Assignment 2 Solution

Mohamed Bengezi bengezim

February 28, 2017

The purpose of this software design exercise is to write a Python program that creates, uses, and tests an ADT that stores circles. The program consist of the following files: `pointADT.py`, `lineADT.py`, `circleADT.py`, `deque.py`, `testCircle.py` and `Makefile`, as shown in Appendix

Testing of the Original Program

The results of testing my files, were all successful. Most of the simple functions had simple test cases, but more complicated ones, such as rot(), had specialized ones. The following were specific test cases that were notable:

- In rot() for PointT, I had four test cases. One was for a negative theta, such as -$\pi/2$, one was for a positive theta, such as $\pi/2$, one was for a theta of 0, and lastly one was for a theta above pi, such as 2π. The rot function proved successful in all four test cases
- For the functions that returned point objects, such as beg, I tested the xcoord and ycoord of the resulting point to make sure it matched the expected output.
- I felt no need to test the LineT rot() function more than once, seeing as it relies solely on the PointT rot(), which I tested heavily already.
- My assumption for the intersect() method was that p is a point that is the result of the averages of the x's and y's of the centre's of both circles. It would then see if p was in both circles. This assumption was incorrect. This was realized after submission

```
C:\Users\Mohamed\bengezim\A2\src>python -m unittest testCircleDeque
.............................
----------------------------------------------------------------------
Ran 29 tests in 0.000s

OK
```

Figure 1: Results of testing My files

```
C:\Users\Mohamed\bengezim\A2\src\srcPartner>python -m unittest testCircleDeque
....E.......................
======================================================================
ERROR: test_circle_insideCircle (testCircleDeque.CircleDeque)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "testCircleDeque.py", line 105, in test_circle_insideCircle
    self.assertTrue(cADT.insideCircle(b,a) == True)
AttributeError: 'module' object has no attribute 'insideCircle'

----------------------------------------------------------------------
Ran 29 tests in 0.000s

FAILED (errors=1)
```

Figure 2: Results of testing Partner files w/o my insideCircle

Testing of the Partner Files

As seen in the next figure, my partner's files fully passed the tests. Although they did not create an insideCircle method for CircleADT, so without it the test for insideCircle threw an AttributeError. I had to put mine in there temporarily. The main reason for this is that they have made different assumptions than me

```
C:\Users\Mohamed\bengezim\A2\src\srcPartner>python -m unittest testCircleDeque
.............................
----------------------------------------------------------------------
Ran 29 tests in 0.016s

OK
```

Figure 3: Results of testing Partner files w/ my insideCircle

Discussion

0.1 Summary

In general, all test results were successful, although those for intersect were misleading, as will be discussed next. In general, this assignment taught me the knowledge needed to successfully read an MIS, as well as implement discrete methematics and convert this logic into code. It also taught me to use PyUnit for testing, which makes the testing process much more efficient and manageable. I strongly believe this assignment was layed out and implemented much better than the previous one. it was much more structured, and therefore allowed me to implement with ease.

0.2 Issues with my code

As discussed earlier, my intersect() method was implemented incorrectly. The reason for this is that my assumptions were incorrect. Although my test cases did not make it fail, I realized it after submission, while I was playing around with the code. I made this assumption because the assignment MIS specified that a point p must reside in both circles for them to intersect, although I was not sure how to find p, and this seemed like a logical method. Other than intersect, I believe there are no other issues in the code

0.3 Issues with partners code

As far as testing goes, all my tests passed for my partners circleADT.py. This leads me to believe that there are no issues with respect to my and my partners code being implemented together

0.4 Module Specification

The specification for this assignment was far more helpful and specific than Assignment 1. It detailed exactly how each function should be implemented, and which math formulas to use. This allowed for a more independent design when it came to testing other students files. All files should be implemented the same way, so testing someone else's files should still pass. Assignment 1 was very general, and therefore lead to more failures due to people implementing functions incorrectly. Using PyUnit for testing is extremely convienient, as it provides formatting that one would have to create manually otherwise, as well as has many options for testing. I find this much more efficient than the testing for Assignment 1, seeing as it was all done manually, and all the formatting was done in the print statements, which is highly unprofessional.

0.5 Specification for totalArea() and averageRadius()

Deq_totalArea():

- output

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

Deq_averageRadius():

- output

$$out := \frac{+(i : \mathbb{N} | i \in [0..|s| - 1] : s[i].\text{rad}())}{|s|}$$

0.6 Circle Module Interface Critique

- Consistent: For the most part, the module was consistent. Although, the naming conventions differed from time to time. At the beginning, the names were shortened, such as cen(), but towards the end they were long, such as intersect() or connection(). As for the ordering of parameters, they were uniformly listed throughout. Lastly, the exception handling was all consistent

- Essential: The module was most definitely essential. There was no more than one way to access each function, and no unnecessary featuress were added.

- General: The module was fairly general. Although it depends on the user implementing pointADT and lineADT, which reduces its generality and makes it less useful for others. If someone wanted to implement circleADT solely, they would not be able to do so.

- Minimal: Minimality was clearly present in the circle module. No access routine had two separate functions or services to complete.

- Opaque: Lastly, the module was moderately opaque. Each service was contained in a separate function, hiding it and allowing users to change the function without changing the entire module. Although it was a fairly simple module, so there was not much room or need for information hiding

0.7 disjoint() w/ One Circle

For a deque with one circle, it will have a size of 1. In the mathematical specification, the predicate to the intersect method will never pass. This is because the i and j go from 0 to 1-1 (0), which satisfy the first two conditions of the predicate, but the last condition

is that i $\neq$ j, which fails. So the intersect expression never occurs. In my code, the same thing would happen, but because it fails the if i $\neq$ j statement, the function would return True.

A Code for pointADT.py

```
##Mohamed Bengezi
##400021279

## @file pointADT.py
#  @author Mohamed Bengezi
#  @brief ADT for a point, for use in lineADT and circleADT.
#  @date Feb 19, 2017
import math

class PointT:
    ## @brief Constructor for pointT
    #  @details Creates a point with coordinates x and y
    #  @param x  x-coordinate of point
    #  @param y y-coordinate of poin
    def __init__(self,x,y):
        self.xc,self.yc = x,y
    ## @brief Returns the x-coordinate of the point
    #  @return the x-coordinate
    def xcrd(self):
        return self.xc
    ## @brief Returns the y-coordinate of the point
    #  @return the y-coordinate
    def ycrd(self):
        return self.yc
    ## @brief Returns the distance of the point from another point
    #  @return the distance
    def dist(self,p):
        return math.sqrt((self.xc-p.xcrd())**2+(self.yc-p.ycrd())**2)
    ## @brief Rotates the point by angle theta(radians)
    #  @details rotation occurs about the origin, ccw
    #  @param theta The angle of rotation
    def rot(self, theta):
        x = self.xcrd()
        y = self.ycrd()
        self.xc = math.cos(theta)*x - math.sin(theta)*y
        self.yc = math.sin(theta)*x + math.cos(theta)*y
```

B Code for lineADT.py

```
##Mohamed Bengezi
##400021279

## @file lineADT.py
#  @author Mohamed Bengezi
#  @brief ADT for a line, for use in circleADT.
#  @date Feb 19, 2017
import pointADT as pADT
import math

## @brief Local function for calculating the average
#  @param x1 First number
#  @param x2 Second Number
#  @return the average of x1 and x2
def avg(x1,x2):
    return ((x1 + x2)/2.0)
## @brief LineT class for the line ADT
class LineT:
    ## @brief Constructor for LineT
    #  @details Creates a line object
    #  @param p1 The starting point
    #  @param p2 The end point
    def __init__(self,p1,p2):
        self.b,self.e = p1, p2
    ## @brief Returns the beginning point object
    #  @return Beginning point
    def beg(self):
        return self.b
    ## @brief Returns the ending point object
    #  @return Ending point
    def end(self):
        return self.e
    ## @brief Returns the length of the line
    #  @details Uses the dist function in pointADT
    #  @return length of the line
    def len(self):
        return self.b.dist(self.e)
    ## @brief Returns the midpoint of the line
    #  @details Uses the avg function and calculates the avg of the x's and y's
    #  @return The midpoint of the line
    def mdpt(self):
        mdptx = avg(self.b.xcrd(),self.e.xcrd())
        mdpty = avg(self.b.ycrd(),self.e.ycrd())
        return pADT.PointT(mdptx,mdpty)
    ## @brief Rotates the line by angle theta(radians)
    #  @details Rotation occurs about the origin, ccw
    #  @param theta The angle of rotation
    def rot(self,theta):
        self.b.rot(theta)
        self.e.rot(theta)
```

C Code for circleADT.py

```
##Mohamed Bengezi
##400021279

## @file circleADT.py
#  @author Mohamed Bengezi
#  @brief ADT for a circle, for use in DequeCircleModule.
#  @date Feb 19, 2017
import pointADT as pADT
import lineADT as lADT
import math

## @brief Local function for determing whether a point is inside a circle
#  @param p the point
#  @param c the Circle
#  @return true(if point is inside the circle) or false
def insideCircle(p,c):
    return (p.dist(c.cen()) <= c.rad())
## @brief CircleT class for the circle ADT
class CircleT:
    ## @brief Constructor for CircleT
    #  @details Creates a Circle object
    #  @param cin The centre point
    #  @param rin The radius of the circle
    def __init__(self,cin,rin):
        self.c = cin
        self.r = rin
    ## @brief Returns the centre point of the circle
    #  @return Centre point
    def cen(self):
        return self.c
    ## @brief Returns the radius of the circle
    #  @return Radius
    def rad(self):
        return self.r
    ## @brief Returns the area of the circle using Pi*radius**2
    #  @return Area of the circle
    def area(self):
        return math.pi*self.r**2
    ## @brief Checks whether two circles intersect
    #  @details creates a new point thats in between the two circles, then checks whether the point is
        inside both. If it is, return True
    #  @param ci The second circle to compare
    #  @return True or False
    def intersect(self, ci):
        px = lADT.avg(ci.cen().xcrd(),self.cen().xcrd())
        py = lADT.avg(ci.cen().ycrd(),self.cen().ycrd())
        p = pADT.PointT(px,py)
        if ((insideCircle(p,ci) == True) and (insideCircle(p,self) == True)):
            return True
        return False
    ## @brief Checks whether two circles intersect
    #  @details creates a new line that connects the centre's of each circle's
    #  @param ci The second circle to connect
    #  @return The line created
    def connection(self,ci):
        return lADT.LineT(self.c,ci.cen())
    ## @brief Calculates the force between two circles
    #  @details Uses a lambda function to accomplish this
    #  @param f The gravitational law
    #  @return The line created
    def force(self, f):
        return lambda x: self.area()*x.area()*f(self.connection(x).len())
```

D Code for deque.py

```
##Mohamed Bengezi
##400021279

## @file deque.py
#  @author Mohamed Bengezi
#  @brief A deque class that uses circleADT objects
#  @date Feb 19, 2017

import circleADT as cADT
import math

## @brief A Deque class designed specifically for the circleADT module
class Deq:
    MAX_SIZE = 20
    s = []
    ## @brief A constructer for the deque
    #  @details Initializes the EMPTY deque
    @staticmethod
    def init():
        Deq.s = []
    ## @brief Adding a circleT object into the deque
    #  @details pushes it onto the end of the stack
    #  @param c The CircleT object
    #  @exception A FULL deque
    @staticmethod
    def pushBack(c):
        s = Deq.s
        size = len(Deq.s)
        if size == Deq.MAX_SIZE:
            raise FULL("Maximum size exceeded")
        s = s[0:size] + [c]
        Deq.s = s
    ## @brief Adding a circleT object into the deque
    #  @details pushes it onto the front of the stack
    #  @param c The CircleT object
    #  @exception A FULL deque
    @staticmethod
    def pushFront(c):
        s = Deq.s
        size = len(Deq.s)
        if size == Deq.MAX_SIZE:
            raise FULL("Maximum size exceeded")
        s =  [c] + s[0:size]
        Deq.s = s
    ## @brief Removing a circleT object from the deque
    #  @details pops the last object from the end of the stack
    #  @exception An EMPTY deque
    @staticmethod
    def popBack():
        s = Deq.s
        size = len(Deq.s)
        if size == 0:
            raise EMPTY("Nothing to pop")
        s = s[0:size-1]
        Deq.s = s
    ## @brief Removing a circleT object from the deque
    #  @details pops the first object from the front of the stack
    #  @exception An EMPTY deque
    @staticmethod
    def popFront():
        s = Deq.s
        size = len(Deq.s)
        if size == 0:
            raise FULL("Nothing to pop")
        s = s[1:size]
        Deq.s = s
    ## @brief Returns the last element on the deque
    #  @return The last CircleT object on the deque
    #  @exception An EMPTY deque
    @staticmethod
    def back():
        size = len(Deq.s)
        if size == 0:
            raise EMPTY("EMPTY deque")
        return Deq.s[len(Deq.s)-1]
    ## @brief Returns the first element on the deque
```

```
    #  @return The first CircleT object on the deque
    #  @exception An EMPTY deque
    @staticmethod
    def front():
        size = len(Deq.s)
        if size == 0:
            raise EMPTY("EMPTY deque")
        return Deq.s[0]
    ## @brief Returns the size of the deque
    #  @return size of the deque
    @staticmethod
    def size():
        return len(Deq.s)
    ## @brief Checks if none of the circles intersect each other
    #  @details Uses the intersect method from CircleT
    #  @return True(if none intersect) or False
    #  @exception An EMPTY deque
    @staticmethod
    def disjoint():
        size = len(Deq.s)
        if size == 0:
            raise EMPTY("EMPTY Deque")
        for i in range(0, size):
            for j in range(0, size):
                if i != j:
                    if ((Deq.s[i]).intersect(Deq.s[j])):
                        return False
        return True


    ## @brief Sums all the forces in the x plane
    #  @details Uses the local method Fx, and sums all the forces on the first circle with the other
        circles
    #  @param f The gravitational law
    #  @return The sum of all forces
    #  @exception An EMPTY deque
    @staticmethod
    def sumFx(f):
        size = len(Deq.s)
        t = 0
        if size == 0:
            raise EMPTY("EMPTY deque")
        for i in range(1,size):
            t += Fx(f,Deq.s[i],Deq.s[0])
        return t

    ## @brief A total area function, though the implementation is not required
    #  @details  It sums the areas of all the circle in the deque. The exceptions are coded as a
        precaution
    #  @exception An EMPTY deque
    @staticmethod
    def totalArea():
        size = len(Deq.s)
        if size == 0:
            raise EMPTY("EMPTY deque")
    ## @brief An average radius function, though the implementation is not required
    #  @details Gets the average radius. The exceptions are coded as a precaution
    #  @exception An EMPTY deque
    @staticmethod
    def averageRadius():
        size = len(Deq.s)
        if size == 0:
            raise EMPTY("EMPTY deque")

## @brief Local function for calculating the Force between two ciircles
#  @param f The gravitational law
#  @param ci The first circle
#  @param cj The second circle
#  @return The force between the circles
def Fx(f,ci,cj):
    xcomp(ci.force(f)(cj),ci,cj)

## @brief Local function used for determining the x-component of the force
#  @param F The force
#  @param ci The first circle
#  @param cj The second circle
#  @return The x component of the force between the circles
def xcomp(F,ci,cj):
    F*((ci.cen().xcrd()-cj.cen().xcrd())/ci.connection(cj).len())
## @brief The exception for a FULL deque
```

```
class FULL(Exception):
  def __init__(self, value):
    self.value = value
  def __str__(self):
    return str(self.value)
## @brief The exception for an EMPTY deque
class EMPTY(Exception):
  def __init__(self, ivalue):
    self.ivalue = ivalue
  def __str__(self):
    return str(self.ivalue)
```

E Partner's circleADT.py

```
## @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.
def insideCircle(p,c):
    return (p.dist(c.cen()) <= c.rad())
## @brief CircleT class for the circle ADT
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
```

F Makefile

Figure is on next page....

```
PY = python
PYFLAGS = -m unittest discover -s
DOC = doxygen
DOCFLAGS =
DOCCONFIG = doxConfig

SRC = src/

.PHONY: all test doc clean

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

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

all: test doc

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

Figure 4: Makefile