CISC 140: Lab 3

Contents

1 Introduction

This programming lab will expand upon object-oriented design principles practiced in Lab 2 and that we have been reinforcing in class. This lab will require you to show that you understand these design and implementation principles by testing, altering, and extending objects discussed in Chapter 3.4.

The project for this lab is based upon several of the Creative Exercises from Section 3.4 in the textbook.

2 Testing the Body Class

The textbook defines the `Body` class on page 500 to represent an object with mass and gravity. This class is central to our simulation since our `Universe` is designed to simulate how the gravity of two or more `Body` objects interact and cause the bodies to interact. However, in order to have faith in our simulation, we must first have faith that our Body object is collecting the proper information and accurately representing a body with mass.

Test Plan for the Body Class **5pts.**

Design a test plan for the Body class. What conditions for each method will you need to test? What will be the expected outcomes? Be sure to include discussions of special methods, input conditions that should break the object, and how will you check object state?

The test plan for the body class is to make the bodytest.py code in the test code the first method tested should be the method to that creates the bodies in the body.py code there is code for the position, velocity and mass. so in the test code give it the position velocity and mass.

Next is the move method in the test code I have a set number of seconds for the td.

the method forceFrom which handles the distance and in the test code i have the it moaginitue is the force from body 1 to body 2 and vise-versa

finally is the draw method that draws the bodies and animation of the velocity of not 0

Test Body Implementation

Implement the test plan for the Body class as designed above. This code should be included in an appropriate if statement (if __name__== "__main__") within the body.py file. **5 pts.**

Analysis Element

Based upon the above test code, what is wrong with the Body class (with respect to sanitizing of inputs)?

The issue with the body class is that it will create the bodies even though they are not vectors and later down the line when the program needs them then the program will be break. To insure this is not an issue i will implement a float for mass and for position and velocity i will implement a isinstance to raise a TypeError is if the position and velocity in the test code is not an array which would be which will error out and throw the error in the correct spot where the error is occurring which will be in the creation of the body.

Write code that will appropriately handle these conditions and ensure that Body objects are not corrupted by bad inputs. **5pts.**

3 Body Modification

Up until this point, we've been just making the base Body object do what it's supposed to, but better. Now we want to start to tweak the Body class so that it, or its subclasses, behave differently.

Body Scaling

Right now, all objects represented by the Body class are the same size, regardless of mass. How would you change the design of the Body class to allow the object to be displayed proportionally to its mass. Be specific about the methodology as well as your mass to size ratio.

i will simply add size to the arguments in the body init

then i will define the size and add a float to it so that it only takes real

numbers and to keep the size from 0.0 - 1.0. in the test code i will

update the body and add a different size to each body.

Write code that will implement this change in the Body class. **5pts.**

Body Scaling Redux: Design **5pts.**

This change of a "proportionally large" Body is more accurate than the "fixed size" object originally defined. However, it is still not accurate as objects of different masses may be of very similar size. As an example, a white dwarf is the remnant of a low to medium mass star when it reaches the end of its stellar life cycle. Per NASA, "[a] typical white dwarf is about as massive as the Sun, yet only slightly bigger than the Earth."

If we wanted to create a class where size is proportional to both mass and relative density, how would you do this? Use proper object-oriented design principles to describe what you would need to do to create a new class called DensityBody that would include this change in state and behavior:

This new class will similarly to the body class with the exception of having density in the parameters. Both position and velocity will have the type error check and mass and density will have the float for real numbers to be used. Then i will have a function for where density is is divided by mass and it should return the number. draw function is next. What this will change and be different from the body class is that since the density of a planet is light then the planet floats and if its denser it sinks so depending on the number returned the drawing will mimic.

Body Scaling Redux: Implementation

Implement the DensityBody class as a new class within the body.py code file. Be sure to alter the test plan already contained within the file to include testing for the DensityBody class on top of the Body class as designed above. **10 pts.**

Gravity-less Bodies? **5pts.**

Suppose we wanted to create a class where the size of the body is proportional to the mass and density of the body but which exerts no gravity. How would you design this class?

Will be the same as the body class but in the body-test there will be no values passed in for the velocity and i have the bodies a position of 10 for the x and y which will be positive and they will be planted to the top of the canvas

Implement this class as the NoGravityBody class within the body.py code file. Be sure to alter the test plan already contained within the file to include testing for the NoGravityBody class on top of the Body class as designed above. **5 pts.**

Regular & Gravity-less Bodies? **5pts.**

So…what happens if we have a universe of just gravity-less bodies? What is that universe like? What about a universe where some bodies have gravity and others don't?

If there is no gravity then the bodies would just float to the top of the canvas. If the universe has gravity but some of the bodies dont then the ones that do will take there natural course according to the math and the ones that done have gravity they will just float to the top

Designing A Three-Dimensional Universe **5pts.**

Currently, our bodies are created in two dimensions and our Universe class simulates in a similar way. What aspects of the DensityBody and/or Universe classes would have to be changed to accommodate a three-dimensional Universe? How would they need to be changed?

Implementing A Three-Dimensional Universe

Implement two new classes, DensityBody3D and Universe3D, to implement the above functionality. Add the DensityBody3D class as a new class within the body.py code file. Create a new file universe3D.py to contain the Universe3D class. Be sure to alter the test plan already contained within the file to include testing for the DensityBody3D class on top of the Body, DensityBody, and NoGravityBody classes as designed above. Also, be aware that you may need to alter the input files feeding the simulation. **10 pts.**

Designing A Random Body **5pts.**

This is all well and good but simulations are not terribly useful if we need to seed the simulation with specific values each time we run it. What would be more interesting would be to have DensityBodies that could be randomly generated to see how they would interact. This object would have its instance variables randomly initialized to "sane" values, then the system would be simulated.

Define the process of initializing these random DensityBodies. What bounds will you put onto the various instance variables of each object and why?

Implementing A Randomized Simulation

Implement two new classes, RandomBody and RandomUniverse, to implement the above functionality. Add the RandomBody class as a new class within the body.py code file. Create a new file random_universe.py to contain the RandomUniverse class. Be sure to alter the test plan already contained within the file to include testing for the DensityBody3D class on top of the Body, DensityBody, NoGravityBody, and DensityBody3D classes as designed above. **15 pts.**

4 A New Universe

Everything that we've done up until this point in the lab has been based upon our universe with the rules and constants defined within that universe. But just because we experience these things in this way does not mean that the universe had to be that way (although it might need to be this way for us to exist). So, let's play God and muck with the fundamental features of the universe! (At least as far as this simulation is concerned.)

Designing A New Universe **5pts.**

"Design a new universe with interesting properties…This exercise is an opportunity to be creative!"

We want you to design here the rules of the new universe. You may play with constants, features, behaviors, or whatever you want and may base you derived bodies/universe off of any of the previous designed and implemented bodies/universes from this exercise. Just be sure to explain to us which you're starting with and what your rules are.

Predicted Behavior **5pts.**

How do you imagine bodies in this newly-conceived universe will behave? Why? Be as specific as you think you can be.

Implementing the New Universe **10pts.**

Implement two new classes, NewBody and NewUniverse, to implement the above functionality. Add the NewBody class as a new class within the body.py code file. Create a new file new_universe.py to contain the NewUniverse class. Be sure to alter the test plan already contained within the file to include testing for the DensityBody3D class on top of the Body, DensityBody, NoGravityBody, DensityBody3D, and RandomBody classes as designed above.

Does your universe behave as predicted? If so, how? If not, how and why do you think this is?

5 Rubric

Question	Points	Score
Testing the `Body` Class	15	
Body Scaling	20	
Gravity-less Bodies?	10	
A Three-Dimensional Universe	15	
Randomly Determined Simulation	20	
New Universe	20	
Total:	100	