Project Report on the Computational Analysis of the Monty Hall Problem: An In-depth Examination of Probability and Decision Strategy

Yufei Huang | Master's in Computer Science | Northeastern University

Abstract:

This project report delineates the comprehensive analysis of the Monty Hall problem, a renowned probability puzzle, employing Python for simulations and algorithmic strategies. It encapsulates the development of interactive games and analytical models to unravel the puzzle's complexities. By integrating core computer science algorithms and principles, the report presents a novel perspective on this classic problem, contributing to the field of computational probability and decision-making.

1. Introduction:

The Monty Hall problem, stemming from a game show scenario, presents a unique challenge in probability theory and decision-making. This project aims to apply advanced computer science techniques, particularly in Python programming, to analyze and simulate the problem. The objective is to provide empirical evidence supporting the theoretical probability models and to create an interactive understanding of the problem's dynamics.

2. Methodology:

2.1 Bayesian Analysis:

2.1.1. Setting Up the Problem:
- Hypothesis (H): The car is behind door 1.
- Evidence (E): Monty opens a door revealing a goat.

2.1.2. Bayesian Formula Application:

$$P[H|E] = \frac{P[E|H] \cdot P[H]}{P[E]} = \frac{P[E|H] \cdot P[H]}{P[E|H] \cdot P[H] + P[E|not\ H] \cdot P[}$$

- The H ---- the chosen door has a car behind it
- not H ---- the chosen door has a goat behind it.
- E ---- Monty has revealed a door with a goat behind it

- P(H) = 1/3
- P(not H) = 1−P(H) = 2/3
- P(E|H) = 1
- P(E|not H) = 1

$$P(H|E) = \frac{1 \times \frac{1}{3}}{1 \times \frac{1}{3} + 1 \times \frac{2}{3}} = \frac{1}{3}$$

Here, (P(H|E)) is the probability that the car is behind door 1 given that Monty has shown a goat.

2.1.3. Determining Probabilities:

- P(H) : The prior probability that door 1 has the car, which is 1/3 as there are three doors.
- P(E|H) : The probability is 1 that Monty shows a goat given the car is behind door 1. Since Monty will always show a goat.
- P(E|notH) : The probability is 1 that Monty shows a goat given the car is not behind door 1.

2.2 Python Simulation:

In addition to the React development, we have implemented Python code to simulate the Monty Hall problem, providing statistical analysis and visualizations. This section of the report explains the Python code used to simulate and analyze the Monty Hall problem.

2.2.1 Initialization and Setup:

- Importing Libraries: We import necessary Python libraries such as NumPy, Pandas, Random, and Matplotlib for data manipulation, random number generation, and data visualisation.
- `doors_array` Function: This function initializes the doors. It randomly assigns a car (marked as 1) behind one of the three doors (marked as 0 if no car).
- `win_loose` Function: Determines the outcome of the game based on the player's strategy (change or unchange). It checks whether the player wins (collects 1 point) or loses (collects 0 points) based on their final choice.

2.2.2 Simulation and Results Collection:

- `total_results` Function: Runs the Monty Hall game multiple times and collects the results. It records the door configurations, initial choices, host's choices, final decisions, and game outcomes.

- Dataframes Creation: Using Pandas, we create dataframes to store and display the outcomes of games where the player either changed or did not change their initial door choice.

2.2.3 Visualization and Statistical Analysis:
- Plotting Results: We use Matplotlib to plot the outcomes of each strategy (change or unchange) over multiple iterations, providing a visual representation of the probability of winning.
- `average_win` Function: Calculates the average probability of winning the game over multiple iterations for both strategies.
- **Comparative Analysis:** The plots compare the effectiveness of changing versus not changing the door, visually demonstrating the higher probability of winning when changing the door.

2.2.3. Results:
The Python simulations revealed a consistent pattern: switching doors resulted in a success rate of approximately 67%, while sticking to the original choice yielded a success rate of about 33%. These findings align with the theoretical probability models and underscore the counterintuitive nature of the problem.

2.3 React Game Implementation:

In addition to our Python-based simulations, we have expanded our analysis of the Monty Hall problem by developing an interactive game using React. This web-based application allows users to directly engage with the Monty Hall scenario, further enhancing the understanding of the problem's dynamics.
- Game Setup: The game initializes with three doors, one of which randomly conceals a car while the others hide goats. This setup is achieved using the React state management system.
- User Interaction: Players select a door, after which one of the remaining doors with a goat is revealed. The player then has the option to either stick with their original choice or switch to the other unopened door.
- Game Stages: The game progresses through stages, from initial selection to the final reveal, utilizing React's state updates to manage these transitions.
- Game Logic: The core logic of the game reflects the Monty Hall problem's probabilistic nature. It includes functions to handle door selection, reveal a non-prize door, and determine the game's outcome based on the player's final choice.

2.3.1 React Component Structure:
- The `ThreeDoorsGame` component encapsulates the entire game logic. It uses React hooks such as `useState` and `useEffect` for state management and lifecycle methods.
- State Variables:
  - `doors`: Represents the doors in the game.
  - `carBehindDoor`: Randomly sets which door has the car.
  - `chosenDoorIndex`: Tracks the player's chosen door.
  - `revealedDoorIndex`: Indicates the door revealed by the host.
  - `gameStage`: Manages the different stages of the game.
  - `gameOver`: Flags the end of the game.

- Event Handlers:
  - `handleDoorClick`: Manages the player's door selection and the game's progression.
  - `revealGoatDoor`: Reveals a door with a goat, excluding the chosen and winning doors.
  - `checkWin`: Determines if the player wins based on their final choice.

2.3.2 User Experience:
- The game provides a simple and interactive interface, allowing players to click on doors and make decisions, closely mimicking the real-life game show scenario.
- Alerts are used to inform the player of the game's outcome, whether they have won the car or found a goat.
- A 'Restart' button allows players to reset the game and try different strategies.

3. Conclusion and Future Enhancements:

Our team reflects on the enriching mathematical and probabilistic knowledge gained through this project. The Monty Hall problem exploration has yielded valuable insights into decision-making processes and the practical application of probability theory in real-life scenarios. We envision this report as a cornerstone for subsequent academic pursuits and projects, contributing substantially to our scholarly and professional development.

Future endeavors include expanding the algorithmic complexity to encompass more intricate scenarios and integrating machine learning models to predict outcomes based on different decision-making patterns.

- User Data Analysis: Collecting data from player choices in the React game could provide insights into common decision-making patterns and strategies.
- Advanced Game Features: Implementing additional features such as a score tracker, various difficulty levels, and enhanced user interfaces can further enrich the game experience.

In summary, this report offers a comprehensive examination of the Monty Hall problem, integrating mathematical theories, Bayesian analysis, computational simulations, and game theory principles. Our findings extend beyond resolving the central query, contributing to a broader comprehension of decision-making and probability. Despite inherent limitations, our study establishes a strong foundation for future research in related fields.

Appendix:

Appendix A Python Scripts

In [1]:
```python
import numpy as np
import pandas as pd
import random
import matplotlib as plt
import matplotlib.pyplot as plt
```

In [2]:
```python
# normal question
```

In [11]:
```python
def doors_array():
    '''
    the function is desigend for initialize the door list
    And if the gift behind the door, we mark it as 1, otherwise, it is 0
    '''
    doors = [0,0,0]
    gift_i = random.randint(0,2)
    doors[gift_i] = 1
    return doors,gift_i

def win_loose(doors_array,change):
    '''
    the function is desigend for check if the guest win the game based on two stratgies

    change: the string the represented change a door or not
        change: switch to another unopend door
        unchange: stick in the initial choice

    Output:
        win_change_nums: if the guest win the game, we collect 1 point, otherwise, collect 0 point
        choose_i: the initial chooice
        gift_i: the door which hide the gift
        change_choose_i: final decision of the guest
        host_i: the door opened by the host
    '''
    #users choose a door
    choose_i = random.randint(0,2)
    doors = doors_array()[0]
    gift_i = doors_array()[1]
    # host open the door
    host_i = -1
    for i in range(len(doors)):
        if doors[i] != 1 and i != choose_i:
            host_i = i
            break
    # user1 -- unchange
    if change == 'unchange':
        win_unchange_nums = 0
        if doors[choose_i] == 1:
            win_unchange_nums +=1
        change_choose_i = choose_i
        return win_unchange_nums,choose_i,gift_i,change_choose_i,host_i

    if change == 'change':
        win_change_nums = 0
        for i in range(len(doors)):
            if i != host_i and i != choose_i:
                change_choose_i = i
        if doors[change_choose_i] == 1:
            win_change_nums+=1
        return win_change_nums,choose_i,gift_i,change_choose_i,host_i
```

In [12]:
```python
# get the results
def total_results(times,change_or_not,doors_array,win_loose):
    '''
    the function is designed for help us to run the model multiple times and collect results

    Args:
    times: how many times we play
    change_or_not: the string that represented change the door or not
    doors_array: the function that represented initilize the door list
    win_loose: the function that generate the result of each time

    Outputs:
    total_door: the list represented the three doors
    win_or_lose: the result of playing the game
    open_door: the user's initial choice
    gift_door: the door that hide the gift
    host_door: the door opened by the host
    final_open: the user's final decision
    prob_win: the probability of winning the game

    '''
    times_num = times

    # Behind Door
    total_door = []

    # if the car(1) behind the door, get 1 point; if the sheep(0) behind the door, get 0 point
    win_or_lose = []

    # initial choice of the door
```

```
    open_door = []

    # the gift behind this door
    gift_door = []

    # the door opened by the host
    host_door = []

    # final decision of the user
    final_open = []
    for i in range(times):
        total_door.append(doors_array()[0])
        doors_result = win_loose(doors_array,change_or_not)
        win_or_lose.append(doors_result[0])
        open_door.append(doors_result[1])
        gift_door.append(doors_result[2])
        final_open.append(doors_result[3])
        host_door.append(doors_result[4])

    #the probability of win the game (the car behind the door)
    prob_win = sum(win_or_lose)/times

    return total_door,win_or_lose,open_door,gift_door,host_door,final_open, prob_win
```

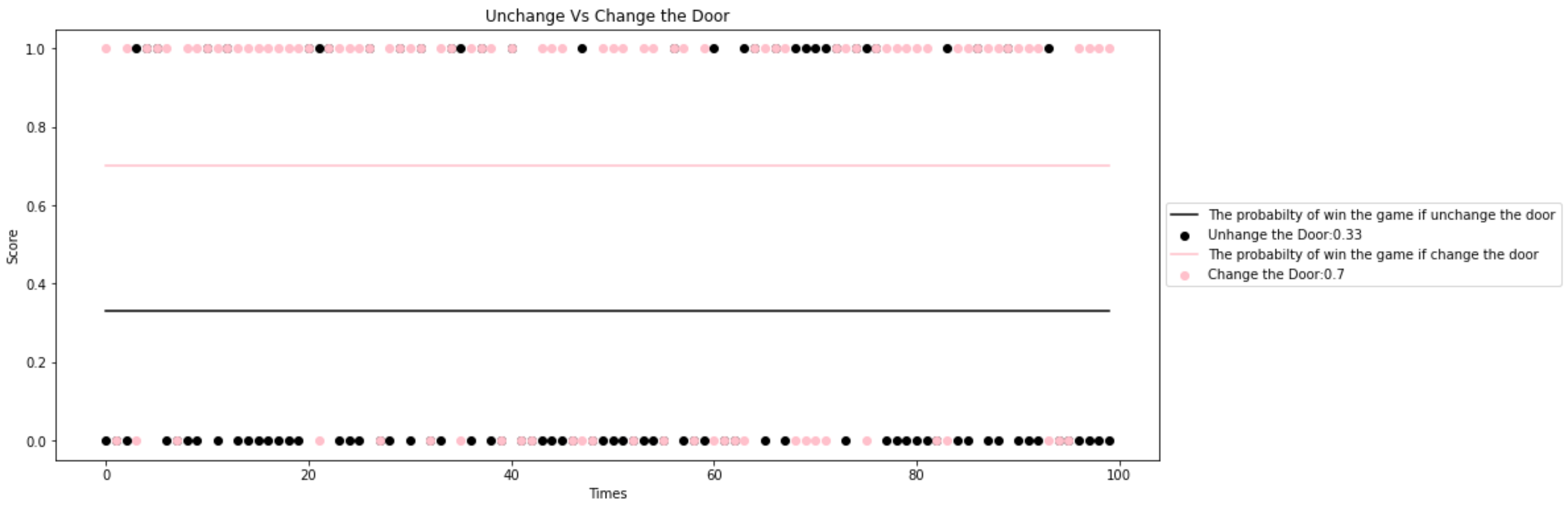
In [ ]:

In [13]:
```
#generate a table to visualize results
times = 100
total_unchange_result = total_results(times,'unchange',doors_array,win_loose)
unchange_result = pd.DataFrame({'Behind Door':total_unchange_result[0],
                        'Unchange: Initial Choice':total_unchange_result[2],
                        'Unchange: Host Choice':total_unchange_result[4],
                        'Unchange: Final Decision': total_unchange_result[5],
                        'Unchange: Gift': total_unchange_result[3],
                        'Unchange: Win or Lose':total_unchange_result[1]})
prob_win_unchange = total_unchange_result[-1]

total_change_result = total_results(times,'change',doors_array,win_loose)
change_result = pd.DataFrame({'Behind Door':total_change_result[0],
                        'Change: Initial Choice':total_change_result[2],
                        'Change: Host Choice':total_change_result[4],
                        'Change: Final Decision': total_change_result[5],
                        'Change: Gift': total_change_result[3],
                        'Change: Win or Lose':total_change_result[1]})
prob_win_change = total_change_result[-1]
```

In [19]:
```
# plot the results for one sub sample
plt.figure(figsize=(15,6))
plt.plot([prob_win_unchange]*times,color = 'black',label = 'The probability of win the game if unchange the door')
plt.scatter(range(times),unchange_result['Unchange: Win or Lose'],color = 'black',label = 'Unhange the Door:{}'.format
plt.plot([prob_win_change]*times,color = 'pink',label = 'The probability of win the game if change the door')
plt.scatter(range(times),change_result['Change: Win or Lose'],color = 'pink',label = 'Change the Door:{}'.format(prob_
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.title('Unchange Vs Change the Door')
plt.xlabel('Times')
plt.ylabel('Score')
```

Out[19]:
```
Text(0, 0.5, 'Score')
```



In [17]:
```
unchange_result
```

Out[17]:

	Behind Door	Unchange: Initial Choice	Unchange: Host Choice	Unchange: Final Decision	Unchange: Gift	Unchange: Win or Lose
0	[0, 1, 0]	0	1	0	0	0
1	[0, 0, 1]	2	0	2	2	0
2	[0, 1, 0]	1	2	1	0	0
3	[0, 1, 0]	0	1	0	0	1
4	[1, 0, 0]	2	0	2	1	1
...	...	...	...	...	...	...
95	[1, 0, 0]	2	0	2	2	0
96	[0, 1, 0]	0	1	0	0	0
97	[1, 0, 0]	1	2	1	2	0
98	[1, 0, 0]	0	2	0	0	0
99	[1, 0, 0]	1	0	1	1	0

100 rows × 6 columns

In [18]:

```
change_result
```

Out[18]:

	Behind Door	Change: Initial Choice	Change: Host Choice	Change: Final Decision	Change: Gift	Change: Win or Lose
0	[0, 1, 0]	0	1	2	1	1
1	[1, 0, 0]	0	1	2	0	0
2	[0, 0, 1]	2	0	1	2	1
3	[0, 0, 1]	1	0	2	1	0
4	[0, 1, 0]	0	1	2	2	1
...	...	...	...	...	...	...
95	[0, 0, 1]	2	0	1	1	0
96	[0, 1, 0]	2	0	1	0	1
97	[0, 0, 1]	0	2	1	1	1
98	[1, 0, 0]	1	2	0	2	1
99	[0, 0, 1]	1	0	2	2	1

100 rows × 6 columns

In [75]:

```
# statisical analysis
```

In [76]:

```
def average_win(total_results,times,change_or_not,doors_array,win_loose):
    '''
    the function is help us to calculate the average probability of winning the gift

    output:
    result: the list represented the probability of winning the game of each sub sample
    avg_prob: the average probability of winning the game
    '''
    result = []
    for i in range(times):
        total_result = total_results(times,change_or_not,doors_array,win_loose)
        result.append(total_result[-1])
    avg_prob = round(sum(result)/len(result),4)
    return result, avg_prob
```

In [77]:

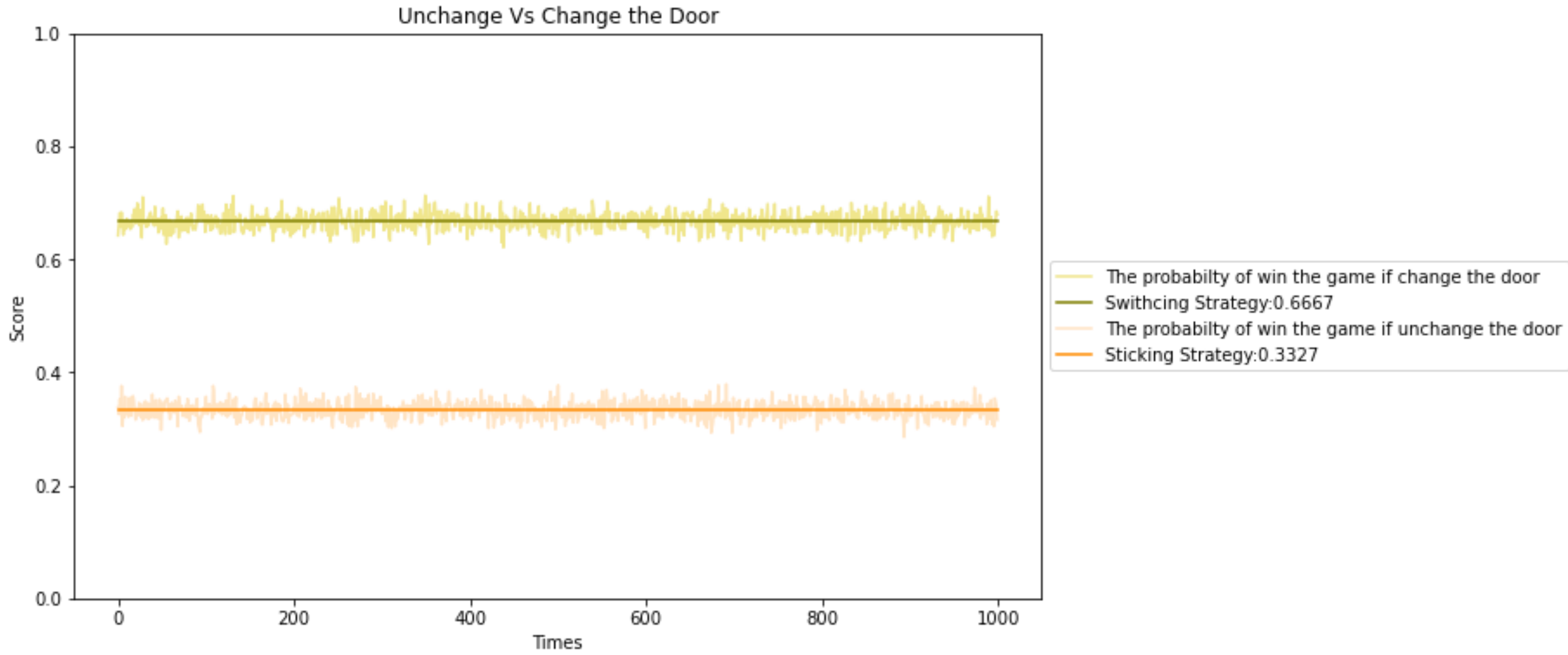
```
times = 1000
unchange_result = average_win(total_results,times,'unchange',doors_array,win_loose)[0]
unchange_avg_prob = average_win(total_results,times,'unchange',doors_array,win_loose)[1]
change_result = average_win(total_results,times,'change',doors_array,win_loose)[0]
change_avg_prob = average_win(total_results,times,'change',doors_array,win_loose)[1]
```

In [78]:

```
plt.figure(figsize=(10,6))
plt.plot(change_result,color = 'khaki',label = 'The probabilty of win the game if change the door')
plt.plot([change_avg_prob]*times,color = 'olive',label = 'Swithcing Strategy:{}'.format(change_avg_prob))
plt.plot(unchange_result,color = 'bisque',label = 'The probability of win the game if unchange the door')
plt.plot([unchange_avg_prob]*times,color = 'darkorange',label = 'Sticking Strategy:{}'.format(unchange_avg_prob))
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.title('Unchange Vs Change the Door')
plt.xlabel('Times')
plt.ylabel('Score')
plt.ylim(0, 1)
```

Out[78]:

```
(0.0, 1.0)
```


Unchange Vs Change the Door
1.0
0.8
0.6
0.4
0.2
0.0
Score
0
200
400
600
800
1000
Times
The probabilty of win the game if change the door
Swithcing Strategy:0.6667
The probabilty of win the game if unchange the door
Sticking Strategy:0.3327

In [ ]:

In [ ]:

Appendix B React Code

```
import React, { useEffect, useReducer, useState } from 'react'
import { Door } from './Door';
import goatImg from './goat_tr.png'
import Stack from '@mui/material/Stack';

import Button from '@mui/material/Button';
import carImg from './car_tr.png'
/**
 * Generates a random integer between 0 and n-1.
 * Note: Due to the implementation, if n is greater than 10, this function will not
 * be able to generate all numbers in the range [0, n-1].
 *
 * @param {number} n - The upper limit (exclusive) for the random number generation.
 *                     The function returns a number in the range [0, n-1].
 * @returns {number} A random integer between 0 and n-1.
 */
const getRandomNumber=(n)=>
{
   return ((Math.floor(Math.random() * 10))%n)
}



export const PlayingArea = ({resetGame,gameEnd,totalGame,winningGames,switchingWins}) => {

    let a=[0,1,2]

    const [doorState,changeState]=useState([0,0,0,0])
    const [winingState,changeWinningState]=useState(0)

    useEffect(()=>
    {
        changeWinningState(getRandomNumber(3))
    },[])

    let isSwitch=[]

    const openDoorHandler=(id)=>
    {

          if(!doorState[id])
          {
               isSwitch.push(id)
                 if(!doorState[3])
                 {
                      let x=doorState
                      if(winingState==id)
                      {
```

```
            let openDoorNumber=getRandomNumber(2)+1
                       x= x.map((element,index)=>
                           {
                               if(index!=id)
                               {
                                   openDoorNumber--;
                                   if(openDoorNumber==0)
                                   {
                                       return 1;
                                   }
                                   else
                                   {
                                       return 0;
                                   }
                               }
                               else
                               {
                                   return 0;
                               }
                           })
       }
       else
       {
           let openDoorNumber=getRandomNumber(2)+1
           x= x.map((element,index)=>
               {
                   if(index!=id&&index!=winingState)
                   {
                       return 1;
                   }      Ravi Chuahan, 12 months ago • inital phase completed
                   else
                   {
                       return 0;
                   }
               })
       }
       x[3]=1
        changeState([...x])
}
else if(doorState[3]==1)
{
     //   check win or not
      let x=doorState
      x[id]=1
      x[3]=2
      changeState([...x])
      if(id==winingState)
```