Phys 499

Homework II

Due Date: October 16th, 2024

Q1 30 pts Monty Hall Problem

In class we discussed simulating a simple version of the Monty Hall Problem. In this exercise you will modify the program I gave you in the class to simulate the scenario in which the game host opens one of the doors, shows the gamer that there is nothing behind the door, and asks the game player whether he/she wants to change the door.

References:

"A problem in probability (letter to the editor)“ . *The American Statistician*. **29** (1): 67–71.

https://www-jstor-org.i.ezproxy.nypl.org/stable/2683689

and

https://en.wikipedia.org/wiki/Monty_Hall_problem#CITEREFSelvin1975a

Imagine a game with three doors labeled A, B, and C. There is a new car behind one of these doors and nothing behind the other two doors. You pick one of these doors and if the car is behind the one you picked, then the car is yours as your reward. The probability that you will pick the right door is of course 1/3.

Now let’s assume you pick door A. The host of the game show knows which door is the right one and opens one of the other two doors, say door C, and shows you that there is nothing behind it. At this time you know that the car is either behind door A you picked or behind door B. Now the host asks you whether you want to change your decision and pick door B. I will argue that changing your pick will increase your chances of winning the car, it will be 2/3. This may be unintuitive, but it is true.

- Simulate this full version of the game using Python and prove that the probability of winning becomes 2/3 if you change your decision. Note that your Python simulation should randomly choose the door with car behind it and not just set it to door A as in this example.

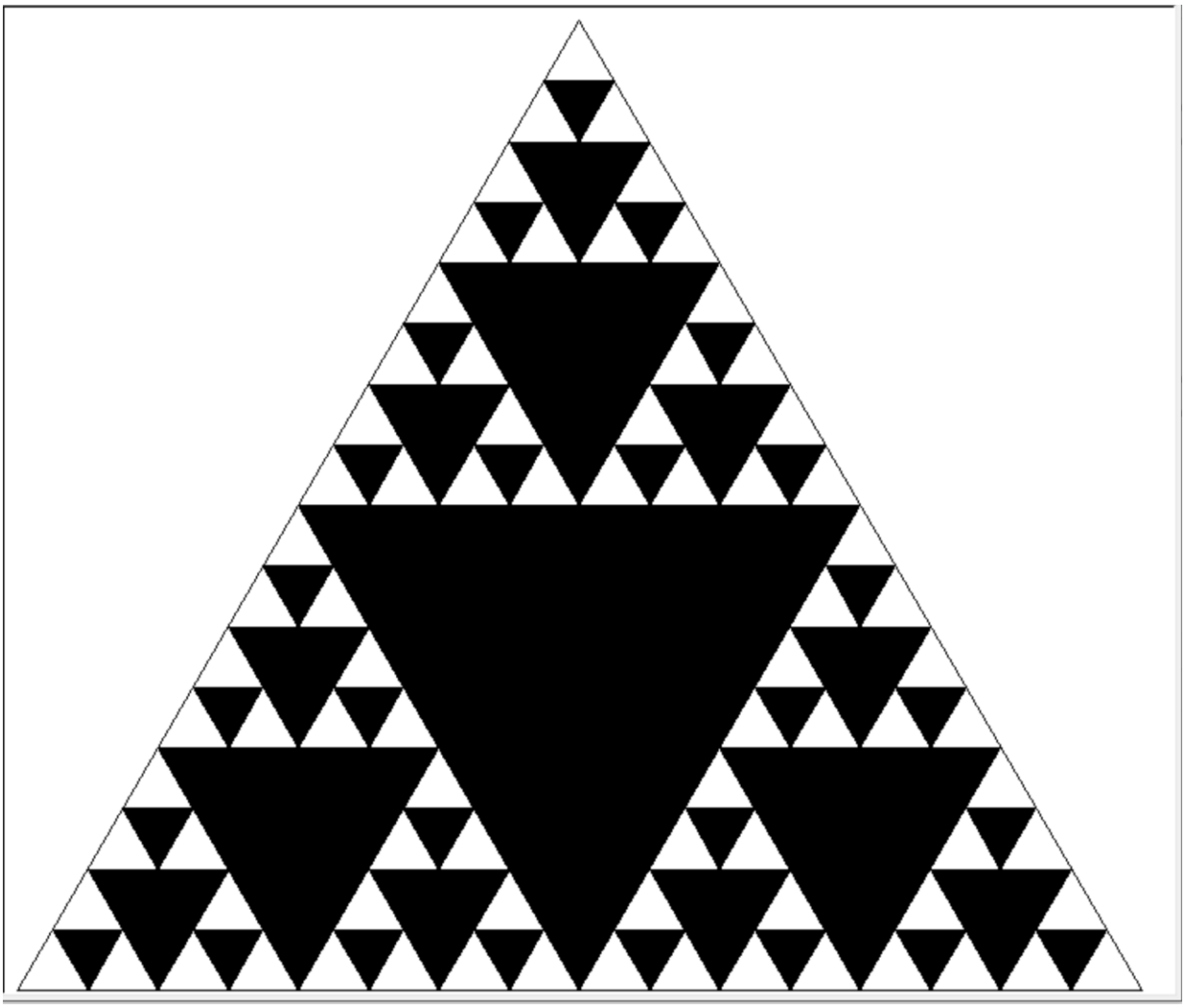
  Hint: You will need to write a function to simulate the host opening one of the doors that does not have the car behind. Then, the game player will change the decision to pick the remaining door. Also, see the basic version of the simulation program I shared in the class.

What I need from you

1. Your Python program that simulates the game
2. An output text file from the simulation of the program with N, the number of times game played, set to 10,000, 100,000, and 1,000,000

Q2 (30 pts)

Design and develop a Python program that uses a recursive function to plot a Sierpinski triangle, an example of which is shown below.



The *Sierpinski triangle* is a fractal pattern described by the Polish mathematician Wacław Sierpiński in 1915. However, it is known that it was in Italian art since the 13th century.

- Your task is to write a program sierpinski.py with a recursive function called sierpinski() that should draw one filled equilateral triangle (pointed downwards), and then, *call itself recursively 3 times (with an appropriate stopping condition) to generate the next set of Sierpinski triangles to the left, to the right, and above the current triangle* you just drew.
    - Look at the figures at the end of this document and think about what the parameters to draw these three set of triangles to the left, to the right and above the current one should be. You will need to work out some basic Geometry to find the x and y

coordinates and the side length of these triangles. Remember, the lower left corner's coordinates is (0,0) and the lower right corner's coordinates are (0,1).

- I provide two Python programs to help you with this work:
  - sierpinski_template.py: rename this sierpinski.py. It has the empty sierpinski() function that you will be working on, and also, has the main() function that does the following:
    - Prompts a user for the number of recursive iterations and reads it off
    - Initializes the Turtle graphics object
    - Sets the origin of the graphics panel to the lower left corner
    - Draws an unfilled equilateral triangle with side length of 1 pointing upwards
    - Makes the initial call to the recursive sierpinski() function to build the fractal pattern
  - drawpolygon.py: this is the Python software file that has the functions to draw unfilled or filled triangles given the coordinates of the vertices of a triangle.
- Your program will take *one* integer command-line argument *N*, to control the depth of the recursion. All of the drawing should fit snugly inside the equilateral triangle with endpoints (0, 0), (1, 0), and (1/2, √3/2).

Here is the signature of the recursive Python function you will need to develop:

```
#
# Draws one triangle with bottom vertex at coordinate (x, y)
# and side length of s.
#
# Then, recursively calls itself three times to generate
# the next set of Sierpinski triangles to the left, to
# the right, and above the current triangle you just drew.
def sierpinski(t, n, x, y, s):
```

The parameters are:

t: Turtle Graphics object for drawing
n: the number of recursive iterations
x, y: the coordinates of the vertex of the equilateral filled triangle with side length of s to be drawn pointing down.
s: The side length of the equilateral filled triangle to draw

Graphics Functions from drawpolygon.py:

```
# Draws a triangle with vertices (x0, y0), (x1, y1), and (x2, y2)
# t: Turtle Graphics object for drawing
# x: a vector of 3 numbers correspondging to the x coordinates of the vertices of the triangle to draw
# y: a vector of 3 numbers correspondging to the x coordinates of the vertices of the triangle to draw
def draw_triangle(t, x, y):
```

```
# Draws a filled triangle with vertices (x0, y0), (x1, y1), and (x2, y2)
# t: Turtle Graphics object for drawing
# x: a vector of 3 numbers correspondging to the x coordinates of the vertices of the triangle to draw
# y: a vector of 3 numbers correspondging to the x coordinates of the vertices of the triangle to draw
def draw_filled_triangle(t, x, y):
```