Monty Hall Problem Task

This problem is extensively explained in many Internet Sources. My intention is not to expose a theoretical solution, but to explain the approach of my implementation.

The goal is clear : Create a program that runs a simulation of this problem a number of times and returns the chances of winning after Staying with the original choice and Switching to the remaining door after the presenter's hint.

Initial Approach

Having a class that encapsulates the Monty Hall Problem in it's different states. As many iterations would be run, it's a good idea to reuse the instance to avoid heap allocations or stack windings.

- Implement a RESET method that chooses the door where the car is.
- Implement a PICK method that sets the door that the player choose.
- Implement a RESULT method that returns if the player wins the prize after Stay or Switch.
- From the main loop execute N iterations and count the success of Staying and Switching the choices. Recap with a global averaging to get the percentage of both.

Challenge of Initial Approach

After the player chooses a door, the presenter reveals a door that's not a win.
The challenge was to determine what is the door to check if the player decides to Switch.

Solution

If the player picked the car's door, the switch door is a goat.
(Because the presenter would reveal a goat and the remaining is a goat)

If the player picked a goat's door, the switch door is the car.
(Because the presenter would reveal the remaining goat and the remaining is car)

Final Approach

There is no need to have a specific class to simulate this Problem. The whole business is reduced to generating 2 random numbers between 1 and 3. If the players choose to Stay, the numbers must be equal to win. If the player choose Switch, the numbers must be different to win.

Wrapping up

The main process obtains the number of iterations of the simulation from the program's arguments.
A loop generates these 2 random numbers and perform the count of the results. Staying and Switching using the Final Approach logic.

Disclaimer

I am aware that this problem could be modelled out with classes and states, even adding support to a Visual Controller, to make it interactive and with a visual aid. This would include UI update events, and other messages. But for the sake of this simulation, the priority was to make it as efficient as possible.

Also, I could have added another random generator, or even an entry point to change it. I could have used an interface or even the random number generator as a parameter of a template. Many choices there. It is important not to use the default randomizers and seed them properly before running the simulation. Not done here.