The Monty Hall Problem: Simulation and Generalization

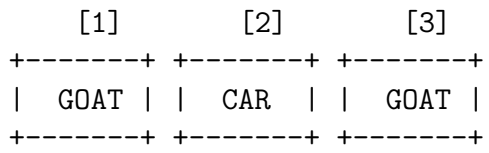
Sumit Sah
Texas State University
Instructor: Prof. Thomas M Keller

April 29, 2025

Introduction

The Monty Hall problem is probably one of the most controversial (and confusing!) topics in probability. If you've ever browsed Reddit threads or math forums about it, you've likely seen heated debates, passionate arguments, and people absolutely refusing to believe the correct answer — even after seeing the math.

The basic setup is simple: you're on a game show, and you're asked to pick one of three doors. Behind one door is a car, and behind the other two, goats. After you pick a door, the host — who knows what's behind every door — opens one of the remaining two doors and reveals a goat. You're then asked: "Do you want to switch your choice to the other unopened door?"



Most people say, "Why switch? It's 50-50 now." But that's actually not true. Mathematically, switching gives you a 2/3 chance of winning, while staying only gives you 1/3. It sounds crazy — and that's exactly why people argue about it online all the time.

For me, it started to make sense when my professor explained it in class. But even though the math checked out, I'm also a programmer — and I really wanted to see it in action. So I decided to write a C++ program to simulate the game and test the results for myself. Then I went further and built a general version with more than three doors, just to see how wild the probabilities could get.

This report walks through both the 3-door classic and the generalized version using interactive games and simulations. Let's go!

Theoretical Explanation

Classic 3-door Version

Before diving into the test results and outputs, it's important to understand what the Monty Hall problem predicts mathematically.

In the classic 3-door version, when a player initially picks a door, there is a $\frac{1}{3}$ chance (approximately 0.333) that they have chosen the car, and a $\frac{2}{3}$ chance (approximately 0.666) that the car is behind one of the other two doors. After Monty, who knows where the car is, opens one of the other doors to reveal a goat, the probability does not split evenly between the remaining two doors. Instead, the full $\frac{2}{3}$ probability shifts to the other unopened door.

Thus, if the player decides to stay with their original choice, their chance of winning remains $\frac{1}{3}$ (around 33.3%). However, if they switch, their chance of winning jumps to $\frac{2}{3}$ (about 66.6%). This is what makes the Monty Hall problem so counterintuitive and controversial.

Generalized Version

In the generalized version with N doors, the same logic extends. The probability that the player initially picks the car is $\frac{1}{N}$, and the probability that the car is behind one of the other doors is $\frac{N-1}{N}$. After Monty opens all possible goat doors, leaving only two doors unopened, the player faces the same choice: stay or switch.

If the player switches in the generalized case, their probability of winning becomes $\frac{N-1}{N}$. For example:

- With 10 doors, switching wins approximately 90% of the time.
- With 100 doors, switching wins approximately 99% of the time.

In contrast, if the player stays with their original choice, their chance of winning stays at just $\frac{1}{N}$.

The takeaway is clear: as the number of doors increases, switching becomes an even stronger strategy. The simulations in the next section will put this theory to the test.

What I Built

To test the Monty Hall problem properly, I wrote four different C++ programs — two for the classic 3-door version, and two for a more general version that works with any number of doors.

Here's a quick breakdown:

- `monte_game_3doors.cpp` — This one is the interactive version where you actually play the game with 3 doors. You pick a door, Monty reveals a goat, and then you decide whether to switch. It prints out the result and lets you keep playing.

- `monte_sim_3doors.cpp` — This version runs thousands of simulations automatically (you get to choose how many), and shows the win rate if you always switch vs if you always stay.

- `monte_game_ndoors.cpp` — This is where things get interesting. You can choose how many doors to play with (minimum 3), and Monty will open all the goat doors he can — except one — just like in the famous "100 doors" explanation. Then you're left with two doors and the same question: do you switch?

- `monte_sim_ndoors.cpp` — Just like the 3-door sim, this one runs automated tests — but with any number of doors. It prints the win percentages for both switching and staying, and it's a cool way to watch switching become more powerful as the number of doors increases.

Each file is short, clean, and does one thing well. I separated them on purpose to make it easy to compare the classic version and the generalized one.

Example Runs and Output

Here are a few screenshots from running the programs.

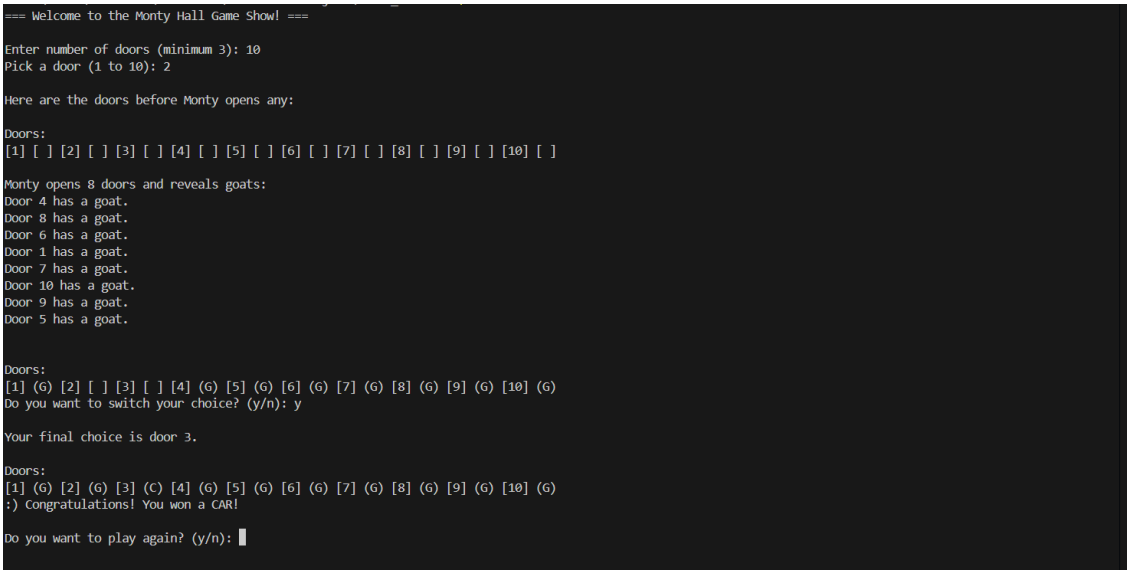
Classic 3-Door Game (User Playthrough)

This is a sample run of the interactive 3-door game. You can see the prompt to pick a door, Monty revealing a goat, and the result after switching or staying.

3-Door Simulation (Auto Mode)

This is the result of running 100,000 simulations with switching vs staying. Switching wins 66.7% of the time — just like the theory says.

General N-Door Game (10 Doors Example)



This is from playing the general version with 10 doors. Monty opens 8 goat doors, leaving you with two unopened ones. The player switches and wins.

N-Door Simulation (100 Doors)

A sample simulation with 100 doors and 100,000 rounds. As expected, switching wins nearly 99% of the time. Crazy, right?

What I Learned

Honestly, the Monty Hall problem is one of those puzzles that messes with your head until you actually see it in action. At first it's like, "Why would switching help? It's 50/50!" But once I ran the simulations and saw that switching really does win about 66.7% of the time (and up to 99% with more doors), it clicked.

Writing the code made the theory real for me. It's one thing to believe the math — it's another to watch your own program prove it, game after game. I also appreciated how adding complexity (like more doors) made the strategy even clearer.

The biggest takeaway? Our intuition isn't always reliable when it comes to probability. But code? Code doesn't lie.

Conclusion

This project was a fun and eye-opening mix of probability and programming. I got to test the Monty Hall problem in different ways, see real statistical results, and push the logic into more generalized territory.

It's definitely one of those "simple on the surface, deep underneath" problems — and it turns out, switching doors really is the way to go. Thanks, Monty.

P.S. If this ever happens in real life, you know what to do. Switch.

Project Repository

The full source code for this project, including all four C++ programs and screenshots, is available on GitHub:

`https://github.com/sumit-sah314/monty-hall-simulation`

Feel free to fork it, test the code, or run your own simulations!