HW 3- The Odds of Successfully Navigating an Asteroid Field are Approximately 3,720 to 1!

ENGR 3H

Due 11:59 PM, Thursday, May 2

1 Never Tell Me the Odds (50 Points)

This week, we are going to use recursion to find a safe path through a treacherous asteroid field! Download from Gauchospace the files `mf.png`, `asteroids.mat`, and `animate_system.m`.

The file `asteroids.mat` contains (in order) the x- and y- coordinates for the centers of a series of circles and the radii for these circles. These circles will represent our asteroids. Our ship, the *Millennium Falcon*, may not look like much, but she's got it where it counts–but more importantly for our purposes, she'll be represented by a circle as well, with radius .5 and initially at the coordinates $(10, 0)$.

At each time step, the *Millennium Falcon* moves 1 space in the y-direction. It can move either 1 space left, 1 space right, or maintain the same x-coordinate. So, for example, on the first turn, the *Falcon* can go from $(10, 0)$ to $(9, 1)$, $(10, 1)$ or $(11, 1)$. For this assignment, we also require that our ship's x-coordinate stay between 0 and 20, that is, $0 \leq x_{MF} \leq 20$.

If the *Falcon* overlaps with an asteroid, it crashes. We want to avoid that! Two circles overlap if the distance between their centers is less than the sum of their radii. The distance between centers is given by the Pythagorean theorem:

$$d = \sqrt{(x_{MF} - x_{Ast})^2 + (y_{MF} - y_{Ast})^2}. \tag{1}$$

1.1 Making the Kessel Run in 12 Parsecs (30 Points)

To find our way through the asteroid field, we're going to write a recursive function called `find_path.m`. The function should take as an input the current position, the maximum depth of the tree, and the array of asteroid coordinates. It should return an array of coordinates for the ship's trajectory and the y-coordinate reached by the ship (for example, if it crashes at $y = 9$, it should return 8; if it makes it to the end, it should return the value of the maximum depth). The function should recursively find the a path that avoids asteroids up to the maximum depth. The process for this function should look as follows:

1. Check if, at the current coordinates, the ship has hit an asteroid. If it has, return the current coordinates and a value of $y_{MF} - 1$.
2. If the ship has not crashed, check if the maximum depth has been reached. If it has, return the current coordinates and a value of y_{MF}.
3. If neither of the first two conditions are met, find the path that results from moving left in the next step, moving right, and moving in neither direction. The function should return the value of whichever path had the greatest value (calculated recursively) and an array of the current coordinates followed by the coordinates of each step in the highest-value path.

Remember: if we are at one of the boundaries (0 or 20), our options are limited and we cannot go left or right, respectively.

1.2 It'll Take a Few Moments to Get the Coordinates from the Navicomputer (15 Points)

We want to get our ship to make it safely to $y = 20$. Write a script that implements your recursive function to find the best path, with a maximum depth of 20. Then, use that path to animate the system using `animate_system.m`, which takes as inputs the coordinates at each time step for the *Falcon* and the array of asteroid coordinates. Your script should be self-contained–the matrix containing the asteroid coordinates should be loaded within the script. Also, create a global counter variable in the script, initialized to zero. Run the script using run and time, and note how many times your recursive function is called. How does it compare to the maximum number of times it could be called given the maximum depth? Please note that this will take a little bit of time–on my laptop, it took just over 6 minutes. I recommend trying a smaller maximum depth first to verify everything is working as intended.

Hopefully you got through the asteroid field, but there's no way we'd have six minutes to calculate the correct coordinates! Let's modify our script now: at each time step, we compute the next 10 steps with the best value, rather than the full 20. That means we're going to compute the next 10 steps 20 times instead of the full 20 steps once.

As before, calculate the maximum number of possible paths and compare to the number of times your function was actually called according to run and time. As you can see, we significantly reduced the number of time it took to calculate the route. Be warned, we can't guarantee we find success with fewer paths considered!

1.3 The Weekly Challenge- I've Made a Lot of Special Modifications, Myself (5 Points)

The goal of this week's weekly challenge is to successfully reach a distance of 50 units starting from a different starting point with the fewest evaluations of your recursive function. You essentially have two variables you can adjust:

1. The depth `find_path.m` goes to;
2. How many steps the ship moves before calculating a new best path.

Your script should be written with these two variables able to be readily redefined (that is, they should be declared at the beginning of the script, and any time they're required later on, these variables are referenced). For the weekly challenge, submit your values for these two variables; the person who makes it to the end in the fewest calculations (or, if no one makes it, the person who made it the farthest in the fewest) wins!