Assignment 6: Physics simulation: collision

EC602 Design by Software

Fall 2017

Contents

1 Introduction

This problem involves predicting and movement and collisions of a large number of objects moving in a two-dimensional space.

The problem is relevant, for example, to air-traffic control, self-driving cars, or game simulations (asteroids, curling, pool)

The objects undergo perfect elastic collision.

1.1 Assignment Goals

The assignment goals are to help you learn about physical modeling for a simulator or game.

1.2 Group Size

For this assignment, the maximum group size is 3.

1.3 Due Date

This assignment is due 2017-10-26 at midnight.

1.4 Submission Link

You can submit here: collision submit link

1.5 Points

This assignment is worth 10 points (5 for C++, 5 for python)

2 Collision Detector

Your task is to determine collisions between moving objects given a starting scenario.

The program will be called "collision.cpp" or "collision.py" (one for C++, one for python)

2.1 Starting scenario

The objects are specified in a file which gives an ID, a location at time $t = 0$ (x/y coordinates in meters) and velocity (x/y coordinates, in meters/second).

2.2 Motion model

The objects travel on a two-dimensional surface called the “field”. Each object travels in a straight line at a constant velocity. Each object will continue travelling in this direction forever, unless it collides with another object.

2.3 Collision model

Each object has a radius of 5 meters.

Two objects collide at the moment the distance between them becomes equal to the *collision distance*. For this problem, the collision distance is defined as 10 meters.

When two objects collide, they undergo elastic collision. The objects have equal mass.

See wikipedia: elastic collision and the vector equations (not the ones with trig) in section “Two-dimensional collision with two moving objects”.

Time did not exist before $t = 0$, so if two objects would have collided in the past, we ignore this event. It never happened.

3 Program requirements

3.1 Input format

The input will be from *stdin*

Each line of input specifies one of the objects. It specifies the object ID, the location at time $t = 0$ (x/y coordinates in meters) and the initial velocity (x/y components, in meters/second).

Here is an example of the format for three objects:

```
2MU133 -34.94 -69.13 0.468 -0.900
0WI913 -43.08 92.12 -0.811 -0.958
6UP738  2.97 -66.25 -0.077 0.074
```

The first object has a license plate or ID number 2MU133.

Its initial position is (-34.94,-69.13) and its initial velocity is 0.468 m/s in the x-direction and -0.900 m/s in the y-direction.

This is the first part of the file called random10.coordinates

3.2 Program configuration

Your program should print out a report for each future time specified on the command line, in order of increasing time.

For example

```
collision 10 50
```

This program will print out a positional report for times 10 seconds and 50 seconds.

```
collision 10 50 <random10.coordinates >random10.results
```

This saves the results into a text file called `random10.results`

3.3 Output format

Your program should print the time followed by the object ID, location, and velocity of each object, like this:

```
10
2MU133 -34.94 -69.13 0.468 -0.900
0WI913 -43.08 92.12 -0.811 -0.958
6UP738  2.97 -66.25 -0.077 0.074
50
2MU133 -34.94 -69.13 0.468 -0.900
0WI913 -43.08 92.12 -0.811 -0.958
6UP738  2.97 -66.25 -0.077 0.074
```

Note that the positions and velocities should be updated.

The objects should be output in the same order they were specified in the input.

3.4 Notes

- The ID is a single contiguous string in any format.
- the objects are never in an initially collided state.
- you may assume that every object is at least the collision distance away at time $t = 0$, and the first collision occurs at $t > 0$.

3.5 Error handling

3.5.1 Bad input

If the input format has any problems (too many fields on one line, invalid numbers, etc) the program should exit with return value 1.

3.5.2 Command line problems

If any value on the command line is not a number, the program should exit with return value 2.

Negative time values should be ignored.

If there are no valid values on the command line, the program should exit with return value 2.

4 The assignment requirements

You should implement the program in C++ `collision.cpp` AND in python, `collision.py`

4.1 Visualization tools

The following are python scripts for visualizing the problem:

- visualize takes an input situation and draws the positions and velocities
- visualize_motion takes the output of `collision` programs and creates a movie using matplotlib.