CS 2ME3 Assignment 2 Report

Name: Polly yao   MacId: yaos5

February 27, 2017

A Code for pointADT.py

```
## @file pointADT.py
#  @title pointADT
#  @author Polly Yao
#  @date 2/19/2017

## @brief This class represents a point
#  @details This class represents a point as x and y define the x and y corrdinates of a point


import math

## @brief Constructor for PointT
#  @param x coordinate of a point
#  @param y coordinate of a point
class PointT:
    def __init__(self, x, y):
        self.x = x
        self.y = y

## @brief This function receives x from the constructor
#  @return x coordinate of the circle
    def xcrd(self):
        return (self.x)

## @brief This function receives y from the constructor
#  @return y coordinate of the circle
    def ycrd(self):
        return (self.y)

## @brief This function calculates the distance of two points
#  @param p is another instance of point
#  @return distance between two point
    def dist(self, p):
        A = (self.x - p.x)**2
        B = (self.y - p.y)**2
        result = math.sqrt(A+B)
        return result

## @brief This function rotates the x and y coordiantes
#  @param phi a radian

    def rot(self, phi):
        cx = (math.cos(phi)*self.x) + (-math.sin(phi)*self.y)
        cy = (math.sin(phi)*self.x) + (math.cos(phi)*self.y)
        self.x = cx
        self.y = cy
```

B Code for lineADT.py

```
## @file line.py
#  @title lineADT
#  @author Polly Yao
#  @date 2/19/2017

## @brief This class represents a line
#  @details This class represents a point as p1 and p2 define two point of a line


import math
import pointADT


## @brief Constructor for LineT
#  @param p1 a point at the front of the line
#  @param p2 a point at the back of the line
class LineT:
    def __init__(self, p1, p2):
        self.p1 = p1
        self.p2 = p2

## @brief This function receives p1 from the constructor
#  @return the beginning point of the line
    def beg(self):

        return (self.p1)

## @brief This function receives p2 from the constructor
#  @return the back point of the line
    def end(self):
        return (self.p2)

## @brief This function calculates the length of the line
#  @return the length
    def len(self):
        result = (self.p1).dist(self.p2)
        return result


## @brief This function calculates the midpoint of the line
#  @return the a midpoint(PointT)
    def mdpt(self):

        a0 = (self.p1.xcrd())
        a1 = (self.p2.xcrd())
        af = (a0 + a1)/2

        b0 = (self.p1.ycrd())
        b1 = (self.p2.ycrd())
        bf = (b0 + b1)/2

        res = pointADT.PointT(af, bf)

        return res
## @brief This function rotates front point and back point of the line
#  @param phi a radian
    def rot(self, phi):
        (self.p1).rot(phi)
        (self.p2).rot(phi)
```

C Code for circleADT.py

```
## @file CircleADT.py
#  @title CircleADT
#  @author Polly Yao
#  @date 2/19/2017

## @brief This class represents a circle
#  @details This class represents a circle as cin define the centre of the circle
#   and rin defines the radius of the circle

import math

import pointADT

import lineADT

## @brief Constructor for CircleT
#  @param cin as the coordinate for the centre of the circle
#  @param rin as the radius of the circle
class CircleT:
    def __init__(self, cin, rin):
        self.cin = cin

        self.rin = rin

## @brief This function receives cin from the constructor
#  @return centre(PointT) of the circle
    def cen(self):
        return (self.cin)

## @brief This function receives the radius from the constructor
#  @return radius of the circle
    def rad(self):
        return (self.rin)

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

## @brief This function checks if two circles intersect
#  @details This function takes in another circle ci
#  @details The two  circles intersect if the sum of two radius is greater and equal to the sum of
     differences between x and y coordinate of two circles
#  @param ci is the second instance of CircleT
#  @return True if two circles intersect, False if they do not
    def intersect(self, ci):
        distance = math.sqrt((ci.cin.y - self.cin.y)**2 + (ci.cin.x - self.cin.x)**2)
        radii = self.rin + ci.rin

        if (radii >= distance):
            return True

        else:
            return False

## @brief This function forms a line between two circles
#  @details This function takes in another circle ci
#  @return the line
    def connection(self, ci):
        out = lineADT.LineT(self.cin, ci.cen())
        return out

## @brief This function is built for the force function
#  @details This function takes in another circle x
#  @return a answer that is used by force function

    def counter(self, x):
        r = self.connection(x).len()
        b = (6.672*(10**-11))/r**2
        area = x.area()
        result = b * area
        return result

## @brief This function fcalculates the force between two circles
#  @details This function takes in a function counter
#  @return the force between two circles
    def force(self, method):
        return (self.area() * method)
```

D Code for deque.py

```
## @file deque.py
#  @title deque
#  @author Polly Yao
#  @date 2/19/2017

## @brief This class represents a queue



import circleADT
import pointADT
import lineADT
import math

class Deque:

  MAX_SIZE = 20

  d = []

## @brief Constructor for the queue
  @staticmethod
  def Deq_init():
    Deque.d = []

## @brief This function push circles onto the back of the queue
#  @param c takes in circle
  @staticmethod
  def Deq_pushBack(c):
    d = Deque.d
    size = len(d)
    if size == Deque.MAX_SIZE:
      raise Full("Maximum size exceeded")

    d.append(c)


    Deque.d = d


## @brief This function push circles onto the front of the queue
#  @param c takes in circle
  @staticmethod
  def Deq_pushFront(c):
    d = Deque.d
    size = len(d)
    if size == Deque.MAX_SIZE:
      raise Full("Maximum size exceeded")

    d.insert(0, c)


    Deque.d = d


## @brief This function remove circles from the back of the queue

  @staticmethod
  def Deq_popBack():
    d = Deque.d
    size = len(d)
    if size == 0:
      raise Empty("Queue is empty")

    d.pop()


    Deque.d = d

## @brief This function remove circles from the front of the queue
  @staticmethod
  def Deq_popFront():
    d = Deque.d
    size = len(d)
    if size == 0:
            raise Empty("Queue is empty")

        d.pop(0)


        Deque.d = d


## @brief This function returns the last circle in the queue
#  @return the last circle in the queue
    @staticmethod
    def Deq_back():
        size = len(Deque.d)
        if size == 0:
            raise Empty("Queue is empty")


        return Deque.d[size -1]

## @brief This function returns the first circle in the queue
#  @return the first circle in the queue
    @staticmethod
    def Deq_front():
        size = len(Deque.d)
        if size == 0:
            raise Empty("Queue is empty")


        return Deque.d[0]

## @brief This function measure the length the queue
#  @return the the length of the queue
    @staticmethod
    def Deq_size():
        size = len(Deque.d)
        return size


## @brief This function if there are any circles intersect each other in the queue
#  @return True if there is at least one circle intersect with another, return False otherwise
    @staticmethod
    def Deq_disjoint():
        size = len(Deque.d)

        if size == 0:
            raise Empty("Queue is empty")
        array = []
        for i in range(len(Deque.d)):
            for j in range(i+1, len(Deque.d)):
               distance = math.sqrt((Deque.d[i].cen().y - Deque.d[j].cen().y)**2 + (Deque.d[i].cen().x -
                    Deque.d[j].cen().x)**2)
               radii = Deque.d[i].rad() + Deque.d[j].rad()

               if (radii >= distance):
                   array.append(True)
               else:
                   array.append(False)

        cou = array.count(True)
        if cou > 0:
            return True
        else:
            return False



## @brief This function compute the area for each of the circle in the queue, and add all the area
     together
#  @return total area
    @staticmethod
    def Deq_totalArea():
        size = len(Deque.d)

        if size == 0:
            raise Empty("Queue is empty")

        array = []

        for element in range(len(Deque.d)):
            area = (math.pi)*((Deque.d[element].rad())**2)
            array.append(area)

        a = sum(array)
        return a



## @brief This function compute the average radius of the circles in the queue
#  @return avaerage radius
    @staticmethod
    def Deq_averageRadius():
        size = len(Deque.d)

        if size == 0:
            raise Empty("Queue is empty")
        array = []

        for element in range(len(Deque.d)):

            array.append(Deque.d[element].rad())

        a = sum(array)

        b = len(array)

        average = float(a/b)
        return average
```

E Code for testCircleDeque.py

```
## @file testCircleDeque.py
#  @title testCircleDeque
#  @author Polly Yao
#  @date 2/19/2017


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


class pointTests(unittest.TestCase):

    def setUp(self):
        self.point1 = PointT(1, 2)
        self.point2 = PointT(4, 3)


    def tearDown(self):
        self.point1 = None
        self.point2 = None

    def test_xcrd_are_equal(self):
        self.assertTrue(self.point1.xcrd() == 1)

    def test_ycrd_are_equal(self):
        self.assertTrue(self.point1.ycrd() == 2)

    def testdist(self):
        self.assertAlmostEqual(self.point1.dist(self.point2), 3.1622776601683795,  None, None, 0.1)

    def testrot(self):
        (self.point1).rot(30)

        self.assertAlmostEqual(self.point1.xcrd(), 2.130314698073308,  None, None, 0.1)
        self.assertAlmostEqual(self.point1.ycrd(), -0.6795287243176937,  None, None, 0.1)

        (self.point1).rot(-30)

        self.assertAlmostEqual(self.point1.xcrd(), 1.0,  None, None, 0.1)
        self.assertAlmostEqual(self.point1.ycrd(), 2.0,  None, None, 0.1)

        (self.point1).rot(0)

        self.assertAlmostEqual(self.point1.xcrd(), 1.0,  None, None, 0.1)
        self.assertAlmostEqual(self.point1.ycrd(), 2.0,  None, None, 0.1)

        (self.point1).rot(60)

        self.assertAlmostEqual(self.point1.xcrd(), -0.34279173821072284,  None, None, 0.1)
        self.assertAlmostEqual(self.point1.ycrd(), -2.20963658193253,  None, None, 0.1)



class lineTests(unittest.TestCase):
    def setUp(self):
        point1 = PointT(1, 2)
        point2 = PointT(4, 3)
        point3= PointT(1, 2)

        self.line = LineT(point1, point2)
        self.line2 = LineT(point1, point3)

    def tearDown(self):
        self.line = None

    def testbeg(self):
        self.assertTrue(self.line.beg().x == 1)

    def testend(self):
        self.assertTrue(self.line.end().x == 4)

    def testline(self):
        self.assertAlmostEqual(self.line.len(), 3.1622776601683795,  None, None, 0.1)

    def testmdpt(self):
        self.assertAlmostEqual(self.line.mdpt().x, 2,  None, None, 0.1)
        self.assertAlmostEqual(self.line.mdpt().y, 2,  None, None, 0.1)
        self.assertAlmostEqual(self.line2.mdpt().x, 1,  None, None, 0.1)
        self.assertAlmostEqual(self.line2.mdpt().y, 2,  None, None, 0.1)

    def testrot(self):
        (self.line).rot(60)

        self.assertAlmostEqual(self.line.beg().x, -0.34279173821072295,  None, None, 0.1)
        self.assertAlmostEqual(self.line.beg().y, -2.2096365819325294,  None, None, 0.1)
        self.assertAlmostEqual(self.line.end().x, -2.895220058353975,  None, None, 0.1)
        self.assertAlmostEqual(self.line.end().y, -4.076481425654336,  None, None, 0.1)


class circleTests(unittest.TestCase):
    def setUp(self):
        point1 = PointT(1, 2)
        point2 = PointT(4, 3)

        self.circle1 = CircleT(point1, 3)
        self.circle2 = CircleT(point2, 2)

    def tearDown(self):
        self.circle1 = None
        self.circle2 = None

    def testcen(self):
        self.assertTrue(self.circle1.cen().x == 1)
        self.assertTrue(self.circle1.cen().y == 2)

    def testrad(self):
        self.assertTrue(self.circle1.rad() == 3)

    def testarea(self):
        self.assertAlmostEqual(self.circle1.area(), 28.274333882308138,  None, None, 0.1)

    def testintersect(self):
        self.assertTrue(self.circle1.intersect(self.circle2) == True)

    def testconnection(self):
        (self.circle1).connection(self.circle2)

        self.assertAlmostEqual((self.circle1).connection(self.circle2).beg().x, 1,  None, None, 0.1)

        self.assertAlmostEqual((self.circle1).connection(self.circle2).end().y, 3,  None, None, 0.1)

    def testforce(self):
        self.assertAlmostEqual(self.circle1.force((self.circle1.counter(self.circle2))),
            2.3706000203064544e-09,  None, None, 0.1)

class dequeTests(unittest.TestCase):
    def setUp(self):
        point1 = PointT(1, 2)
        point2 = PointT(3, 4)
        point3 = PointT(10, 7)
        point4 = PointT(8, 2)


        self.circle1 = CircleT(point1, 1)
        self.circle2 = CircleT(point2, 2)
        self.circle3 = CircleT(point3, 1)
        self.circle4 = CircleT(point4, 1)

        Deque.Deq_pushBack(self.circle1)
        Deque.Deq_pushBack(self.circle2)
        Deque.Deq_pushBack(self.circle3)
        Deque.Deq_pushBack(self.circle4)

    def tearDown(self):
        circle1 = None
        circle2 = None

    def testDeq_back(self):


        self.assertTrue(Deque.Deq_back().rad() == 1)
    def testDeq_front(self):


        self.assertTrue(Deque.Deq_back().rad() == 1)


    def testDeq_disjoint(self):
        Deque.Deq_disjoint()

        #self.assertTrue(Deque.Deq_disjoint() == True)

    def testDeq_totalArea(self):

        self.assertAlmostEqual(Deque.Deq_totalArea(), 109.9557428756428,  None, None, 0.1)

    def testDeq_averageRadius(self):

        self.assertAlmostEqual(Deque.Deq_averageRadius(), 1.0,  None, None, 0.1)



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

F Makefile

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

SRC = src/testCircleDeque.py

.PHONY: all prog doc clean

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

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

all: test doc

clean:
	rm -rf html
	rm -rf latex
	rm -rf circleADT.pyc
	rm -rf deque.pyc
	rm -rf lineADT.pyc
	rm -rf pointADT.pyc
```

G Partner's circleADT Code

```
#Michael Balas
#400023244
import pointADT
import lineADT
import math
## @file circleADT.py
#  @title CircleADT
#  @author Michael Balas
#  @date 2/2/2017

## @brief This class represents a circle ADT.
#  @details This class represents a circle ADT, with point cin (x, y)
#  defining the centre of the circle, and r defining its radius.
class CircleT(object):

        ## @brief Constructor for CircleT
        #  @details Constructor accepts one point and a number (radius)
        #  to construct a circle.
        #  @param cin is a point (the centre of the circle).
        #  @param rin is any real number (represents the radius of the circle).
        def __init__(self, cin, rin):
                self.c = cin
                self.r = rin

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

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

        ## @brief Calculates the area of the circle
        #  @return The area of the circle
        def area(self):
                return math.pi*(self.r)**2

        ## @brief Determines whether the circle intersects another circle
        #  @details This function treats circles as filled objects: circles completely
        #  inside other circles are considered as intersecting, even though
        #  their edges do not cross.  The set of points in each circle
        #  includes the boundary (closed sets).
        #  @param ci Circle to test intersection with
        #  @return Returns true if the circles intersect; false if not
        def intersect(self, ci):
                xDist = self.c.xc - ci.c.xcrd()
                yDist = self.c.yc - ci.c.ycrd()
                centerDist = math.sqrt(xDist ** 2 + yDist ** 2)
                rSum = self.r + ci.rad()
                return rSum >= centerDist

        ## @brief Creates a line between the centre of two circles
        #  @details This function constructs a line beginning at the centre of the
        #  first circle, and ending at the centre of the other circle.
        #  @param ci Circle to create connection with
        #  @return Returns a new LineT that connects the centre of both circles
        def connection(self, ci):
                return lineADT.LineT(self.c, ci.cen())

        ## @brief Determines the force between two circles given some parameterized
        #  gravitational law
        #  @details This functions calculates the force between two circles of unit
        #  thickness with a density of 1 (i.e. the mass is equal to the area). Any
        #  expression can be substituted for the gravitational law, f(r), or G/(r**2).
        #  @param f Function that parameterizes the gravitational law. Takes the distance
        #  between the centre of the circles and can apply expressions to it (e.g. multiply
        #  the universal gravitation constant, G, by the inverse of the squared distance between
        #  the circles).
        #  @return Returns the force between two circles
        def force(self, f):
                return lambda x: self.area() * x.area() * f(self.connection(x).len())
```

Testing results of my files

The results of testing my files came out successfully. Twenty test cases all completed in 0.067 seconds. However, I realized I have missed to include some important test cases. Therefore, I decided to add some extra test cases just to see if my code really functions properly. First, I have added a test case for negative rotation, I can see my code functions perfectly since if I rotate the point with the same radian both in positive and negative direction, and the x and y coordinate will evetually go back to its original point. Second, I have added a test case for zero radians, and it also came out successfully, the point did not rotate at all. Third, i have added a test case for midpoint of a line of length zero, and the test was succusseful, the midpoint of a line of length zero is the point itself. Overall

Testing results of my files combined my partner's files

The result of running my partner's files completed in 0.134 seconds. total of 20 test cases with 18 passed and 2 failed. testForce ans test intersect are two cases that failed. After seeing my partner's Force method, I realized my force method is not appropriate, I should not create a counter method myself, and I should use the functional language such as lambda. In the intersect method, I should have named my x and y componet as xc and yc as specified in the beginning of the pointADT module.

Discussion on the test results

As stated previously, my implementation for force and intersect method is different from my partner's. I think implement the correct specification and name is very important, In my intersect method, I missed read the xc and yc coordinate which resulted in a non standardized module. The formal mudule interface specification in this assignment makes the assignment more standardize, not too much assumption need to stated. The advantages of using pyunit is that it it could give the developer a earlier notice of what went wrong in the code instead of leading to a disaster in the end. The pyunit method is also very simple to understand and gives the developer a time about the efficeiency and perfomance of the code.