# ASTEROID COLLISION SIMULATOR

## Introduction:

In the vast expanse of space, the movement and interaction of celestial bodies are governed by physics. Understanding these interactions is crucial for astronomers, astrophysicists, and aerospace engineers. One such interaction is the collision between asteroids. These events can have significant implications, both scientifically and practically. This project simulates the motion of asteroids in a 2D plane and detects potential collisions over a fixed period using velocity vectors and radius constraints. It also provides a dynamic 3D visualization of the simulation to better understand the motion and collision patterns.

## Objectives:

- Simulate the 2D motion of asteroids over time using velocity vectors.
- Detect and record collisions based on proximity and radius.
- Visualize the trajectories using an animated 3D plot.
- Present results in a structured output file.

## Methodology:

The simulation is based on Newtonian motion, where each asteroid moves in a straight line at a constant velocity. At each time step, the position of each asteroid is updated using the kinematic equations, considering the velocity in both the x and y directions. Then, for each pair of asteroids, the distance between them is calculated using the Euclidean distance formula. A collision is detected when the distance between two asteroids is less than the sum of their radii, which means the two asteroids have intersected or are in close proximity to each other.

## *Algorithm Overview-*

**1. Input Parsing:**

- **Objective:** Read asteroid data (ID, position, velocity, radius) from a file and store it as objects.
- **Steps:**
    1. Open and read the file line by line.
    2. Parse each line into asteroid properties and store them in an Asteroid class**.**

**2. Motion Simulation:**

- **Objective:** Simulate the asteroid positions over time.
- **Steps:**
    1. Initialize time step (dt) and max time (max_time).
    2. For each time step from 0 to max_time:
        1. Update each asteroid's position using its velocity (x(t) = x + vx*t and y(t) = y + vy*t).

**3. Collision Detection:**

- **Objective:** Detect collisions between asteroids based on their distances at each timestep.
- **Steps:**
    1. For each pair of asteroids at each timestep:
        1. Calculate the distance between them.
        2. If the distance is less than the sum of their radii, record the collision time.
    2. Store the collision data in a dictionary and sort it by time**.**

**4. Visualization:**

- **Objective:** Visualize the asteroid motion in 3D.
- **Steps:**
    1. Use Plotly to create a 3D plot of asteroid positions and animate them over time.
    2. Add background stars as 3D scatter points to enhance the visual effect.

**5. Output:**

- **Objective:** Output collision times and asteroid pairs to a file.
- **Steps:**

    Write the sorted collision data (time, asteroid IDs) to an output file**.**

# Pseudocode:

1. Read asteroid data from input file.
2. Initialize simulation parameters:
   - time_step (dt)
   - max_time
3. For each time t from 0 to max_time with step dt:
   a. For each unique pair of asteroids:
      i. Calculate their positions at time t.
      ii. Compute the distance between them.
      iii. If distance < sum of radii AND not already recorded:
         - Record collision time for the pair.
4. Sort all recorded collisions by time.
5. Write the collision data to an output file.
6. (Optional) Animate asteroid movement with 3D visualization.

# Implementation Details:

## Programming Language and Libraries

- **Language**: Python
- **Libraries**:
  - math: For distance calculations
  - numpy: For array operations
  - plotly.graph_objects: For 3D animation and visualization

## Code Structure:

- Asteroid: Class definition representing an asteroid.
- parse_input(): Reads and parses asteroid data from a text file.
- detect_collisions(): Simulates motion and detects collisions.
- write_output(): Writes collision results to output file.
- animate_simulation_3d(): Generates 3D animated visualization.
- main(): Entry point for the simulation process.

## How to Run the Code in Jupyter Notebook:

**Install Required Libraries**: !pip install numpy plotly

**Prepare Input Data:** Create or upload an input file called asteroids.txt in the same directory as your notebook.

# Dataset Handling :

### Input Format

Each line in asteroids.txt can be either**:**

- id x y radius vx vy
- x y radius vx vy (auto-assigned ID)

## Example Input

- **100211:**
  Asteroid ID = 1, Position = (0,0), Radius = 2, Velocity = (1,1)
- **5510.5:**
  Auto-assigned ID, Position = (5,5), Radius = 1, Velocity = (0.5,0.5)

## Parsing Logic

- Skip malformed or non-numeric lines.
- Assign IDs automatically if not provided.
- Create instances of Asteroid class.

## Output Format

### Output File: collisions.txt

- Each line: time id1 id2
- Time is rounded to 1 decimal place.

# Challenges and Solutions

### Challenge 1: Precision in Collision Detection

**Issue**: Smaller time steps were necessary to detect fast-moving asteroid collisions.
**Solution**: Adjusted time_step to a sufficiently small value (e.g., 0.1s).

### Challenge 2: Visualizing 2D Motion in 3D

**Issue**: Static plots didn’t convey motion well.

**Solution**: Used Plotly's Scatter3d to animate the asteroid motion with z=0.

### Challenge 3: Background Visualization

**Issue**: Wanted more immersive space-like background.

**Solution**: Modified the layout of the 3D scene to include a dark background and light-colored stars.

# Test Case Results

### Sample Test Case

Input: 5 asteroid entries in asteroids.txt.

- Expected: 5 collisions.
- Detected: 5/5 collisions correctly.
- Output matches expected format and order.

### Performance

- Time Complexity: $O(n^2 * t)$ where t = number of timesteps.
- Memory Usage: Efficient, as it stores only relevant trajectory data.

# Future Improvements

- **Optimization**: Use spatial partitioning (e.g., quadtree) to reduce $O(n^2)$ complexity.
- **3D Simulation**: Add realistic z-axis movement and gravitational interaction.
- **Improved Visualization**:
  - Add animated background.
  - Show impact animations or trajectory changes.

# Resources Used

- **City Dataset** (conceptual reference): Inspired the use of structured text input for asteroid parameters.
- **Jupyter Notebook**: Used as the primary development and execution environment, allowing step-by-step testing, visualization, and debugging.
- **Libraries**:
  - math: For distance calculations.
  - numpy: For efficient array and numerical operations.
  - plotly.graph_objects: For interactive 3D animation and visualization of asteroid movement.