Stay or Switch? My C++ Code Doesn't Lie

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

May 5, 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, say door no. 1, the host — who knows what's behind every door — opens one of the remaining two doors and reveals a goat, say no. 3. You're then asked: "Do you want to stay or switch your choice to the other unopened door?"

Most people say, "It's two doors now? It doesn't matter. 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 video 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

The solution will be more intuitive with more doors rather than just 3. 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. The full source code for this project, including all four C++ programs, is available on my GitHub. Feel free to clone it, fork it, test the code, or run your own simulations as I walk you through the demonstration!

Here's a quick breakdown:

- `monty_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.

  "Okay, let's actually play the Monty Hall game right now. Just once. For real."

  "We've got three doors. I'm going to pick... Door 1."

  "Monty — who knows where the car is — opens Door 3. And it's a goat."

  "Now he asks me: do I want to stay with Door 1 or switch to Door 2?"

  "Since theory says switching gives me a 2/3 chance of winning, I'm going to switch."

  [Pause for effect.]

  "...And I got a goat."

  "Wait — what? Didn't we just say switching was better?"

  "Here's the thing. The math isn't wrong. But this is probability. It doesn't guarantee a win every time — it gives you better odds over many rounds."

  "That's why we can't judge based on just one game. You have to play over and over to see the true pattern emerge.

  This exact phenomenon is part of why the Monty Hall problem became so famous. Many people just couldn't accept the math. In fact, when Marilyn vos Savant first published the correct answer in her *Parade* column in 1990, she received thousands of letters — many from PhDs and mathematicians — insisting she was wrong.

  Years later, When computers advanced simulations and experiments, proved it right! It's just that our intuition doesn't always line up with probability.

  This kind of misunderstanding is known as **base rate neglect**. People tend to ignore the original odds and focus only on the apparent 50/50 choice after a goat is revealed. But Monty's knowledge and actions actually shift the probabilities.

  I played this game 10 times and you can see the results.

  So in the next step, I'm going to run a simulation — thousands of games played automatically — and we'll see if switching really wins more often.

- `monty_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. Say, I want to run 100,000 simulations. This is the result. Switching wins 66 % of the time.

- `monty_game_ndoors.cpp` — This is where things really start to get interesting. You're no longer limited to just three doors — you get to pick how many you want to play with. For this run, I chose 10 doors.

  After I picked one, Monty opened 8 of the remaining doors — all goats, of course. That leaves me with two unopened doors: my original pick, and one other.

  Now it's the same question again: do I stay or do I switch?

  Well, theoretically, the probability of winning by switching in this case is 9 out of 10 — that's 90%. So yes, I switch.

  And... boom! We got a car!

  Of course, it won't always land on the car every single time. But as you keep playing, you'll start seeing the pattern — the probability stacks in favor of switching. You can run this game as many times as you want and you'll see it: switching just wins more often.

- `monty_sim_ndoors.cpp` — Just like the 3-door sim, this one runs automated tests — but with any number of doors. Now let's automate that logic. The file is designed to run simulations with any number of doors — 10, 100, even 1,000 — and tell you exactly how often switching wins.

  For example, let's run it with 10 doors:

  ```
  $ ./monty_sim_ndoors 100000 10
  Simulating 100000 rounds with 10 doors...
  Switching wins: 90004 times (90.00%)
  Staying wins: 9996 times (10.00%)
  ```

  And now with 100 doors:

  ```
  $ ./monty_sim_ndoors 100000 100
  Switching wins: 99014 times (99.01%)
  Staying wins: 986 times (0.99%)
  ```

  This is where the math becomes crystal clear. When you have 100 doors, switching wins almost every time — because the odds of randomly picking the correct door in the beginning are incredibly low.

  It's almost like Monty is handing you the right door — you just have to trust the math and switch.

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. I also created a combined and well-documented version of the program that lets you choose any of the four modes from a single entry point. Whether you want to play the game interactively or run simulations with 3 or N doors, you just select an option, and off you go.

If you're interested in a deeper dive into why Monty Hall solution works, I highly recommend reading the Wikipedia article on the Monty Hall problem. It features some of the clearest explanations from top mathematicians.

For example, Dr. Cecil Adams explains it like: "Monty is saying, in effect: you can keep your one door — or you can have the other two." The 2/3 chance of winning doesn't disappear — it just collapses onto the remaining unopened door.

And Stanford professor Keith Devlin says: "Your original door still has a 1 in 3 chance. But by opening a goat door, Monty uses his knowledge to show you that the remaining closed door now carries the full 2 in 3 probability.

So why do so many people still get this wrong?

If you also got this wrong — don't worry.

You're not alone. A famous study showed that only 13% of people chose to switch. And even Nobel physicists have confidently insisted the problem was 50/50 — and criticized those who said otherwise.

Why? Because this isn't just about not knowing math or probability. It's also a cognitive illusion. According to the Wikipedia article on the Monty Hall problem, the typical behavior of not switching can be explained by several well-known psychological effects: We naturally:

- We tend to overvalue the option we already chose, simply because we feel it's "ours. In psychological literature it's called endowment effect
- Status Quo Bias: We feel more comfortable sticking with our initial decision, even when switching is the better choice
- Omission Bias: People often prefer to make mistakes by doing nothing (staying) rather than by taking action (switching), even if action leads to better results.

The funny thing? Pigeons do better than we do. When exposed to this problem repeatedly, pigeons learn to switch every time. They don't overthink it — they just go with what works.

And there you have it.

What started as a strange little game show puzzle turned out to be a deep dive into probability, psychology, and human intuition.

I wrote four C++ programs, simulated hundreds of thousands of games, and even played a few myself — and every time, the message was the same:

Switching wins.

So next time you're faced with a problem that feels "50/50"... maybe take a step back. Run an experiment. Simulate it. Write some code.

Because code don't lie — and neither did Monty.

Thanks for watching. If you want to try the simulations yourself, check out the GitHub link in the description. And if you liked this breakdown, hit like, subscribe, or drop a comment with your thoughts or questions.

Until next time — keep questioning, keep coding

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

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.

```
Enter the number of simulations to run for each strategy: 100000

=== Monty Hall Simulation Results ===

       Strategy          Wins         Losses      Win Percentage     Loss Percentage
-------------------------------------------------------------------------------------
         Switch         66324          33676              66.32%              33.68%
           Stay         33242          66758              33.24%              66.76%
PS C:\Users\Sumit Sah\Documents\Monte Hall Project\monte_3doors>
```

3-Door Simulation (Auto Mode)

This is the result of running 100,000 simulations with switching vs staying. Switching wins 66% of the time — just like the theory says.

General N-Door Game (10 Doors Example)

```
Enter number of doors (minimum 3): 10

   ==========================
      MONTY HALL - 10 DOORS
   ==========================

    [1]     [2]     [3]     [4]     [5]     [6]     [7]     [8]     [9]     [10]
   +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+
   | ??? | | ??? | | ??? | | ??? | | ??? | | ??? | | ??? | | ??? | | ??? | | ??? |
   +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+

   - One door hides a CAR
   - The others hide GOATS
   - Monty will reveal all goat doors except one

Pick a door (1 to 10): 3

Monty opens 8 goat doors:

   Doors:
   [1]     [2]     [3]     [4]     [5]     [6]     [7]     [8]     [9]     [10]
   +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+
   |GOAT | |GOAT | | ??? | |GOAT | |GOAT | |GOAT | | ??? | |GOAT | |GOAT | |GOAT |
   +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+
Do you want to switch your choice? (y/n): y

Final reveal:

   Doors:
   [1]     [2]     [3]     [4]     [5]     [6]     [7]     [8]     [9]     [10]
   +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+
   |GOAT | |GOAT | |GOAT | |GOAT | |GOAT | |GOAT | | CAR | |GOAT | |GOAT | |GOAT |
   +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+

:) You WON the car!

Do you want to play again? (y/n):
```

```
   Doors:
   [1]   [2]   [3]   [4]   [5]   [6]   [7]   [8]   [9]   [10]
   +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
   |GOAT | |GOAT | |GOAT | | CAR | |GOAT | |GOAT | |GOAT | |GOAT | |GOAT | |GOAT |
   +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+

:) You WON the car!

Do you want to play again? (y/n): y

Pick a door (1 to 10): 5

Monty opens 8 goat doors:

   Doors:
   [1]   [2]   [3]   [4]   [5]   [6]   [7]   [8]   [9]   [10]
   +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
   | ??? | |GOAT | |GOAT | |GOAT | | ??? | |GOAT | |GOAT | |GOAT | |GOAT | |GOAT |
   +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
Do you want to switch your choice? (y/n): n

Final reveal:

   Doors:
   [1]   [2]   [3]   [4]   [5]   [6]   [7]   [8]   [9]   [10]
   +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
   | CAR | |GOAT | |GOAT | |GOAT | |GOAT | |GOAT | |GOAT | |GOAT | |GOAT | |GOAT |
   +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+

:(You got a goat. The car was behind door 1.

Do you want to play again? (y/n): n

=== Game Summary ===
Total games played: 4
Switched and won: 2
Switched and lost: 0
Stayed and won: 1
Stayed and lost: 1
PS C:\Users\Sumit Sah\Documents\Monte Hall Project\monte ndoors>
```

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

10 Doors

```
Enter number of doors (minimum 3): 10
Enter number of simulations to run for each strategy: 100000

=== Monty Hall Simulation Results ===
Number of Doors: 10
Simulations per strategy: 100000

       Strategy          Wins        Losses      Win Percentage     Loss Percentage
-------------------------------------------------------------------------------------
         Switch         90119          9881              90.12%              9.88%
           Stay         10031         89969              10.03%             89.97%
PS C:\Users\Sumit Sah\Documents\Monte Hall Project\monte_ndoors>
```

A sample simulation with 10 doors and 100,000 rounds. As expected, switching wins nearly 90% of the time.

100 Doors

```
Enter number of doors (minimum 3): 100
Enter number of simulations to run for each strategy: 100000

=== Monty Hall Simulation Results ===
Number of Doors: 100
Simulations per strategy: 100000

       Strategy          Wins        Losses      Win Percentage     Loss Percentage
-------------------------------------------------------------------------------------
         Switch         99028           972              99.03%              0.97%
           Stay           980         99020               0.98%             99.02%
PS C:\Users\Sumit Sah\Documents\Monte Hall Project\monte_ndoors>
```

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!