Monty Hall Problem Simulation

Ishita Bhatt

Introduction

The Monty Hall Problem

The Monty Hall problem is a famous probability puzzle, which was first seen on the American television game show ‘Let's Make a Deal’ and was named after its original host, Monty Hall.

The original problem statement is :

> *“Suppose you're on a game show, and you're given the choice of three doors: Behind one door is a car; behind the others, goats. You pick a door, say No. 1, and the host, who knows what's behind the doors, opens another door, say No. 3, which has a goat. He then says to you, "Do you want to pick door No. 2?" Is it to your advantage to switch your choice?”*

Theoretical Solution

Initially:

- Probability that the chosen door has the car: **P(Car) = 1/3**
- Probability that the chosen door has a goat: **P(Goat) = 2/3**

Case 1: If the player chooses to keep the choice:

If the player chooses to keep the choice, he wins if the initial choice was the car.
∴ Probability of winning= **1/3**

Case 2: If the player chooses to switch:

Case 2.1 : If the player initially chose the door with the goat (P(Goat) = 2/3), the other door that was opened would reveal the other goat. So, the closed door has the car.

Thus switching is better.

Case 2.2: If the player initially chose the door with the car(P(Car) = 1/3), the other door that was opened would reveal one of the goats. So, the closed door has the other goat.

Thus keeping the choice is better.

Result:

Therefore:

- If your initial pick was wrong, switching is better- P(Switching is better)=P(initial pick is goat)=**2/3**
- If your initial pick was right, switching is bad- P(Switching is bad)=P(initial pick is car)=⅓

∴ Probability that switching is better = 2/3 (≈ 66.666%)

Simulation Algorithm

- Programming Language: Python

Steps:

1. Setup: Create 3 doors(indices 0, 1, 2). Randomly assign the Prize to one door.
2. The player makes a choice: Player selects a door(D_player).
3. The player enters number of simulations to run(N)
4. The host(code) identifies the unchosen doors. From those, the host opens a door that does not contain the prize(D_host).
5. Identify the remaining closed door that is neither the player's choice nor the opened door (D_switch)
6. Determine Outcomes:
    - If (D_player == Prize), keeping the choice is better.
    - If (D_switch == Prize), switching is better.
7. Host asks the player if he would like to keep the choice or switch and prints the result.
8. Repeat: Run this loop N times (e.g., 100,000).
9. Calculate: Divide total wins for each strategy by N to get probabilities.

Code:

Python

```
import random
import sys

# define the simulation function
def run_monty_hall_simulation():

    print("------ Monty Hall Simulation ------")

    # taking user inputs and exception handling
    while True:
        try:
            N= int(input("Enter number of simulations to run (N): "))

            if N <= 0:
                raise ValueError("Number of simulations must be greater than
0.")

            player_choice = int(input("Select a door (0, 1, or 2): "))

            if player_choice not in [0, 1, 2]:
                raise ValueError("Door must be 0, 1, or 2.")

            break

        except ValueError as e:
```

```
            print(f"Error: {e}. Please try again.")


    # setup
    theoretical_rate = 2/3
    doors = [0, 1, 2]
    wins_stay = 0
    wins_switch = 0


    # Main simulation
    for i in range(N):
        # Randomly assign prize to one of the doors
        prize_door = random.choice(doors)


        # identify doors available for the host to open and chose one randomly
        available_for_host = [d for d in doors if d != player_choice and d !=
prize_door]
        host_opens = random.choice(available_for_host)


        # Identify the door that the player can switch to
        switch_choice = [d for d in doors if d != player_choice and d !=
host_opens][0]


        # Result of one simulation
        # If player decides to keep the choice
        if player_choice == prize_door:
            wins_stay += 1


        # If player decides to Switch the choice
        if switch_choice == prize_door:
```

```
            wins_switch += 1

        if i % 1000 == 0: progress_bar(i, N)

    # Final probabilities
    prob_stay = wins_stay / N
    prob_switch = wins_switch / N

    error_rate = abs(prob_switch - theoretical_rate) * 100

    # Display results
    print(f"""
        {'='*30}
        SIMULATION RESULTS FOR (N={N} SIMULATIONS)
        {'='*30}
        Player Choice: Door {player_choice}
        {'='*30}
        Host opens: Door {host_opens}
        Prize door: Door {prize_door}
        {'-'*30}
        Strategy: KEEP   | Wins: {wins_stay:<5} | {wins_stay/N:.2%}
        Strategy: SWITCH | Wins: {wins_switch:<5} | {wins_switch/N:.2%}
        {'-'*30}
        Theoretical Win Rate (Switch): {theoretical_rate:.2%}
        Actual Win Rate (Switch):      {prob_switch:.2%}
        ERROR RATE (Difference):       {error_rate:.4f}%
        {'='*30}
    """)
```

```
    # Conclusion
    if prob_switch > prob_stay:
        print("CONCLUSION: Switching is better.")
    else:
        print("CONCLUSION: Keeping the choice is better.")

# progress bar function
def progress_bar(current, total, bar_length=20):
    percent = float(current) * 100 / total
    arrow = '-' * int(percent/100 * bar_length - 1) + '>'
    spaces = ' ' * (bar_length - len(arrow))

    sys.stdout.write(f"\rProgress: [{arrow+spaces}] {percent:.2f}%")
    sys.stdout.flush()

# Run the function
if __name__ == "__main__":
    run_monty_hall_simulation()
```

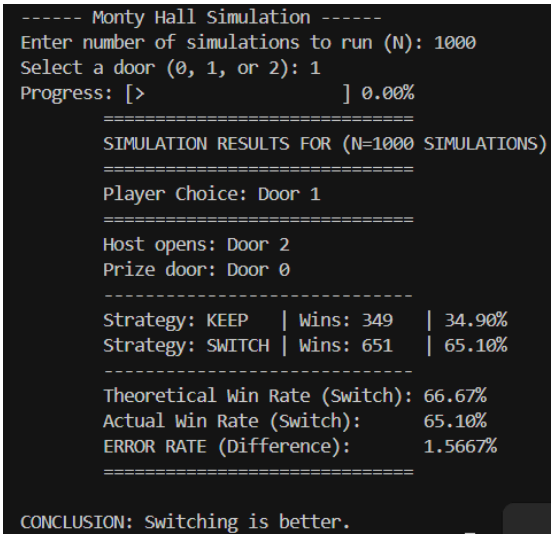
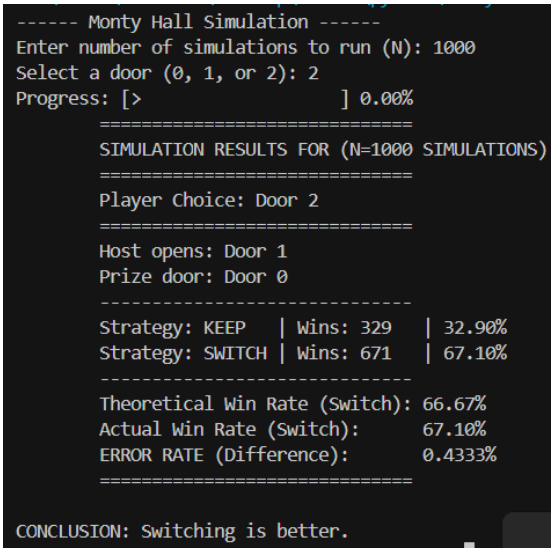
Code Explanation:

- `run_monty_hall_simulation()` function is called N times where N is the number of simulations
- `Random` library is used to generate random values
- User inputs: the program takes N(number of simulations to run) and
- Exception Handling to avoid the program from crashing: The chosen door must be 0, 1 or 2 and the number of simulations must be positive. If not, the program prints the message “Door must be 0, 1, or 2” or number should be positive respectively and asks

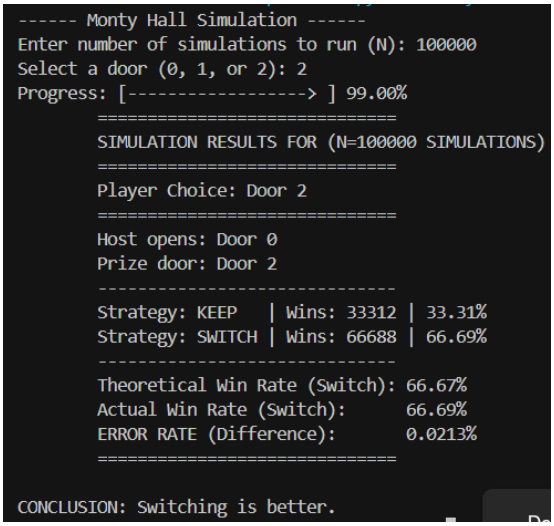
for a new value in case the entered value is outside the expected range. We use try block to avoid crashing in case the user input is a string instead of an integer

- Initially:
    - doors = [0, 1, 2]
    - wins_stay = 0
    - wins_switch = 0
- The program randomly assigns the prize to one of the doors
- Doors available for the host to chose: (Not chosen by the player AND Not the prize door)- we store these in a list `available_for_host`
- Host chooses one randomly out of the list
- Doors available for the player to switch to: (Not chosen by player AND not Opened by host) We store this door in a variable `switch_choices`
- The program calculates the result for this simulation and increment the number of wins by one
- The loop is run N times
- Finally, the probability is calculated by dividing the total number of wins of each choice by N
- Print final results

Theoretical vs Simulation Results:

N=1000 Player_choice= Door 1	
N=1000 Player_choice= Door 2	

N=100000
Player_choice= Door 2

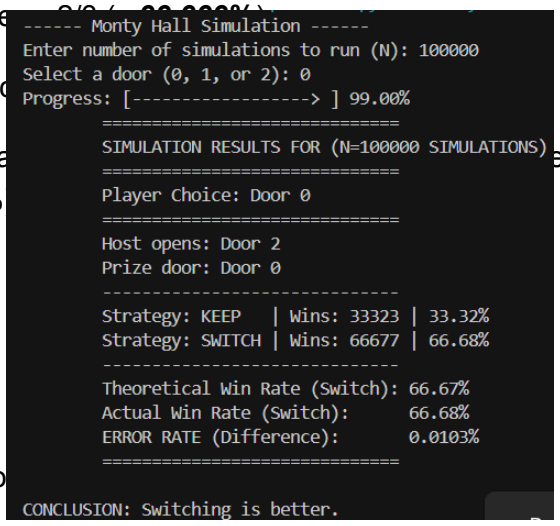


Theoretical: Probability that switching is bette

Simulation: The simulations give us a value

In general, the more the simulations(higher va esult closer to the theoretical value 2/3 (≈ 66.666%

N=100000
Player_choice= Door 0

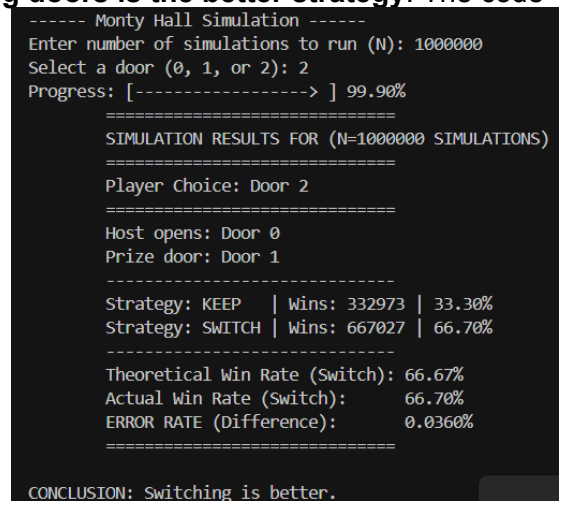


Conclusion

The simulation results are approximately equal to

This confirms the theoretical result that **switching doors is the better strategy**. The code successfully simulates the Monty Hall problem.

N=1000000
Player_choice=Door 2



Bibliography

- Problem statement: Wikipedia (en.wikipedia.org/wiki/Monty_Hall_problem)
- The original problem**:** Selvin, S**.** (1975). A problem in probability (Letter to the editor). *The American Statistician*, 29(1), 67.
- Sys library: https://docs.python.org/3/library/sys.html