TASK_3

Drone Simulation Report

Introduction

The goal of this project was to simulate the movement of multiple drones without colliding with each other. The simulation was done using Python, and visualized using Matplotlib. The simulation was extended to create a video of the drone movement using FFmpeg.

Methodology

Drone Class

A Drone class was created to represent each drone in the simulation. Each drone was initialized with a position, velocity, maximum speed, and radius. The update_position method updated the position of each drone based on its current velocity and a time step. The avoid_collisions method checked if the drone was too close to any other drone, and if so, updated its velocity to move it away from the other drone. The get_too_close method checked if a drone was within a safety distance of another drone, and if so, returned that drone. The separate method calculated the direction to move away from all nearby drones, and set the length of the resulting vector to the maximum speed of the drone. Finally, the set_magnitude method set the length of a given vector to a specified magnitude.

Simulation

The simulate_drones function was created to simulate the movement of multiple drones over a specified number of iterations. The function took as input the list of drones, the time step, the safety distance, and the maximum number of iterations. At each iteration, the positions of all drones were updated based on their current velocities, and the avoid_collisions method was called on each drone to ensure that they did not collide with each other. Finally, the positions of all drones were saved at each iteration.

ALGORITHM

1.Define a class Drone that represents a single drone. The Drone class should have the following attributes:

position: a 2D numpy array representing the current position of the drone

velocity: a 2D numpy array representing the current velocity of the drone

max_speed: a scalar representing the maximum speed of the drone

radius: a scalar representing the radius of the drone

2.Define a method update_position in the Drone class that takes a time step as input and updates the drone's position based on its current velocity.

3.Define a method get_too_close in the Drone class that takes a list of other drones and a safety distance as input and returns a list of drones that are too close to the current drone. A drone is considered too close if its distance to the current drone is less than the safety distance and it is not the current drone.

4.Define a method separate in the Drone class that takes a list of other drones as input and returns a separation vector that moves the drone away from the other drones. To calculate the separation vector, first calculate the vector from the current drone to each of the other drones. Then take the sum of these vectors and divide by the number of other drones to get the average direction away from the other drones. Finally, set the magnitude of the separation vector to be the maximum speed of the drone.

5.Define a method avoid_collisions in the Drone class that takes a list of other drones and a safety distance as input and updates the drone's velocity to avoid collisions with the other drones. To avoid collisions, first call the get_too_close method to get a list of drones that are too close to the current drone. If the list is not empty, call the separate method to get a separation vector and add a fraction of this vector to the current velocity to move the drone away from the other drones.

6.Define a function simulate_drones that takes a list of drones, a time step, a safety distance, and a maximum number of iterations as input and returns an array of positions for each drone at each iteration. To simulate the drones, loop through the specified number of iterations and:

Update the position of each drone using the update_position method

Call the avoid_collisions method on each drone to avoid collisions with the other drones

Append the current positions of all drones to the array of positions

7.Define a function create_random_drones that takes the number of drones, the maximum speed, the radius, and a position range as input and returns a list of randomly initialized drones. To create the drones, loop through the specified number of drones and:

Choose a random position within the position range

Choose a random velocity within the maximum speed

Initialize a new drone with the chosen position, velocity, radius, and maximum speed

Append the new drone to the list of drones

8.Create a list of drones using the create_random_drones function.

9.Simulate the drones using the simulate_drones function and save the resulting positions.

10.Use the matplotlib library to create a video of the drone simulation. Loop through the positions array and:

Clear the current plot

Plot the positions of all drones at the current iteration

Set the axis limits

Save the current plot as a frame of the video

Repeat for all iterations

Visualization

The positions of each drone at each iteration were plotted using Matplotlib. The resulting plot showed the movement of all drones over the specified number of iterations.

Video

The resulting plot was used to create a video of the drone movement using FFmpeg. The animation module from Matplotlib was used to create the animation, and FFmpeg was used to save the animation as a video. The resulting video was saved to disk.

Results

The simulation was run with five drones, a time step of 0.1, a safety distance of twice the radius of each drone, and a maximum number of iterations of 1000. The resulting simulation showed the movement of each drone without colliding with any other drone. The resulting video was one minute long and had a file size of 1 MB.

Conclusion

In conclusion, I have successfully simulated the movement of multiple drones without colliding with each other. The simulation was visualized using Matplotlib, and the resulting plot was used to create a video using FFmpeg. The resulting video was one minute long and had a file size of 1 MB. This simulation could be extended to include more drones, different drone types, and more complex movement patterns.

Git Hub link :

https://github.com/Gowtham-sys/TASK_3_Droame

Simulation video link:

https://drive.google.com/file/d/1M9nKkw84eyXCC3KN5huF1CmegS1ISBXL/view?usp=share_link

linkedin :

https://www.linkedin.com/in/gowtham-g-s-926355241/