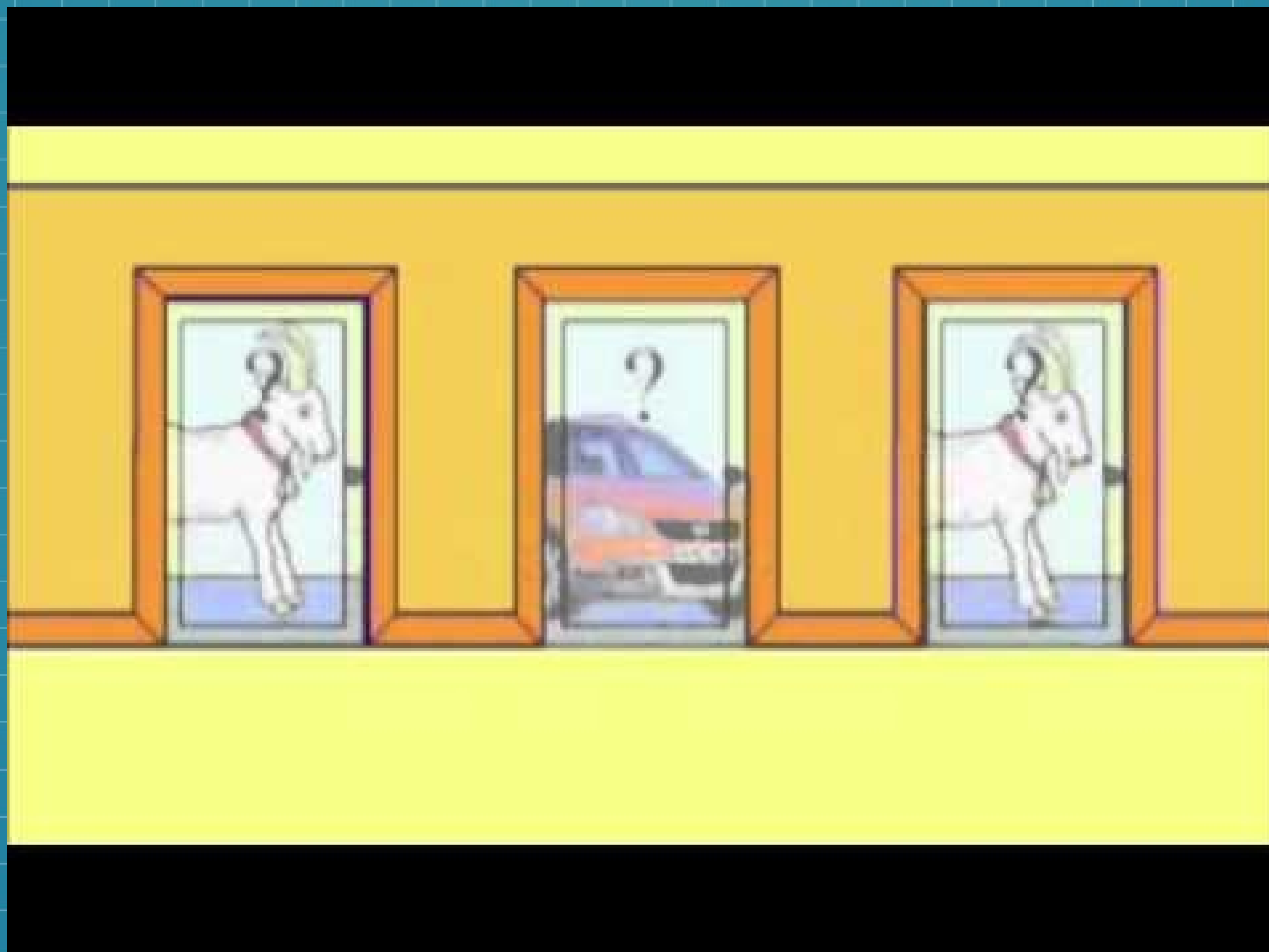
Le problème de Monty Hall

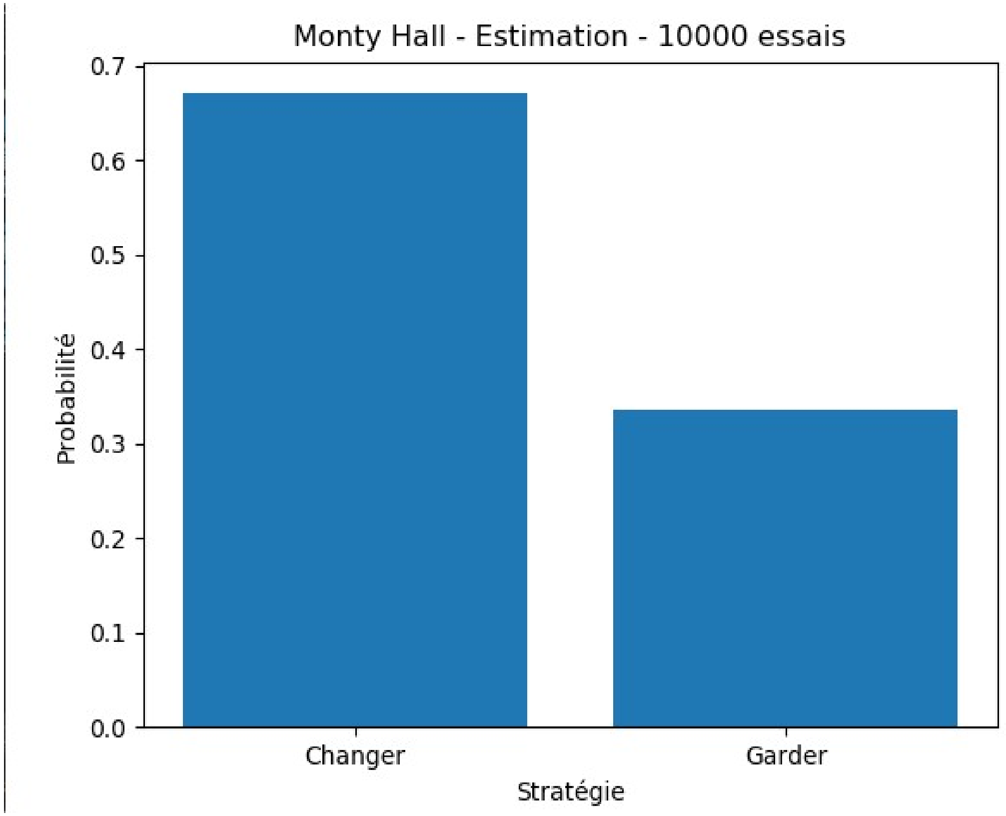


Présentation du problème.

En quoi consiste le problème ?

Le jeu oppose un présentateur à un candidat (le joueur). Ce joueur est placé devant trois portes fermées. Derrière l'une d'elles se trouve une voiture et derrière chacune des deux autres se trouve une chèvre. Il doit tout d'abord désigner une porte. Puis le présentateur doit ouvrir une st ni celle choisie par le elle cachant la voiture (le p ait quelle

Programme python.



Nous avons effectué des tests pour démontrer la solution du problème grâce à un programme python.

```
from enum import Enum
#Pour pouvoir rendre plus explicite la notion de choix nous utiliserons cette abstraction
#Nous définirons stratégie et dirons qu'il y en a deux possibles : changer et garder

from random import *
#L'aléatoire sera généré grace à random

import matplotlib.pyplot as plt


#définition des stratégies
class Strategie(Enum):
    GARDER = 0
    CHANGER = 1

# Simulation d'une seule partie
def partie_unitaire(strategie):

    seed()
    portes = [0,1,2] # Les portes seront représentées par une liste à 3 valeurs possibles
    porte_gagnante = randint(0,2) # La porte gagnante est générée aléatoirement entre 0 1 et 2 (correspondant
    # aux index du tableau de porte)
    choix_de_base_joueur = randint(0,2) # Le choix du joueur est généré aléatoirement entre 0 1 et 2 (correspondant
    # aux index du tableau de porte)

    #Partie logique :

    # Si la stratégie est de Garder
    # Une victoire correspond à ce que le choix du joueur soit celui la porte gagnante
    # Sinon c'est une défaite
    if (strategie == Strategie.GARDER):
        if (portes[porte_gagnante] == choix_de_base_joueur) :
            return True
        else:
            return False

    # Si la stratégie est de Changer
    # Une victoire correspond à ce que le choix du joueur ne pas soit celui la porte gagnante
    # Sinon c'est une défaite
    elif (strategie == Strategie.CHANGER):
        if (portes[porte_gagnante] != choix_de_base_joueur) :
            return True
        else:
            return False


    else:
        return 'Anomalie'

#fonction de répétition d'opérations
def jouer_plusieurs_parties(strategie, nombredeparties):

    liste_des_victoires = []

    for i in range(nombredeparties):
        if partie_unitaire(strategie) == True :
            liste_des_victoires.append(1)
        else :
            liste_des_victoires.append(0)

    return sum(liste_des_victoires)


def main(n):
    # Affichage des résultats via console
    print("Avec la stratégie : Garder, nous gagnons {}/{}"
          .format(jouer_plusieurs_parties(Strategie.GARDER, n), n))
    print("Avec la stratégie : Changer, nous gagnons {}/{}"
          .format(jouer_plusieurs_parties(Strategie.CHANGER, n), n))
    #Affichage des résultats via Matplotlib
    plt.figure()
    plt.bar([1,2],[jouer_plusieurs_parties(Strategie.CHANGER, n)/n,
                   jouer_plusieurs_parties(Strategie.GARDER, n)/n],
                   tick_label=["Changer","Garder"])
    plt.title("Monty Hall - Estimation - {} essais".format(n))
    plt.xlabel("Stratégie")
    plt.ylabel("Probabilité")
    plt.show()


## Lancement du jeu 10000 FOIS
main(10000)
```

Solution :

En changeant son choix le joueur a donc une probabilité de 2⁄3 × 1 = 2⁄3 de trouver la voiture. L'aide apportée par l'animateur est donc d'éliminer le mauvais choix (la chèvre) dans deux cas sur trois à condition bien sûr que le joueur change son choix initial. Il y a donc 1/3 de chances de gagner sans changer, 2/3 de chances de gagner en changeant.

Conclusion

Le problème de Monty Hall est un problème qui nécessite de la proportionnalité. En effet, comme démontrer avant on double avec une simple réflection nos chances d'avoir la voiture en changeant de porte. Ce problème est donc simple si il est réfléchis par la personne qui y joue. Il nous faudra donc changer de porte pour pouvoir gagner.