CS 2316

Individual Homework 5 – The Monty Hall Problem

Due: Wednesday February 12th, 2013
Out of 100 points

Files to submit: 1. HW5.py

This is an INDIVIDUAL assignment!

Collaboration at a reasonable level will not result in substantially similar code. Students may only collaborate with fellow students currently taking CS 2316, the TA's and the lecturer. Collaboration means talking through problems, assisting with debugging, explaining a concept, etc. You should not exchange code or write code for others.

For Help:
- TA Helpdesk – Schedule posted on class website.
- Email TA's or use T-Square Forums

-Don't forget to include the required comments and collaboration statement (as outlined on the course syllabus).
-Do not wait until the last minute to do this assignment in case
you run into problems.

Premise:
In this assignment, you will be tasked with simulating the "Monty Hall Problem". The scenario is such: You are on a game show and the host allows you to pick one of three doors. Behind one of the doors there is a car, and behind the others there is goat. After you make your choice, the game show host (knowing what is behind each door) opens one of the doors, which hides a goat. The host then gives you the option to remain with your original guess or switch to the only remaining door. Does it matter if you switch?

Watch the video here http://www.youtube.com/watch?v=Zr_xWfThjJ0 for an example.

We are going to answer this question by randomly simulating both situations a significant amount of times, while using a GUI.

You will be required to write the following methods in a class:

1. __init__()
2. importCSV()

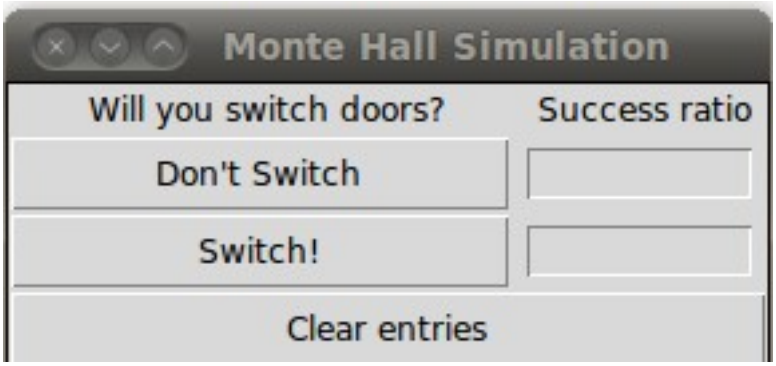
3. randomGuess()
4. stayWithIt()
5. changeToWin()
6. maxProb() ~ Bonus

File Format Information

This homework uses a CSV file containing 10,000 pseudo-randomly generated rows with three strings in each row. The strings represent what is behind door 1 through 3. There can only be a goat or a car.

Function Name: **__init__**

This method is automatically called whenever you create a new instance of your object. The __init__ method is responsible for arranging and initializing your GUI. You should create a GUI which looks like the following:



The “Don’t Switch” button should trigger the stayWithIt() method. The “Switch!” button should trigger the “changeToWin()” method. The “Clear entries” button should trigger a helper function that you will create in order to remove the contents of both entries.

From this __init__ method you will also call the “importCSV()” method and the “randomGuess()” method to fill the self.doorList and self.randGuessList instance variables.

Function Name: **importCSV**

Parameters:
    None
Return Value:
    None
Instance Variable:
    self.doorList – List containing each row of CSV file.

Description:

This method should read in the file “randomSetup.csv” by using the CSV module. It should then store the data in the instance variable “self.doorList”. You can either store the strings directly (e.g. ['Goat','Car','Goat']), or you can convert them into a numerical representation, such as [0,1,0].

Function Name: **randomGuess**

Parameters:
    None
Return Value:
    None
Instance Variables:
    self.randGuessList

Description:

This method will use Python’s random module to simulate 10,000 cases in which a contestant chooses one of the three doors (randomly each time). Instead of explicitly writing what is behind each door, you will use the number 1 to represent the door that hides the car, and 0’s for the other two. You will store the guesses in “self.randGuessList”. For example, a single “guess” would be sored in a sub-list of [0,1,0], which means that the user initially decided to pick the middle door. You will need to use a method from the random module to build the list of random guesses.

Function Name: **stayWithIt**

Parameters:
    None
Return Value:
    None
Instance Variables:
    self.randGuessList
    self.doorList
Description:

This method will compare the “user’s guesses” with the values in “self.doorList”. It should keep track of the amount of times the user guesses correctly, and then compute the ratio of successes. After it has been computed, it should be added to the GUI. For example, if the entry in the self.randGuessList was a [0,1,0], while the entry in the self.doorList was a ['Goat','Goat','Car'], they would NOT win, but if it was a ['Goat','Car','Goat'] (or [0,1,0 if you were using numbers) they WOULD win. Because the user is not switching their choice, you do not need to simulate what the gameshow host does.

Keep track of how many guesses were made, and how many cars were won. Divide these numbers to get the “success ratio” and place that number in the appropriate entry on the GUI.

Function Name: **changeToWin**

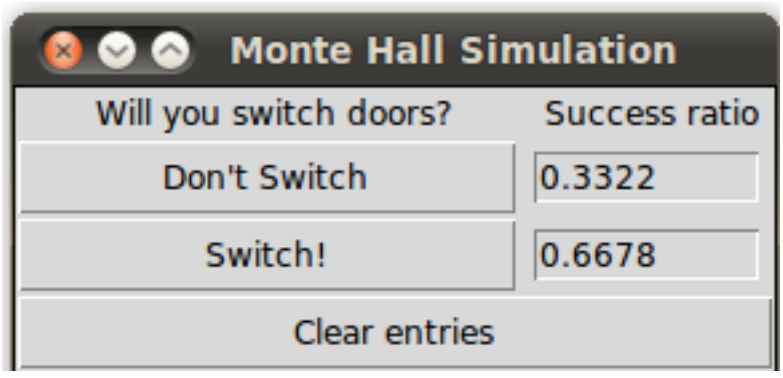
Parameters:
    None
Return Value:
    None
Instance Variables:
    self.randGuessList
    self.doorList

Description:

This method will simulate the case where the contestant decides to switch their original guess after the game show host opens one of the doors that hide a goat. Similar to the “stayWithIt()” method, it should compute the success ratio and put that number into the appropriate entry on the GUI.

In the changeToWin method, you need to simulate what the game show host does (which other door they open) to know which door the contestant changes their guess to. The gameshow host will never open the door that the contestant picked initially, and will never open the door that has a car behind it. In the case that the contestant initailly picked the door with the car behind it, the gameshow host will pick one of the other two doors randomly. In the changeToWin method, the contestant will ALWAYS switch their choice from the initial choice (in the self.randGuessList to the other (non-opened) door.

After pressing both buttons in the GUI you should get an output similar to the following:



Note: You will probably get a slightly different numerical answer every time a button is pressed.
Your grading TA should be able to clear the entries and press each button again without losing functionality. (e.g. your system should allow the “Don't Switch” and “Switch!” buttons to be pressed multiple times.)

Extra Credit Opportunity:

Function Name: **maxProb**

Parameters:
None
Return Value:
None

Description:

Now, what would happen if the contestant continuously chooses the same initial door? Find the initial door choice that maximizes the contestant’s chance of winning given the current data set. Once you find the door, you must add a label to the bottom of your GUI stating “The optimal door is: Door number ___”. You may assume that the contestant chooses the optimal strategy during the game (i.e. they switch doors after one of the other two is shown.).

Grading:

You will earn points as follows for each piece of functionality that works correctly according to the specifications

The GUI		**30**
The GUI has all required components	15	
All buttons and entries work properly	15	
importCSV()		**15**
Properly reads in csv file	10	
Makes data accessible to other methods	5	
randomGuess()		**15**
Properly generates random guesses	15	
stayWithIt()		**15**
Keeps track of the number of correct guesses	10	
Computes correct average and places it in GUI	5	
changeToWin()		**20**
Correctly identifies which door to switch to.	10	
Keeps track of the number of correct guesses	5	
Computes correct average and places it in GUI	5	
Miscellaneous		**5**
Code is commented and is written in proper style	5	
maxProb() ~ Bonus		**5**
Solved correctly		