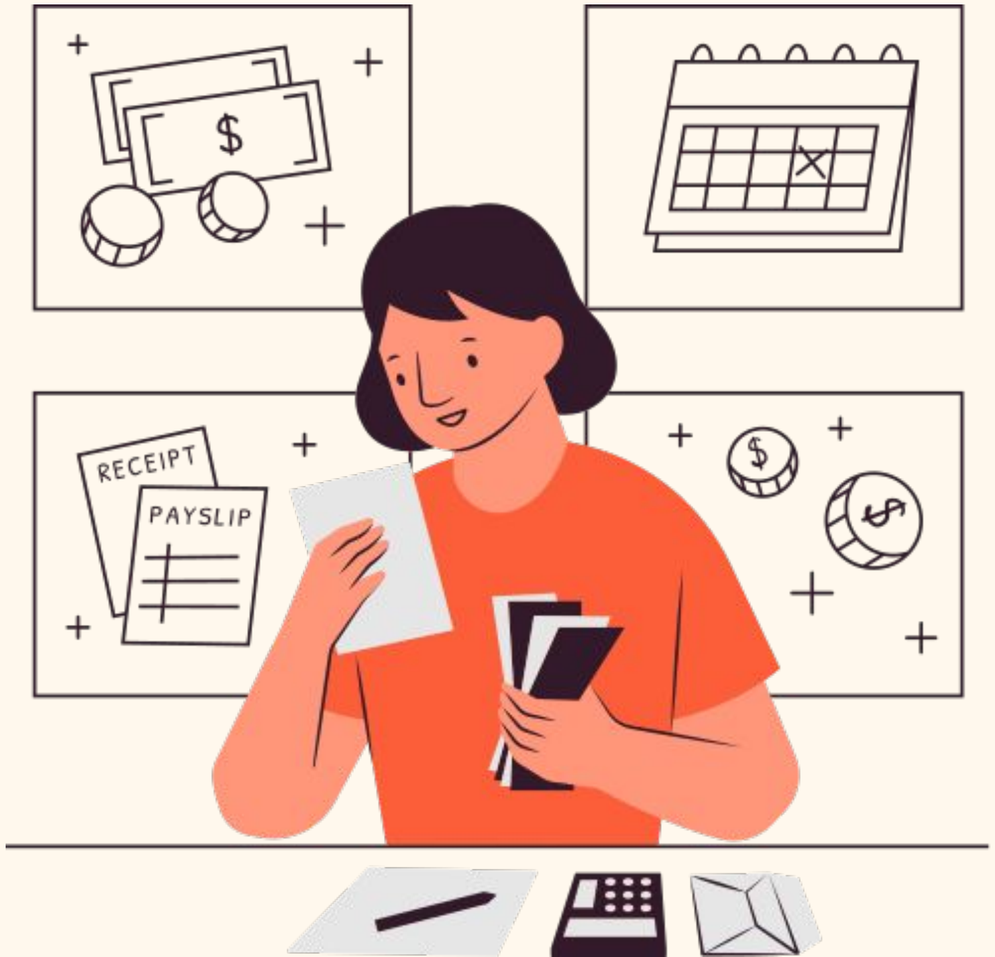
Term Project: Visualizing the Monty Hall Problem

Aziz Mutabanna

TABLE OF CONTENTS

The Idea

Take my knowledge of the Monty Hall problem's solution and use it to create a program that can visually show why the puzzle's seemingly irrational answer is the way it is, using a tree data structure.



01

The Monty Hall Problem

The task is to choose from one of three doors. Behind two of the doors are goats, and behind the one remaining door is a car. The goal is to try to choose the door with the car behind it.

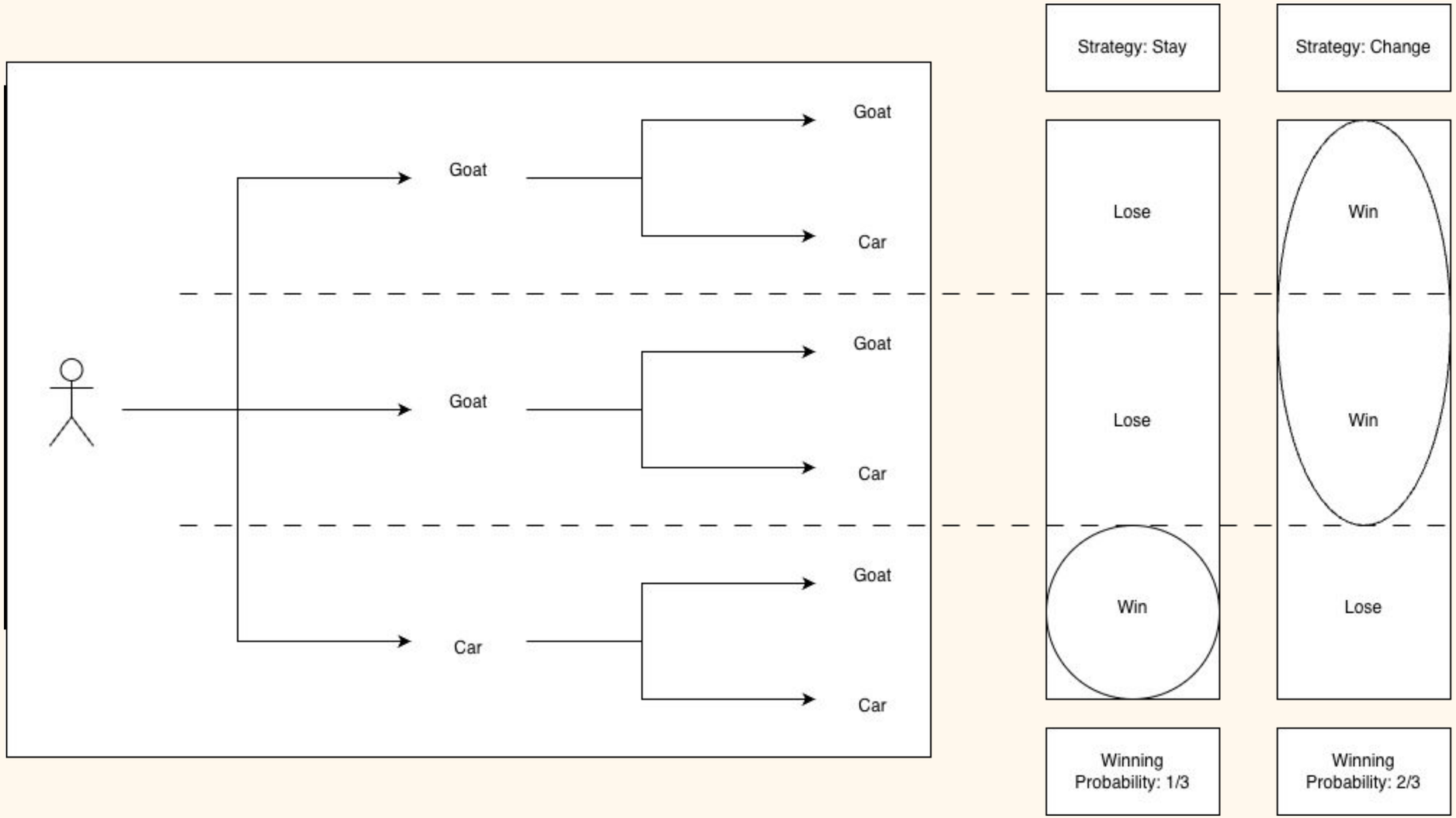
Once a door has been chosen, the game show host reveals what's behind one of the two doors remaining that wasn't picked—and he always only reveals a door with a goat behind it.

Now there are two doors left to choose from to find the one with the car behind it (since you know that the one the game show host opened has a goat behind it).

What should you do?

Should you stick with the door you originally guessed, switch your guess to the other unopened door, or does making a choice at this stage even really matter in terms of your chances of selecting the door with the car behind it?

The Concept

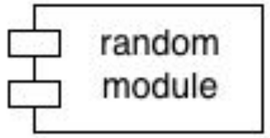

Strategy: Stay
Strategy: Change
Goat
Goat
Car
Goat
Goat
Car
Goat
Car
Car
Lose
Lose
Win
Win
Win
Lose
Winning Probability: 1/3
Winning Probability: 2/3

02

Modeling the Program

Main Workflow

main()	
objects	**functionality**
ktree	DecisionTree returned by montyhall_simulator() when "keep" strategy is passed to it; printed for visualization
k_win_rate	float returned by montyhall_simulator() when "keep" strategy is passed to it; added to win_rates as value
ctree	DecisionTree returned by montyhall_simulator() when "change" strategy is passed to it; printed for visualization
c_win_rate	float returned by montyhall_simulator() when "change" strategy is passed to it; added to win_rates as value
rtree	DecisionTree returned by montyhall_simulator() when "random" strategy is passed to it; printed for visualization
r_win_rate	float returned by montyhall_simulator() when "random" strategy is passed to it; added to win_rates as value
win_rates	dict with strategy names as keys and strategy win rates as values
best_strat	dict with "percent_wins", "strategy" as keys and highest strategy win rate, corresponding strategy name from win_rates as values



Classes

Door

+ num: int
  numerical label
+ behind: str
  prize of goat or car

+ print_door():
  prints label and prize
+ open_door():
  returns prize

Doors

+ door1: Door
  first door
+ door2: Door
  second door
+ door3: Door
  third door
+ door_list: lst
  list of all three doors

+ set_up_doors():
  puts prizes behind doors and adds doors to list
+ remove_door(Door):
  removes specified door from list

DecisionNode

+ count: int
  num of times decision was made
+ next_goat: DecisionNode
  choosing goat next
+ next_car: DecisionNode
  choosing car next

+ decide():
  increments count by 1
+ print_node(int, str):
  recursively prints node and subtree

DecisionTree

+ root: DecisionNode

+ set_up_tree():
  sets up decision levels for 1st and 2nd guess
+ make_guesses(Door, Door):
  increments total decisions count and moves to first guess DecisionNode
+ make_guesses_recursive(DecisionNode, Door):
  recursively increments decision level counts
+ print_tree():
  prints decision tree

Functions

montyhall_gameplay(strategy)

objects	functionality
strategy	str passed to function that determines guess2 selection
doors	Doors object to play game
goat_door_list	list to track Door objects hiding goats
guess1	Door object randomly chosen from doors; returned
door_revealed	Door object hiding a goat that isn't guess1; removed from doors
guess2	Door object chosen from doors using strategy; returned along with "win" or "lose" str depending on if guess2 hiding goat or car

montyhall_simulator(strategy, n)

objects	functionality
strategy	str passed to function to pass to montyhall_gameplay()
n	int passed to function to specify number of times montyhall_gameplay() is run
my_tree	DecisionTree to track decision counts; returned to main
my_wins	int to track number of times game is won
first_guess	Door object returned by montyhall_gameplay() and used to populate first decision level of my_tree
second_guess	Door object returned by montyhall_gameplay() and used to populate second decision level of my_tree
result	str describing whether game was won or lost
my_wins_percent	int initialized through calculation of my_wins / n * 100; returned to main

03 Implementation

— It’s time to see and run the code!

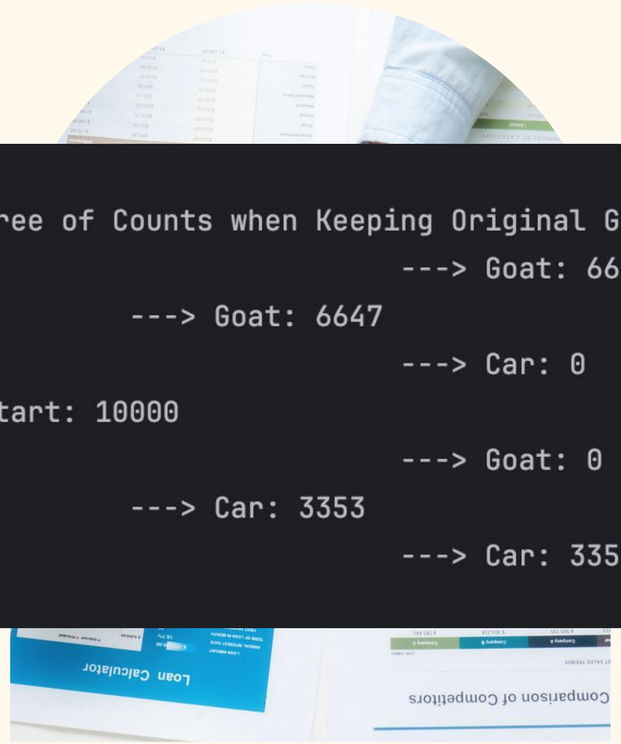
04 Results: Interpreting Output

1) Determining the Best Strategy

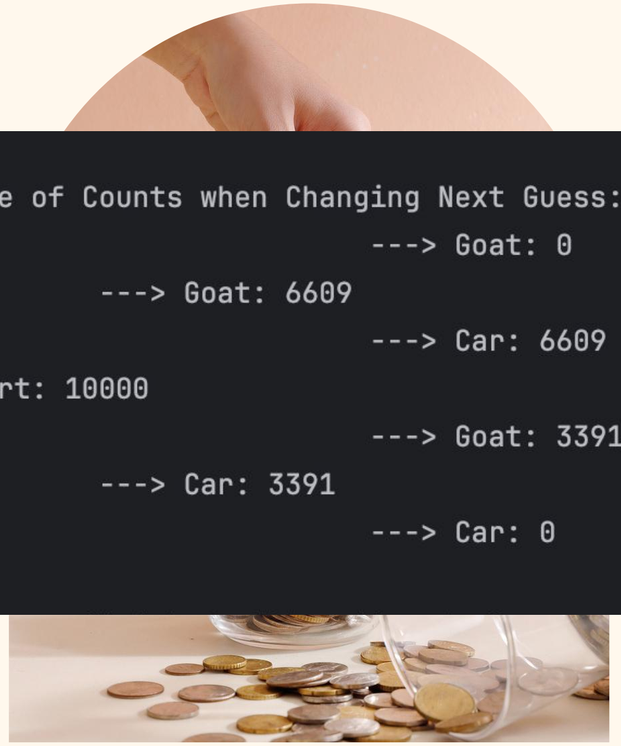
```
/usr/local/bin/python3.13 /Users/azizmutabanna/Documents/CS 313E/Term Project/testcase_beststrat.py

The best strategy is to change your guess due to its 66.10% win rate.

Process finfinished with exit code 0
```

2) Visualizing the Decisions being Made at Each Stage of the Game

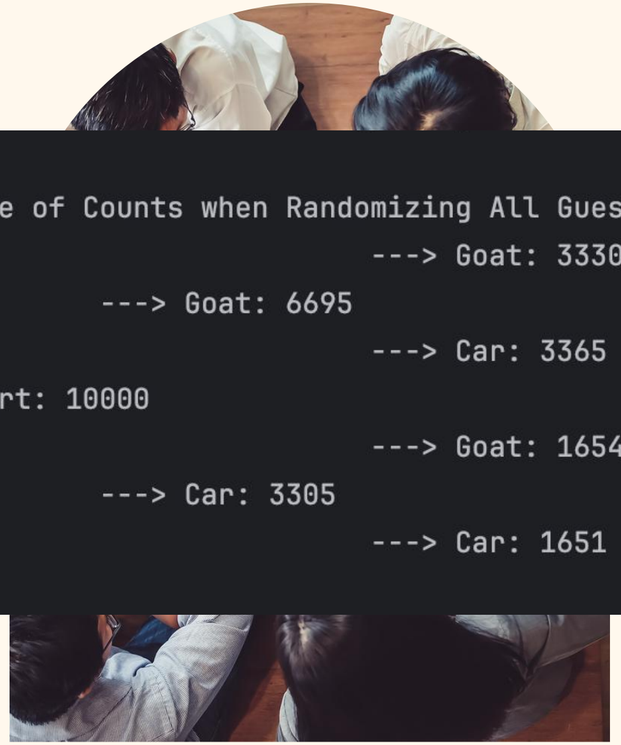
Tree Visualizations




```
Tree of Counts when Keeping Original Guess:
                        ---> Goat: 6647
        ---> Goat: 6647
                        ---> Car: 0
Start: 10000
                        ---> Goat: 0
        ---> Car: 3353
                        ---> Car: 3353
```



```
Tree of Counts when Changing Next Guess:
                        ---> Goat: 0
        ---> Goat: 6609
                        ---> Car: 6609
Start: 10000
                        ---> Goat: 3391
        ---> Car: 3391
                        ---> Car: 0
```



```
Tree of Counts when Randomizing All Guesses:
                        ---> Goat: 3330
        ---> Goat: 6695
                        ---> Car: 3365
Start: 10000
                        ---> Goat: 1654
        ---> Car: 3305
                        ---> Car: 1651
```

THANKS FOR WATCHING!