CSCI 1100 — Computer Science 1 Homework 7
Dictionaries and Classes

Homework Overview

This homework is worth **130 points** toward your overall homework grade and is due **Thursday April 21, 2016 at 11:59:59 pm**. It is the longest homework of the semester and consists of two completely independent parts. The first part provides practice with using dictionaries, while the second part requires the use of classes. Lab 9, which will be posted shortly, also provides practice with classes, and we strongly suggest you begin work through at least part of that before you start with Part 2 of this homework.

These things are all a bit crammed together because lab had to be canceled during GM week. As a final note, **Homework 8**, due during the last week of the semester, will give you practice combining dictionaries and classes.

As always, good coding style is expected and low quality code will be penalized.

Part 1: The N Busiest Individuals in the IMDB

In this problem you will continue to explore the Internet Movie Database starting from the Lectures 16 and 17 example code using dictionaries. First, we weren't quite right about how many movies each individual was involved in because there are many cases where an individual is listed more than once for each movie. Therefore, in deciding how many movies an individual has been involved in we need to avoid counting a movie more than once for each individual. Your solution to the following problem will depend on getting this right.

Here is the main problem: given a positive integer N, provided by the user, output the top N individuals involved in the most movies. If there are ties for the N-th individiual you will have to print out all tied individuals. Here is example output for $N = 20$:

```
Enter the name of the IMDB file ==> imdb_data.txt
imdb_data.txt
Enter the number of top individuals ==> 20
20
338: Blanc, Mel
205: Dunn, John W.
121: Morrison, Tom
120: Maltese, Michael
 93: Foster, Warren
 89: Lee, Christopher
 87: Carradine, David
 86: Caine, Michael
 80: Madsen, Michael; Pierce, Tedd
 79: Estevez, Joe; Hopper, Dennis
 76: Roberts, Eric
 75: Pleasence, Donald
 74: Fine, Larry; Howard, Moe
 72: Black, Karen; Mitchell, Cameron; Wynn, Keenan
 70: Butler, Daws; Cushing, Peter
```

Please follow this output format carefully. Note that the tied individuals are output in alphabetical order.

You may use any code we have provided you for Lectures 16 and 17. The key to the solution is to make effective use of dictionaries, and in fact part of your grade will depend on this. Use of functions here is suggested but not required. Submit your solution as `hw7_part1.py`.

Part 2: CS 1 Birds

Computer games such as Angry Birds are based on simulating the basic rules of physics, or at least some rough approximation to these rules. These simulations involve a simple basic loop where in each iteration

1. The positions and sometimes the velocities (not here) of moving objects are both updated by a small amount.
2. Objects are checked for collisions, and changes are made to the simulation based on these collisions.

While never 100% accurate, realistic looking results and physically useful predictions (for scientific simulations) can be obtained by making sure the changes in each loop iteration are small.

We will apply this idea to a simple version of angry birds called *CS 1 Birds*. Along the way you will get practice writing classes.

The simulation occurs over a rectangular region whose lower left and upper right corners are locations $(0, 0)$ and $(1000, 1000)$. The simulation will include birds and pigs with both represented by circles. The pigs are stationary, but the birds will move along a line (no gravity!). Each bird will move in turn, slowing down when it strikes a pig, and stopping when it becomes too slow or when it goes outside the game rectangle. When a bird strikes a pig, the pig will "pop" and disappear from the simulation. The simulation ends when either all pigs have been "popped" or when all birds have stopped, whichever occurs first.

Input Files

The program should ask the user for two input files, one for the birds and one for the pigs. Each line of the bird file will be in the form

```
name|x0|y0|radius|dx|dy
```

where `name` is a string giving the name of the bird, `x0` and `y0` specify the bird's initial position (center of the circle), `radius` is the bird's radius, and `dx` and `dy` specify how far the bird moves in each time step.

The pig file will have a similar format. Each pig will be on one line in the form

```
name|xc|yc|radius
```

where `name` is a string giving the name of the pig, `xc` and `yc` specify the pig's center position, and `radius` is the pig's radius.

You may assume all input is properly formatted, all birds are entirely inside the bounding rectangle at the start, and none of the birds or pigs intersect each other at the start. You may also assume that each bird's movement (dx, dy) is much smaller than the radii of the pigs, so that you do not need to worry about a bird completely passing through a pig in one time step.

After reading all birds and pigs, your program should have two lists, one of `Bird` objects and one of `Pig` objects. At this point print out the number of birds, the name of each bird and its starting center location, the number of pigs, and the name of each pig and its center location. The order should be the order the birds/pigs were read in. Follow the format of the output examples we have provided in the separate files.

Simulation and Output

The simulation starts at time 0 with the first bird in its initial location. For this, output

```
Time 0: Freddie starts at (28.3,29.1)
```

All position output should be accurate to 1 decimal place. Please follow the output precisely as shown — it should be fairly straightforward

In each time step:

1. Increment the time counter
2. Move the current bird by its dx and dy values. (To be clear, the first bird's first step will be at time 1.)
3. Check to see if the bird's circle intersects the circle of a pig that has not yet been popped. (Intersection occurs when the distance between the two circle centers is less than or equal to the sum of the two circle radii.) If so,
   (a) Print one line giving the time, the name of the bird, the position of the bird, and the name of the pig. For example,
   ```
   Time 25: Freddie at (15.3,19.1) pops Wilbur
   ```
   (b) "Pop" the pig. One easy thing to do is to remove the pig from the list of pigs.
   (c) Decrease the dx value (the x speed ONLY) of the bird by 50% (Throughout your code you might want to make sure that python knows that the speed is a float...) Note that this changes the direction the bird is heading. Print one more line,
   ```
   Time 25: Freddie at (15.3,19.1) has (dx,dy) = (4.5,5)
   ```

   Note: the data will be such that at most one pig can be popped in one time step.
4. Check two more things in the following order:
   (a) If the bird reaches a speed below `MINSPEED==6`, stop moving the bird, remove it from the simulation, and move on to the next bird. (Note that the speed is $\sqrt{dx^2 + dy^2}$.) When this occurs output the time, the name, the location and the speed the bird, e.g.
   ```
   Time 281: Super Chicken at (668.0,364.7) with speed 4.6 stops
   ```
   (b) If any part of the bird's circle has gone outside the rectangle (its x position minus its radius is less than 0 or its x position plus its radius is greater than 1000, or its y position minus its radius is less than 0 or its y position plus its radius is greater than 1000). When this occurs, output the time, the name and location of the bird. For example,
   ```
   Time 25: Big-Bird at (975,234) has left the game
   ```

   When a bird stops or leaves the field and there are more birds and more pigs left, start the next bird immediately at its initial location. Output something like
   ```
   Time 25: Daffy starts at (123.7,88.5)
   ```
   Just to be clear, the time Daffy starts is the same time the previous bird (Big-Bird) leaves the board. Daffy does not move until the next time.
5. If all pigs are popped then output a message saying something like

```
Time 33: All pigs are popped. The birds win!
```

and stop the game.

6. Otherwise, if there are no more birds, output a message with the names of the surviving pigs (ordered by their order of input). For example,

```
Time 33: No more birds. The pigs win!
Remaining pigs:
Porky
Brick
```

and stop the game.

The logic of this is a bit complicated, with a number of cases to check, so implement it carefully!

Use of Classes and Functions

You **must** use at least two classes, one for a pig and one for a bird. The pig class in our solution is very simple:

```
class Pig(object):
    def __init__( self, n, x0, y0, r0 ):
        self.name = n
        self.x = x0
        self.y = y0
        self.radius = r0
```

and is placed in `Pig.py`. However, we strongly suggest you put a lot of functionality into the `Bird` class.

To help you get started, we recommend that you begin with the parsing of the input file, in which your code should create two lists, one of bird objects and one of pig objects. Writing this parsing code will require that you write the initializer function for the `Bird` class, which should be very similar to the initializer for the `Pig` class shown above.

To determine the remaining functionality of the `Bird` class, we suggest you outline the main functionality of the overall program and even start to write the main loop and the functions. Stick close to the outline we gave above. As you do this, the methods that you want the `Bird` class to have will become more clear, and as you think of these methods, you should make a note of what these functions should be in your `Bird.py` file. You can do this without actually writing the methods. In our implementation, the Bird class does things like (a) move the bird, (b) check for a collision between the bird and a pig (the pig is passed as a parameter!), and (c) check to see if the bird has crossed the boundary of the rectangle. There are others.

Test Cases

We will post four data files to help you in your development and testing.

- `birds1.txt` and `pigs1.txt`. There are two pigs and none of the birds will pop either. This will allow you to test just your code that moves the birds and checks the boundaries. In this case, of course, the pigs will win.
- `birds2.txt` and `pigs2.txt`. In this case, each bird will pop a pig, and the birds will win.
- `birds3.txt` and `pigs3.txt`. This is a more involved example where the pigs win.
- `birds4.txt` and `pigs4.txt`. This is a more involved example where the birds win (Daffy pops 2 pigs, and the others each pop one).

We will put these exact four test cases on the HSS as part of the verification testing, and will give substantial partial credit for handling even some of them correctly. Full credit, of course, will require that you handle all test cases correctly including these four.

Module Organization and Submission Instructions

Please follow these instructions carefully: Your program should be split across three files, `Pig.py`, `Bird.py`, and `hw7_part2.py`. `Pig.py` can be exactly as described above. `Bird.py` will start with

```
import math
from Pig import *
```

and then continue with the `Bird` class implementation and any testing functionality you write. `hw7_part2.py` should start with

```
from Pig import *
from Bird import *
```

and include the rest of your code. All code should follow our coding guidelines. If you need a refresher, look at HW6.

In order to submit your solution: Submit a zip file called `hw7.zip` containing exactly the three files `Pig.py`, `Bird.py` and `hw7_part2.py`.