CSCI 1100 — Computer Science 1 Homework 8
Classes
A Bird in the Hand

Homework Overview

This homework is worth **120 points** toward your overall homework grade and is due **Thursday November 17, 2016 at 11:59:59 pm**. It consists of three interrelated parts that we will use to build up an **Angry Birds** simulation. Lab 9 should have given you at least some practice and insight with classes. Use what you learned!

As always, good coding style is expected and low quality code will be penalized. Also, please reread the collaboration guidelines we provided with HW 3. This late in the semester, there should be no more confusion about how much collaboration is allowed. Please make sure that you develop your code independently and that you follow the guidelines in HW 3 carefully. We will continue to use a source code comparison tool to detect code that is too similar to have come about by chance.

Part 1: 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 $(1000, 1000)$. **Take note of the playing field.** This is different than we used for the wandering trainer. It now corresponds to the upper right quadrant of a graph. The simulation will include birds, pigs, and barriers all represented by circles. Bird, pig and barrier positions are in **floats** and correspond to points in cartesian or `(x,y)` coordinates. The pigs and barriers are stationary, but the birds will move along a line (no gravity!). Each bird will move in turn, slowing down or stopping when it strikes a pig or a barrier, and stopping when it becomes too slow or when any part of it goes outside the game rectangle. When a bird strikes a pig, the pig will "pop" and disappear from the simulation. When it strikes a barrier, the barrier will take damage, but may or may not disappear depending on its initial strength and how many times it has been hit. The simulation ends when either all pigs have been "popped" or when all birds have stopped, whichever occurs first.

We can summarize the possible behaviors as follows:

- Bird
    1. fly - Update the current position by dx and dy

2. collide with a pig - Decrease x velocity by a factor of 2
3. collide with a barrier - Total velocity becomes 0
4. stops - total velocity becomes `< 6`, or the bird circle crosses the field rectangle

- Pig
  1. bird collides with it - Pops!
- Barrier
  1. collide with a bird - Takes mass times velocity squared damage. Do not let the strength go below 0
  2. crumbles if strength is reduced to 0

Input Files

The input for the exercise will be stored in three separate files, one each for the birds, the pigs, and the barriers. Each line of the bird file will be in the form

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

where `name` is a string giving the name of the bird, `mass` is the bird's mass (used to calculate damage to barriers), `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. All values other than `name` are floats.

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. All values other than `name` are floats.

And finally each line of the barrier file will be in the form

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

where `name` is a string giving the name of the structure, `strength` is a measure of how much damage it can take before being destroyed, `x0` and `y0` specify the structure's position (center of the circle), and `radius` is the structures radius. All values other than `name` are floats.

Simulation and Output

At the start of the simulation ask for the bird file, the pig file and the barrier file and read the data in to create lists of bird, pig and barrier objects. 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 and barriers so that you do not need to worry about a bird completely passing through a pig in one time step.

After reading all birds, pigs, and barriers, your program should have three lists, one of `Bird` objects, one of `Pig` objects and one of `Barrier` 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 number of barriers, the name of each barrier and its center location. The order should be the order the birds/pigs/barriers were read in. Follow the format of the output examples we have provided in the separate files.

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 or barrier. 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) = (4.5,5.0)
   ```

   (d) The data we provide is arranged such that only one pig will be struck at a given time step. Once you have popped the first pig, you can stop reviewing the rest of the pig list.
4. Check to see if the bird's circle intersects the circle of a barrier that has not yet crumbled. (Again, 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, the name of the barrier and the barrier's strength. Be sure to not let the barrier go below 0 strength.

   ```
   Time 25: Freddie at (15.3,19.1) hits Picket, Strength 0.0
   ```

(b) If the barrier's strength is dropped to 0, the barrier crumbles and an additional message should be printed

```
Time 25: Picket crumbles
```

(c) Decrease the dx and dy of the bird to 0. Print one more line,

```
Time 25: Freddie at (15.3,19.1) has (dx,dy) = (0.0,0.0)
```

(d) Again, the data we provide is arranged such that only one barrier will be struck at a given time step. Once you have struck the first barrier, you can stop reviewing the rest of the barrier list.

5. 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 (975.0,234.0) 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, and therefore no tests against pigs or barriers are needed.

6. 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.

7. 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 three classes, one for a pig, one for a bird, and one for a barrier. The pig class is the smallest and can be as simple as:

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

although we added other functionality to our code to make some of the reporting and printing easier. You are welcome to use this part of the pig defintion in your code. We will not consider it in our code comparison. The barriers and birds are increasingly more complicated and we suggest that in particular you add a lot of functionality into the `Bird` class to make the simulation easier.

Test Cases

This is the longest and one of the most complicated assignments we have given so far. The program match portion (autograding) will be worth 90 of the 120 points available for the homework, and we want you to have an opportunity to get substantial parital credit even if you are having trouble completing all three classes correctly. To achieve this, we will be providing you with three scenarios of varying difficulty:

- Scenario 1 will be worth 40 points and will not include any collisions. You will need to read in and report on the birds, pigs and barriers, but when the simulation starts the birds will fly without hitting anything to the end of the board. Obviously, the pigs will win. Example files `bird1.txt`, `pig1.txt` and `barrier1.txt` we provided exemplify this scenario.
- Scenario 2, worth 30 points and exemplified by `bird2.txt`, `pig2.txt` and `barrier2.txt`, will add collisions between birds and pigs, but not striking of barriers by birds.
- Scenario 3, worth 20 points and exemplified by (you guessed it) `bird3.txt`, `pig3.txt` and `barrier3.txt`, will also include birds striking barriers.

Thus, if you just implement the code for scenario 1 you can earn up to 44% of the autograde points. Adding in collisions with pigs can get you all the way up to 78% of the autograded points. Two important notes:

1. Some of the tests we use on Submitty will be different from these example files.
2. Do not ask for help on a later scenario until you can prove that your code works for an earlier scenario. In other words, get Scenario 1 working before you even consider moving on to Scenario 2, and get Scenario 2 working before you even consider Scenario 3

Module Organization and Submission Instructions

Please follow these instructions carefully: Your program should be split across four files, `Pig.py`, `Bird.py`, `Barrier.py` and `hw8.py`. `hw8.py` should start with

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

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

We recommend that when you program your solution you start simple and build up:

1. Create the three class files, `Bird.py`, `Pig.py`, and `Barrier.py` files and write initializers for them.
2. Write the code to ask for the filenames, create the lists, and report on the initial list contents.
3. Test to make sure your input is working correctly
4. Add flying to the bird and write the simulation loop without worrying about collisions
5. Test and verify that you correctly solve the first scenario. **At this point you have earned 40 of the 90 autograding points**. Save this file as a backup.
6. Without breaking scenario 1, add in code to let birds collide with pigs.
7. Test and verify that you correctly solve the first and second scenarios. **At this point you have earned 70 of the 90 autograding points**. Save this file as a backup.
8. Without breaking scenarios 1 or 2, add in code to let birds collide with barriers.
9. Test and verify that you correctly solve the first, second and third scenarios. **At this point you have earned 90 of the 90 autograding points**. Save this file as a backup.

The output from the scenarios and the scenario input files can all be found in `hw08files.zip`. **In order to submit your solution:** You will need to submit your homework by dragging all four of the files `Bird.py`, `Pig.py`, `Barrier.py` and `hw8.py` into the Submitty submission block. Be careful to name them correctly.