SFWR ENG 2AA4: Assignment 2 Solution

Meet Patel, patelm16

March 7, 2017

The purpose of this software design exercise is to write a Python program which creates, uses, and tests an ADT for points, lines and circles. Then, an abstract object module storing a deque of circles is also implemented and tested. The program consists of the following files: `pointADT.py`, `lineADT.py`, `circleADT.py`, `deque.py`, `testCircleDeque.py`, and `Makefile` as shown in the Appendices at the end along with my partner's `circleADT.py` file.

Contents

1 Testing Results

1.1 Results of Testing My Files

Running my testCirclesDeque produced the following output: All test cases from all modules have passed. .... ———————————————————————————- Ran 34 tests in 0.042s

OK

Function Tested	Result	Additional Comments
xcrd	pass	correct x-coordinate
ycrd	pass	correct y-coordinate
dist	pass	correct distance
rot	pass	cases: +, -, 0, and >Pi
beg	pass	correct beg point
end	pass	correct end point
len	pass	correct length of line
mdpt	pass	cases: diff points and line length zero
rot	pass	correctly used PointADT rot
cen	pass	correct centre point
rad	pass	correct radius
area	pass	correct area
intersect	pass	correctly checked intersection
connection	pass	correct beg and end points
force	pass	correct force calculated
pushBack	pass	correct push at back
pushFront	pass	cases: regular push front, full deque push with exception
popFront	pass	cases: regular pop front, empty deque pop with exception
popBack	pass	correct pop from back
back	pass	correct value
front	pass	correct value
size	pass	correct size
disjoint	pass	cases: 3 diff circles, one circle, exception case
sumFx	pass	correct sum
totalArea	pass	correct area
averageRadius	pass	correct radius

Table 1: Results of testing my files

1.2 Testing the Makefile

To check that my makefile was working correctly, I had to make sure that calling make test correctly ran my testCircleDeque module and calling make doc correctly created the HTML and Latex folders for my entire project. I was able to use my knowledge from 2XA3 and the example on the course's gitlab page to get my makefile to work properly.

1.3 Rational for My Test Case Selection

While selecting the cases I wanted to test in testCirclesDeque, I first looked at the marking scheme and made that I incorporate all the required cases. Then I made sure I tested every method at least once so that I know that my methods work. I also tested to see my rot function worked with positive, zero, greater than pi, and negative inputs to double check the math. From the lineADT module I tested the mdpt method more than once with other case being the fact that the length of the line is zero. I also tested my deque thoroughly by making sure the exceptions work for a full and empty deque, as well as the correct result of popping and pushing from the correct side and having the correct circle left in the deque. I also made sure to test disjoint with three different circles, and one circle only.

1.4 Results of Testing Partner's Files

Running my testCirclesDeque on my partner's circleADT produced the following output:

```
All test cases from all modules have passed. ................................. ——————————————
——————————————————————————- Ran 34 tests in 0.042s
   OK
```

Function Tested	Result	Additional Comments
xcrd	pass	correct x-coordinate
ycrd	pass	correct y-coordinate
dist	pass	correct distance
rot	pass	cases: +, -, 0, and >Pi
beg	pass	correct beg point
end	pass	correct end point
len	pass	correct length of line
mdpt	pass	cases: diff points and line length zero
rot	pass	correctly used PointADT rot
cen	pass	correct centre point
rad	pass	correct radius
area	pass	correct area
intersect	pass	correctly checked intersection
connection	pass	correct beg and end points
force	pass	correct force calculated
pushBack	pass	correct push at back
pushFront	pass	cases: regular push front, full deque push with exception
popFront	pass	cases: regular pop front, empty deque pop with exception
popBack	pass	correct pop from back
back	pass	correct value
front	pass	correct value
size	pass	correct size
disjoint	pass	cases: 3 diff circles, one circle, exception case
sumFx	pass	correct sum
totalArea	pass	correct area
averageRadius	pass	correct radius

Table 2: Results of testing my partner's files

2 Discussion

2.1 Discussion of Test Results

When testing my modules, I had to make sure that the edge cases were convered as well as the expected cases. Testing for the expected cases was simple as I was able to get my functions other than disjoint, sumFx, and force to work successfully the first time. For many functions, I had more than one test case to check if the fucntion worked with various kinds of inputs. At the end, when I was successful with all my test cases, I saw that it was a lot of trial and error to get things to work correctly especially with the disjoint and sumFx function. Also, for the hard-coded values, I had to first run the program to see what numbers python gives and also check that it matches to the real calculated value before I used it in my test case. I feel that I covered all necessary test cases from all the modules.

With my partner's module being similar to mine, it suprisingly passed all my test cases. I think this is due to the formal specification where we both had similar code and her circle module code perfectly passed my test cases.

2.2 What I Learnt

One of the biggest things I learnt from doing this assignment was that through a formal specification, it is slightly easier to implement the methods as there is less thinking to be done. I could see that this could be harder as well in other cases, but in this case, the specification made it easier. This assignment also improved my Python coding as I often use Java. The coding seemed slightly easier than assignment one as I was more used to it and also due to the fact that there were lesser thing to worry about than when using Java to code. Although using Doxygen seemed to be tedious at first, I found that commenting in it was very standardized and helpful with files it ends up creating. This assignment also taught me to better test my programs as I ended up making a couple mistakes prior to testing and debugging and testing helped me find out where those mistakes were at. Using PyUnit was very helpful as it was a quicker and more standardized way of testing. Learning about and of Latex was very important to me as I actually find it much quicker to use than Microsoft Word. Lastly, this assignment also me improve my time management and problem solving skills as I thought the testing of the program would be quick and easy; it ended up being the harder part of the assignment as it took a long time to think and figure out the defects of my program. The functions disjoint, sumFx, and force were interesting as they took time and thought to complete. All in all, this assignment was a great learning experience with the new concept of abstract objects as well.

2.3 Problems Found

My program did not have flaws which I found after its completion. An issue I was intially having was coding the methods in abstract object deque as they were slightly different from ADTs. However, I was able to use the Sequence from lecture 10 to learn how to code abstract objects correctly. I also intially found PyUnit tricky to use but that was only because I had never used it before. However, at the end of the assignment I feel like I did not have any problems with my program.

My partner's circleADT module was very similar to my circleADT module and hence, I did not find any problems with it. I jut found that the individual may have redundant code in a couple methods, but the workings of the code were the same as mine.

I did not find any issue with the actual specification of the modules since it was formal and we were taught what it part means and how to implement a program using the MIS format. It was initially hard to convert the math to logic I could understand but the names of the functions were helpful in understanding the math.

2.4 Formal Specification vs Informal Specification

I found a formal specification to be alot more helpful and more precise so that the implementations requirements were easier to come up with. It gives an improved understanding of the program and I feel like I tested my program alot less this time around as I knew what I was doing throughout the implementation as exported and imported constants and modules were specified along with methods and their parameters, outputs, and exceptions. In general, I feel like formal specifications allow you to detect errors earlier and makes less of them when compared to implementing on informal specifications like assignment 1. In the first assignment, I tested alot and had to think what some of the specifications meant and if it covered all bases or not so I found it more difficult to work with.

2.5 Advantages of PyUnit Testing Framework

Testing using PyUnit was much more simpler than the testing method used in the first assignment. PyUnit testing was easier to integrate with the overall program as there were a variety of assertion statements which I could use to test booleans, and integer returns as well as exceptions. Like all other testing methods, I feel like using PyUnit would not allow one to catch all bugs of a program. PyUnit was also more structured in the way it was integrated. Looking at testing on a whole, manual testing is also much more time consuming, boring, and costly.

2.6 Specification of totalArea() and averageRadius()

Deq_totalArea():

- output
$$out := +(i : \mathbb{N} | i \in ([0..|s|-1]) : (i.area()))$$
- exception: $exc := (|s| = 0 \Rightarrow \text{EMPTY})$

Deq_averageRadius():

- output
$$out := +(i : \mathbb{N} | i \in ([0..|s|-1]) : (i.rad())/|s|)$$
- exception: $exc := (|s| = 0 \Rightarrow \text{EMPTY})$

2.7 Critique of Circle Module Interface

The circle module's interface was not essential as the intersect method could be evaluated with the connection method. It was however, a consistent module as it was formatted as the other modules and was internally consistent in how is was layed out. The module is general as this specification defines a normal circle and functions to do with normal circles. It is not an opaque interface as the methods are created by us and whatever is created is seen in the actual module. It does not used too many other modules and their functions. Lastly, the minimality of the program goes together with its essentialness in this case. It is not completely minimal as it has a function that could've been implemented with another.

2.8 Deq disjoint() with One Circle

The output of the specification of disjoint with one circle would give false as there is not j[] circle so the and condition of j[] being in the deque set from 0 to |s|-1 would not be satisified. However the main part of the specification is that i does not intersect with j, so that part is satisfied and hence, with that logic, I implemented my code to return true for one circle as there are no intersections when there is only one circle in the deque.

C Code for pointADT.py

```
## @file pointADT.py
#  @title pointADT
#  @author Meet Patel
#  @date 2/27/2017

from math import *

## @brief This class represents a point as an abstract data type
#  @details This class includes functions that can be performed on the point object.
#  It uses values xc and yc which are the x and y coordinates of the point respectively.
class PointT:

## @brief Constructor for PointT
#  @details Constructor accepts two parameters for the x and y coordinate of point
#  @param xc is the x coordinate of the point
#  @param yc is the y coordinate of the point
    def __init__(self, xc, yc):
        self.xc=float(xc)
        self.yc=float(yc)

## @brief Returns x-coordinate of the point
#  @return x-coordinate of the point
    def xcrd(self):
        return self.xc

## @brief Returns y-coordinate of the point
#  @return y-coordinate of the point
    def ycrd(self):
        return self.yc

## @brief Returns distance between 2 points using distance formula
#  @details Function takes in parameter p as the second point
#  @param p contains information on the second point
#  @return distance between 2 points
    def dist(self, p):
        return sqrt((self.xc - p.xcrd())**2 + (self.yc - p.ycrd())**2)

## @brief Rotates point around axis by phi (in radians)
#  @details Function takes in parameter phi as the rotation factor in terms of phi radians.
#  Current x and y coordinates of point are updated with below equations
#  @param phi is radians the point should be rotated by
    def rot(self, phi):
        self.xc = cos(phi) * self.xcrd() + (- sin(phi) * self.ycrd())
        self.yc = sin(phi) * self.xcrd() + cos(phi) * self.ycrd()
```

This code can also be found at the following link: Link

D Code for lineADT.py

```
## @file lineADT.py
#  @title lineADT
#  @author Meet Patel
#  @date 2/27/2017

from math import *
from pointADT import *

## @brief This class represents a line as an abstract data type
#  @details This class includes functions that can be performed on the line object.
#  It uses values b and e which are the first and second points of the line respectively.
class LineT:

## @brief Constructor for LineT
#  @details Constructor accepts two parameters of type point for beginning and end points.
#  @param p1 is the first point of the line
#  @param p2 is the second point of the line
    def __init__(self, p1, p2):
        self.b = p1
        self.e = p2

## @brief Returns point object for beginning point of the line
#  @return beginning point, b, of the line
    def beg(self):
        return self.b

## @brief Returns point object for ending point of the line
#  @return ending point, e, of the line
    def end(self):
        return self.e

## @brief Returns length between line created by 2 points
#  @details Function uses distance method from pointADT
#  @return distance between two points
    def len(self):
        return self.b.dist(self.e)

## @brief Returns midpoint of the line as a point object
#  @details PointT object (midpoint) is created by using average of both existing points'
#  x and y coordinates. Local function avg takes in two values and divides by 2.0
#  to get a float number average
#  @return point object for midpoint of created line
    def mdpt(self):
        def __avg__(x1,x2):
            return (x1 + x2)/2.0
        return PointT(__avg__(self.b.xcrd(), self.e.xcrd()), __avg__(self.b.ycrd(), self.e.ycrd()))

## @brief Rotates points of line around axis by phi radians
#  @details Uses rot method from pointADT to rotate both points in line
#  which ends up rotating the line
#  @param phi is radians the point should be rotated by
    def rot(self, phi):
        self.b.rot(phi)
        self.e.rot(phi)
```

This code can also be found at the following link: Link

E Code for circleADT.py

```
## @file circleADT.py
#  @title circleADT
#  @author Meet Patel
#  @date 2/27/2017

from math import *
from pointADT import *
from lineADT import *

## @brief This class represents a circle
#  @details This class includes functions that can be performed on the circle
#  object. It represents the circle object with a center point (as PointT)
#  and a radius value.
class CircleT:

## @brief Constructor for CircleT
#  @details Constuctor accepts 2 parameters; one for the center of the circle
#  and one for the radius of the circle
#  @param c is the instance of the PointT object which represents the center point
#  @param r is radius of the circle represented as a real number
    def __init__(self, cin, rin):
        self.c = cin
        self.r = float(rin)

## @brief Returns centre point of the circle
#  @return centre point of the circle
    def cen(self):
        return self.c

## @brief Returns radius of the circle
#  @return radius of the circle
    def rad(self):
        return self.r

## @brief This function calculates the area of the circle
#  @return area of the circle
    def area(self):
        return pi * self.r**2

## @brief This function checks if two circles are intersecting
#  @details Two circles are intersecting if they share at least one point.
#  Assuming that points inside a circle and on border of circle are
#  a part of the circle.
#  @param ci is the other circle object (from CircleT) which the first circle
#  object is comparing with
#  @return Boolean value of true or false of whether the two circles intersect.
#  True is returned if they do intersect and false if they do not.
    def intersect(self, ci):
        return (self.cen().xcrd()-ci.cen().xcrd())**2 + (self.cen().ycrd()-ci.cen().ycrd())**2 <=
            (self.rad() + ci.rad())**2

## @brief This function makes a new line between centers of both circles.
#  @param ci is the other circle object from which to make the line from
#  @return Line created from centers of both circles as a LineT object.
    def connection(self, ci):
        return LineT(self.cen(), ci.cen())

## @brief This function calculates the gravitational force between two circles.
#  @details The force between two circles is calculated by using the function
#  f and the formula given in the assignment sheet
#  @param f is a function used to find force between the two circles
#  @return value of force between the two circles with given function
    def force(self, f):
        return lambda x: self.area()* x.area()*f(self.connection(x).len())
```

This code can also be found at the following link: Link

F Code for deque.py

```python
## @file deque.py
#  @title deque
#  @author Meet Patel, patelm16
#  @date 2/27/2017

from circleADT import *

## @brief This class represents a deque of circle as an abstract object
#  @details A list, s, to store the CircleT objects and a constant, MAX_SIZE,
#  to help with the exceptions are globally created
class Deq:

    MAX_SIZE = 20

    s = []

## @brief Constructor which initializes deque s
    @staticmethod
    def init():
        Deq.s = []

## @brief Inserts a CircleT object into the back of the deque
#  @details The CircleT object is pushed assuming that the deque is not full;
#  if it is full, then an exception is raised
#  @param c is the CircleT object user wants to add to the back of the deque
    @staticmethod
    def pushBack(c):
        if len(Deq.s) == Deq.MAX_SIZE:
            raise FULL("Maximum size of deque is exceeded. Cannot add anymore circles.")
        else:
            Deq.s.append(c)

## @brief Inserts a CircleT object into the front of the deque
#  @details The CircleT object is pushed assuming that the deque is not full;
#  if it is full, then an exception is raised
#  @param c is the CircleT object user wants to add to the front of the deque
    @staticmethod
    def pushFront(c):
        if len(Deq.s) == Deq.MAX_SIZE:
            raise FULL("Maximum size of deque is exceeded. Cannot add anymore circles.")
        else:
            Deq.s = [c] + Deq.s

## @brief Removes a CircleT object from the back of the deque
#  @details The CircleT object is popped assuming that the deque is not empty already;
#  if it is empty, then an exception is raised
    @staticmethod
    def popBack():
        if len(Deq.s) == 0:
            raise EMPTY("The deque is empty; there is nothing to remove.")
        else:
            del Deq.s[-1]

## @brief Removes a CircleT object from the front of the deque
#  @details The CircleT object is popped assuming that the deque is not empty already;
#  if it is empty, then an exception is raised
    @staticmethod
    def popFront():
        if len(Deq.s) == 0:
            raise EMPTY("The deque is empty; there is nothing to remove.")
        else:
            del Deq.s[0]

## @brief Returns the CircleT object from the back of the deque
#  @details If deque is empty, then an exception is raised
#  @return the CircleT object from the back of the deque
    @staticmethod
    def back():
        if len(Deq.s) == 0:
            raise EMPTY("The deque is empty. There is nothing to return.")
        else:
            return Deq.s[-1]

## @brief Returns the CircleT object from the front of the deque
#  @details If deque is empty, then an exception is raised
#  @return the CircleT object from the front of the deque
```

```
    @staticmethod
    def front():
        if len(Deq.s) == 0:
            raise EMPTY("The deque is empty. There is nothing to return.")
        else:
            return Deq.s[0]

## @brief Returns the size of the deque
#  @return the length of the deque
    @staticmethod
    def size():
        return len(Deq.s)

## @brief Compares intersection between all CircleT objects in deque
#  @details If deque is empty, then an exception is raised. Additional if
#  statement included to account for when there is only one item in deque,
#  in which case, the assumption is that it does not intersect with any other
#  circles to provide a return value of true. For loops created in a way so that
#  an item is not compared to itself.
#  @return Boolean value returned; True if no circles in deque are intersecting
#  with any other; False returned if 1 or more intersections occur between 2 circles.
    @staticmethod
    def disjoint():
        if Deq.size() == 1:
            return True
        elif Deq.size() > 1:
            for i in Deq.s[:-1]:
                 for j in Deq.s[(Deq.s.index(i)+1):]:
                     if (i.intersect(j)):
                         return False
            return True
        else:
            raise EMPTY("The deque is empty. There are no circles to compare.")

## @brief Method calculates the sum of forces of the first item in deque with all
#  others in the deque.
#  @details If deque is empty, then an exception is raised. Division by zero
#  error is handled in local function, Fx() which is used to calculate force
#  between two CircleT objects using a function, f, provided by user. To account
#  for division by zero, an if statement is used.
#  @param f is the function inputted by user for which the force between two
#  CircleT objects is calculated with
#  @return the total sum of all forces between the first CircleT object with the others
    @staticmethod
    def sumFx(f):
        def __Fx__(f, ci, cj):
            if not (ci.connection(cj).len() == 0):
                return ci.force(f)(cj) * ((ci.cen().xcrd() - cj.cen().xcrd())/ci.connection(cj).len())
            else:
                return 0
        sumForces = 0
        if len(Deq.s) == 0:
            raise EMPTY("The deque is empty. There are no circles to take the sum of.")
        else:
            for i in Deq.s[1:]:
                sumForces = sumForces + __Fx__(f,i,Deq.s[0])
            return sumForces


## @brief Function calculates total area of all CircleT objects in deque
#  @details If deque is empty, then an exception is raised.
#  @return Returns total sum of all areas of CircleT objects in deque
    @staticmethod
    def totalArea():
        if len(Deq.s) > 0:
            totalSum = 0
            for i in Deq.s:
                totalSum = i.area() + totalSum
            return totalSum
        else:
            raise EMPTY("The deque is empty. No area to calculate and find total of.")

## @brief Function calculates average radius of all CircleT objects in deque
#  @details If deque is empty, then an exception is raised.
#  @return Returns average radius of all radii of CircleT objects in deque
    @staticmethod
    def averageRadius():
        if len(Deq.s) > 0:
            sumRadii = 0
            for i in Deq.s:
                sumRadii = i.rad() + sumRadii
            return sumRadii/float(len(Deq.s))
        else:
            raise EMPTY ("The deque is empty. No radii to calculate and find average of.")


## @brief This is a class for when an exception is needed to be raised for when
#  the deque is full and a function is called to perform an operation on it
class FULL(Exception):
  def __init__(self, value):
    self.value = value
  def __str__(self):
    return str(self.value)

## @brief This is a class for when an exception is needed to be raised for when
#  the deque is empty and a function is called to perform an operation on it
class EMPTY(Exception):
  def __init__(self, ivalue):
    self.ivalue = ivalue
  def __str__(self):
    return str(self.ivalue)
```

This code can also be found at the following link: Link

G Code for testCirclesDeque.py

```
## @file testCirclesDeque.py
#  @title testCirclesDeque
#  @author Meet Patel, patelm16
#  @date 2/27/2017

import unittest
from circleADT import *
from deque import *
from pointADT import *
from lineADT import *

## @brief This class uses PyUnit Unittesting to test methods from the following
#  modules: pointADT.py, lineADt.py. circleADT.py, and deque.py
class testCirclesDeque(unittest.TestCase):

## @brief Initializes main objects and functions to be used in the testing
#  at the start of each test
    def setUp(self):
        self.p1 = PointT(2.0, 3.0)
        self.p2 = PointT(5.0,4.0)
        self.line1 = LineT(self.p1, self.p2)
        self.p5 = PointT(6.0, 9.0)
        self.c1 = CircleT(self.p5, 3.0)
        self.c2 = CircleT(self.p1, 8.0)
        self.c3 = CircleT(self.p2, 11.0)
        self.f = lambda r: 6.672*r**(-3)
        Deq.init()
        self.deq = Deq

## @brief Removes all values at the end of each test
    def tearDown(self):
        self.p1 = None
        self.p2 = None
        self.line1 = None
        self.p5 = None
        self.c1 = None
        self.c2 = None
        self.c3 = None
        self.f = None
        self.deq = None

## @brief tests the xcrd method from pointADT
    def test_xcrd_correct(self):
        self.assertTrue(self.p1.xcrd() == 2.0)

## @brief tests the ycrd method from pointADT
    def test_ycrd_correct(self):
        self.assertTrue(self.p1.ycrd() == 3.0)

## @brief tests the dist method from pointADT
    def test_dist_correct(self):
        self.assertTrue(self.p1.dist(self.p2) == 3.1622776601683795)

## @brief tests the rot method for positive inputs from pointADT
    def test_rot_PositiveInput_correct(self):
        self.p1.rot(3.14)
        self.assertTrue(self.p1.xcrd() == -2.0047754222045393 and self.p1.ycrd() ==
            -3.0031891066056935)

## @brief tests the rot method for zero inputs from pointADT
    def test_rot_ZeroInput_correct(self):
        self.p1.rot(0)
        self.assertTrue(self.p1.xcrd() == 2.0 and self.p1.ycrd() == 3.0)

## @brief tests the rot method for negative inputs from pointADT
    def test_rot_NegativeInput_correct(self):
        self.p1.rot(-1.57)
        self.assertTrue(self.p1.xcrd() == 3.0015917022169702 and self.p1.ycrd() == -2.9992017703755964)

## @brief tests the rot method for inputs greater than pi from pointADT
    def test_rot_GreaterThanPiInput_correct(self):
        self.p1.rot(4.0)
        self.assertTrue(self.p1.xcrd() == 0.9631202441965605 and self.p1.ycrd() == -2.689822666680374)

## @brief tests the beg method from lineADT
    def test_beg_correct(self):
        self.assertTrue(self.line1.beg().xcrd() == 2.0 and self.line1.beg().ycrd() == 3.0)

    ## @brief tests the end method from lineADT
    def test_end_correct(self):
        self.assertTrue(self.line1.end().xcrd() == 5.0 and self.line1.end().ycrd() == 4.0)

    ## @brief tests the len method from lineADT
    def test_len_correct(self):
        self.assertTrue(self.line1.len() == 3.1622776601683795)

    ## @brief tests the mdpt method from lineADT with differerent inputs (length not equal to one)
    def test_mdpt_correct(self):
        self.assertTrue(self.line1.mdpt().xcrd() == 3.5 and self.line1.mdpt().ycrd() == 3.5)

    ## @brief tests the mdpt method from lineADT with a length of zero between points
    def test_mdpt_LengthZero_correct(self):
        p3 = PointT(8.0,8.0)
        p4 = PointT(8.0,8.0)
        line2 = LineT(p3,p4)
        self.assertTrue(line2.mdpt().xcrd() == 8.0 and line2.mdpt().ycrd() == 8.0)

    ## @brief tests the rot method from lineADT
    def test_rot_forLine_correct(self):
       self.line1.beg().rot(-0.6)
       self.line1.end().rot(2.3)
       self.assertTrue(self.line1.beg().xcrd() == 3.344598650004463 and self.line1.beg().ycrd()
           ==0.5875043904768189 and self.line1.end().xcrd() == -6.314200955106001  and
           self.line1.end().ycrd() == -7.373636648073067)

    ## @brief tests the cen method from circleADT
    def test_cen_correct(self):
        self.assertTrue(self.c1.cen().xcrd() == 6.0 and self.c1.cen().ycrd() == 9.0)

    ## @brief tests the rad method from circleADT
    def test_rad_correct(self):
        self.assertTrue(self.c1.rad() == 3.0)

    ## @brief tests the area method from circleADT
    def test_area_correct(self):
        self.assertTrue(self.c1.area() == 28.274333882308138)

    ## @brief tests the intersect method from circleADT
    def test_intersect_correct(self):
        self.assertTrue(self.c1.intersect(self.c2))

    ## @brief tests the connection method from circleADT
    def test_connection_correct(self):
        self.assertTrue(self.c1.connection(self.c2).beg().xcrd() == 6 and
            self.c1.connection(self.c2).beg().ycrd() == 9 and
            self.c1.connection(self.c2).end().xcrd() == 2 and
            self.c1.connection(self.c2).end().ycrd() == 3)

    ## @brief tests the force method from circleADT
    def test_force_correct(self):
        self.assertTrue(self.c1.force(self.f)(self.c2) == 101.15171511042486)

    ## @brief tests the pushBack method from deque
    def test_pushBack_correct(self):
        self.deq.pushBack(self.c1)
        self.assertTrue(self.deq.s[0].cen().xcrd() == 6.0 and self.deq.s[0].cen().ycrd() == 9.0 and
            self.deq.s[0].rad() == 3.0)

    ## @brief tests the pushFront method from deque
    def test_pushFront_correct(self):
        self.deq.pushBack(self.c1)
        self.deq.pushFront(self.c2)
        self.assertTrue(self.deq.s[0].cen().xcrd() == 2.0 and self.deq.s[0].cen().ycrd() == 3.0 and
            self.deq.s[0].rad() == 8.0)

    ## @brief tests the pushFront method from deque with a full deque exception
    def test_pushFrontFullException_correct(self):
        for i in range(20):
            self.deq.pushFront(self.c1)
        self.assertRaises(FULL, self.deq.pushFront, self.c2)

    ## @brief tests the popFront method from deque with an empty deque exception
    def test_popFrontEmptyException_correct(self):
        self.assertRaises(EMPTY, self.deq.popFront)

    ## @brief tests the popBack method from deque
    def test_popBack_correct(self):
        self.deq.pushBack(self.c1)
        self.deq.pushBack(self.c2)
        self.deq.popBack()
        self.assertTrue(self.deq.s[0].cen().xcrd() == 6.0 and self.deq.s[0].cen().ycrd() == 9.0 and
            self.deq.s[0].rad() == 3.0)

## @brief tests the popFront method from deque
    def test_popFront_correct(self):
        self.deq.pushBack(self.c1)
        self.deq.pushBack(self.c2)
        self.deq.popFront()
        self.assertTrue(self.deq.s[0].cen().xcrd() == 2.0 and self.deq.s[0].cen().ycrd() == 3.0 and
            self.deq.s[0].rad() == 8.0)

## @brief tests the back method from deque
    def test_back_correct(self):
        self.deq.pushBack(self.c1)
        self.deq.pushBack(self.c2)
        self.assertTrue(self.deq.back().cen().xcrd() == 2.0 and self.deq.back().cen().ycrd() == 3.0
            and self.deq.back().rad() == 8.0)

## @brief tests the front method from deque
    def test_front_correct(self):
        self.deq.pushBack(self.c1)
        self.deq.pushBack(self.c2)
        self.assertTrue(self.deq.front().cen().xcrd() == 6.0 and self.deq.front().cen().ycrd() == 9.0
            and self.deq.front().rad() == 3.0)

## @brief tests the size method from deque
    def test_size_correct(self):
        self.deq.pushBack(self.c1)
        self.deq.pushBack(self.c2)
        self.deq.pushBack(self.c3)
        self.assertTrue(self.deq.size() == 3)

## @brief tests the disjoint method from deque with three different circles
    def test_disjoint_correct(self):
        self.deq.pushBack(self.c1)
        self.deq.pushBack(self.c2)
        self.deq.pushBack(self.c3)
        self.assertFalse(self.deq.disjoint())

## @brief tests the disjoint method from deque with one circle
    def test_disjoint_oneCircle_correct(self):
        self.deq.pushBack(self.c1)
        self.assertTrue(self.deq.disjoint())

## @brief tests the pushBack method from deque with an empty deque
    def test_disjoint_emptyException_correct(self):
        self.assertRaises(EMPTY, self.deq.disjoint)

## @brief tests the sumFx method from deque
    def test_sumFx_correct(self):
        self.deq.pushFront(self.c1)
        self.deq.pushFront(self.c2)
        self.deq.pushFront(self.c3)
        self.assertTrue(self.deq.sumFx(self.f) == -15192.191286940368 )

## @brief tests the totalArea method from deque
    def test_totalArea_correct(self):
        self.deq.pushBack(self.c1)
        self.deq.pushBack(self.c2)
        self.deq.pushBack(self.c3)
        self.assertTrue(self.deq.totalArea() == 609.4689747964198)

## @brief tests the averageRadius method from deque
    def test_averageRadius_correct(self):
        self.deq.pushBack(self.c1)
        self.deq.pushBack(self.c2)
        self.deq.pushBack(self.c3)
        self.assertTrue(self.deq.averageRadius() == 7.333333333333333)

print "All test cases from all modules have passed."

if __name__ == '__main__':
    unittest.main()
```

This code can also be found at the following link: Link

H Code for Makefile

```
PY = python
PYFLAGS =
DOC = doxygen
DOCFLAGS =
DOCCONFIG = doxConfig

SRC = ./src/testCircleDeque.py

.FORCE: test doc clean

test:
        $(PY) $(PYFLAGS) $(SRC)

doc:
        $(DOC) $(DOCFLAGS) $(DOCCONFIG)
        cd latex && make

clean:
        rm -rf html
        rm -rf latex
```

This code can also be found at the following link: Link

I Partner's circleADT.py Code

```
## @file circleADT.py
#  @title circleADT
#  @author Jenny Feng Chen (fengchej)
#  @date 2/19/2017

from pointADT import*
from lineADT import*
import math

## @brief This class represents a circle.
#  @details This class represent a circle with a center represented by PointT and a radius.
class CircleT(object):

    ## @brief This is a constructor for CircleT.
    #  @details This is a constructor for CircleT that takes two parameters and assigns one to center
         of circle and the radius of circle.
    #  @param cin is an instance of PointT object.
    #  @param rin is a positive real number.
    def __init__(self, cin, rin):
        self.c = cin
        self.r = float(rin)

    ## @brief This method returns the center point of the circle.
    #  @return the center of the circle.
    def cen(self):
        return self.c

    ## @brief This method returns the radius of the circle.
    #  @return the radius of the circle.
    def rad(self):
        return self.r

    ## @brief This method determines the area of the circle.
    #  @return the area of the circle.
    def area(self):
        return self.r**2 * math.pi

    ## @brief This method check whether two circles intersect.
    #  @details This method treat circles as filled objects. The set of points in each circle
    #  includes the boundary (closed sets).
    #  @param ci is an instance of the CircleT.
    #  @return true if the circles intersect; false if not.
    def intersect(self,ci):
        intersect = (self.cen().xcrd()-ci.cen().xcrd())**2 + (self.cen().ycrd()-ci.cen().ycrd())**2 <=
            (self.rad() + ci.rad())**2
        return intersect

    ## @brief This method creates a new line between the center of two circles.
    #  @param ci is an instance of the circle.
    #  @return line which is a new instance of the LineT object.
    def connection(self, ci):
        return LineT(self.cen(), ci.cen())

    ## @brief This method calculates the gravitational force between two circles.
    #  @details This method takes a function and return a function. It is basically calculates the
         gravitational force between two circles using the universal gravitational constant.
    #  @param f is a function that takes a real number an returns a real number.
    #  @return f1 a function that takes an instance of CircleT and returns a real number.
    def force(self, f):
        f1 = lambda x: x.area()*self.area()* f(self.connection(x).len())
        return f1
```

This code can also be found at the following link: Link