the 'swtich door' approach. Like before, for simplicity, we assume that the contestant always chooses door A first. The game show host must then reveal the contents behind either door B or C and the door revealed must not contain the car (Rules 1 & 2).

This is simulated in the following loop:

```
switch_total = 0
for i in range(100000):
    # Find which doors of B and C contain a goat
    goat_doors = []
    if games[i][1] == 0:
        goat_doors.append("B")
    elif games[i][2] == 0:
        goat_doors.append("C")

    # Randomly choose a door that has a goat behind it to be revealed
    revealed_door = random.choice(goat_doors)

    # Switch to the door that is not door A, the original choice, and is not
    # the revealed_door

    if revealed_door == "B":
        # Door B is revealed, so we switch to the contents of door C
        switch_total += games[i][2]

    else:
        # Door C is revealed, so we switch to the contents of door B
        switch_total += games[i][1]

print(f"Switch Door Success Rate: {switch_total/100000}")
```

If our hypothesis is correct, the success rate after switching will be roughly 66.67%, or 2/3, ***double*** the rate of not switching doors!

But...Why?

Well, let's think back to our initial probabilities:

```
P(A) = P(B) = P(C) = 1/3
```

It suffices to say that since there's a 1/3 chance of the car being behind door A, then there's a 2/3 chance of the car being behind door B or C.

If A is our first guess, then the host *must* reveal either door B or C. There's a 2/3 chance of it being behind B *or* C, but we now only have one door to choose from, since the other has been revealed. If we switch to the remaining door, this 2/3 chance has not changed, since the door is still *either* door B or C, just like before.

The key here is that the host was not allowed to reveal the contents of what was behind door A, so it's probability of being correct remains the same as it was at the beginning of the game.

Still Not Convinced?

Let's try one last approach, involving each of the three possibilities for where the car is located and still assuming that the candidate chooses door A first.

1. The car is behind door A
    - Door A is chosen by the candidate
    - The host reveals the contents of door B or C (let's say B)
        - Contestant does not switch: **WINNER**
        - Contestant switches to door C (only remaining door): *LOSER*
2. The car is behind door B
    - Door A is chosen by the candidate
    - The host reveals the contents of door C (cannot reveal door B because of Rule 3)
        - Contestant does not switch: *LOSER*
        - Contestant switches to door B (only remaining door): **WINNER**
3. The car is behind door C
    - Door A is chosen by the candidate
    - The host reveals the contents of door B (cannot reveal door C because of Rule 3)
        - Contestant does not switch: *LOSER*
        - Contestant switches to door C (only remaining door): **WINNER**

- **Success rate for not switching: 1/3**
- **Success rate for switching: 2/3**

∴ the contestant is ***twice as likely*** to win if they switch doors.

Run the python script to see if it's true!

Sample output for the lazy:

```
Door 'A' Success Rate: 0.33447
Switch Door Success Rate: 0.66553
```

Author

- **Jimmy Dudley** - website - email - github