CS2ME3 Assignment 1 report

Mingnan Su
macID: sum1

February 26, 2017

Contents

1 My Codes

1.1 pointADT.py

```
#name: Mingnan Su
#macID: sum1

from math import *
## @file pointADT.py
#  @title pointADT
#  @author Mingnan Su
#  @date 2/15/2017

## @brief An ADT represents a point in 2D graph
#  @details This class represents a point in 2D graph
   with (xc,yc)
#           coordinates the position in x-axis and y-
   axis

class PointT():

    ## @brief creates a point in 2-D graph with x and y
        coordinates
    #  @param x the value of x coordinate
    #  @param y the value of y coordinate
    def __init__(self,x,y):
        self.__xc = x
        self.__yc = y


    ## @brief return the x coordinates of given object
    #  @return Returns x coordinates of the point given
    def xcrd (self):
        return self.__xc


    ## @brief return the y coordinates of given object
    #  @return Returns y coordinates of the point given
```

```
    def ycrd (self):
        return self.__yc


    ## @brief calculate the distance between two points
    #  @param pt another pointT object
    #  @return the distance between pt and self
    def dist (self,pt):
        xcq = (self.__xc - pt.xcrd()) ** 2
        ycq = (self.__yc - pt.ycrd()) ** 2
        return sqrt(xcq + ycq)


    ## @brief rotate the point with given radius
    #  @param rad the radiance of the angle
    def rot(self,rad):
        ## @var a 2*2 Matrix represent by [ [row1] [
           row2] ]
        a = [ [cos(rad) , -sin(rad)] , [sin(rad) , cos(
           rad)] ]
        ## @var x 2*1 Matrix represent by [ row1 row2 ]
        x = [ self.__xc , self.__yc ]
        ## @var r Matrix result in a*x
        r = [ [b*c for b,c in zip(row,x)] for row in a]
        self.__xc = float(format(sum(r[0]),'.5f'))
        self.__yc = float(format(sum(r[1]),'.5f'))

    ## @brief check if two pointT object is equal
    def __eq__(self,other):
        return self.__dict__ == other.__dict__
```

1.2 lineADT.py

```
#name: Mingnan Su
#macID: sum1

## @file lineADT.py
```

```
#  @title lineADT
#  @author Mingnan Su
#  @date 2/16/2017

from math import *
from pointADT import *


## @brief An ADT represents a line
class LineT:

    ## @brief Constructor for LineT
    #  @param pt one pointT at the end of the line
    #  @param pt2 another pointT at another end of the
       line
    def __init__(self,pt,pt2):
        self.__b = pt
        self.__e = pt2

    ## @brief returns the beginning point
    #  @return returns the beginning point of current
       line object
    def beg(self):
        return self.__b

    ## @brief returns the ending point
    #  @return returns the ending point of current line
        object
    def end(self):
        return self.__e

    ## @brief calculates the length of the line
    #  @return returns the length of the current line
       object
    def len(self):
        b = self.__b
        return b.dist(self.__e)
```

```
    ## @brief calculates the middle point of thel ine
    #  @return returns a PointT represents the middle
       point of the line
    def mdpt(self):
        b = self.__b
        e = self.__e
        xc = (b.xcrd() + e.xcrd()) / 2.0
        yc = (b.ycrd() + e.ycrd()) / 2.0
        return PointT(xc,yc)

    ## @brief rotate the line with given angle
    #  @param rad the angle given in radiance
    def rot(self,rad):
        self.__b.rot(rad)
        self.__e.rot(rad)

    ## @brief check if two object is equal
    def __eq__(self, other) :
        return self.__dict__ == other.__dict__
```

1.3 circleADT.py

```
#name: Mingnan Su
#macID: sum1

## @file circleADT.py
#  @title circleADT
#  @author Mingnan Su
#  @date 2/17/2017

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

## @brief an ADT represents a circle
class CircleT:

```

```
    ## @brief CircleT constructor
    #  @details Initializes a circleT object
    #           with a point in center and radius
    #  @param cin the center of the circle, type is
       PointT
    #  @param rin the radius of the circle
    def __init__ (self,cin,rin):
        self.__c = cin
        self.__r = rin

    ## @brief Gets the center of the circle
    #  @return Returns a PointT represents the center
       of the circle
    def cen(self):
        return self.__c

    ## @brief Gets the radiance of the circle
    #  @return Returns a real value represents the
       radius of the circle
    def rad(self):
        return self.__r

    ## @brief calculates the area of the cirlce
    #  @return Returns the area of the circle
    def area(self):
        return pi * (self.__r ** 2)

    ## @brief determines if the given circle intersects
        current circle object
    #  @details Select an arbitrary point inside the
       current circle, if there
    #           existed a point which is also insdie
       the given circle, this
    #           function will return true. Otherwise
       false. A point which just
    #           seats at the boundary of the circle is
       considered as "inside
    #           the circle".
```

```
    #  @param ci circle given to test
    #  @return Returns boolean variable: true if
       circles intersects; false if not
    def intersect(self,ci):
        c = self.__c
        dist = c.dist(ci.cen())
        maxRius = self.__r + ci.rad()
        return maxRius >= dist

    ## @brief draw a line between current circle object
        and given circle
    #  @param ci another circle provided to draw a line
        with
    #  @return Returns a LintT line which is the
       connection between two centers
    def connection(self,ci):
        return LineT(self.__c,ci.cen())

    ## @brief calculates the function of the force
       acting on current circle
    #         object from a given circle
    #  @details This function takes a function as input
        and output another
    #           function. Input function is used to
       parameterize the
    #           gravitational law. The output function
       allows you to calculate
    #           the force acting on current circle
       object from given circle.
    #           Output function is taken a circle as
       input and output the force.
    #  @param f a function which takes in a real value
       and output a part of
    #           the force calculation
    #  @return Returns a function which can take
       another circle and output the force
    def force(self,f):
        area = self.area()
```

```
        return lambda x: area * x.area() * f(self.
            connection(x).len())

    ## @brief check if two object is equal
    def __eq__(self,other):
        return self.__dict__ == other.__dict__
```

1.4 deque.py

```
# Name:  Mingnan Su
# MacID: sum1

## @file deque.py
#  @title deque
#  @author Mingnan Su
#  @date 1/17/2017

from math import *
from circleADT import *

## @brief An abstract object that represents a double
   ended queue of circles
class Deq:


    ## @var MAX_SIZE Exported Constants
    MAX_SIZE = 20
    ## @var s state variables - sequence of CircleT
    s = []


    ## @brief Initialize a deque
    #  @details This method is assumed to be called
       before
    #           any access program
    @staticmethod
    def init():
```

```
        Deq.s = []


    ## @brief Add a circle at the end of the list
    #  @param c A CircleT waiting to be add
    #  @exception FULL throws if the deque is full
    @staticmethod
    def pushBack(c):
        s = Deq.s
        size = len(s)
        if size == Deq.MAX_SIZE:
            raise FULL("Maximum size exceeded")
        s = s + [c]
        Deq.s = s


    ## @brief Add a circle at the beginning of the list
    #  @param c A CircleT waiting to be add
    #  @exception FULL throws if the deque is full
    @staticmethod
    def pushFront(c):
        s = Deq.s
        size = len(s)
        if size == Deq.MAX_SIZE:
            raise FULL("Maximum size exceeded")
        s = [c] + s
        Deq.s = s


    ## @brief Delete a circle from the end
    #  @exception EMPTY throws if the deque is empty
    @staticmethod
    def popBack():
        s = Deq.s
        size = len(s)
        if size == 0:
            raise EMPTY("Deque is empty.")
        s = s[0 : size - 1]
```

```
        Deq.s = s


    ## @brief Delete a circle from the beginning
    #  @exception EMPTY throws if the deque is empty
    @staticmethod
    def popFront():
        s = Deq.s
        size = len(s)
        if size == 0:
            raise EMPTY("Deque is empty.")
        s = s[1 : size]
        Deq.s = s


    ## @brief Get the last circle in the deque
    #  @return Returns the last element in the deque
    #  @exception EMPTY throws if the deque is empty
    @staticmethod
    def back():
        s = Deq.s
        size = len(s)
        if size == 0:
            raise EMPTY("Deque is empty.")
        return s[size - 1]


    ## @brief Get the first circle in the deque
    #  @return Returns the first element in the deque
    #  @exception EMPTY throws if the deque is empty
    @staticmethod
    def front():
        s = Deq.s
        size = len(s)
        if size == 0:
            raise EMPTY("Deque is empty.")
        return s[0]
```

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


    ## @brief Test if all the circle in the deque
       intersects with each other
    #  @details Retun true if all the circles in the
       deque does not interact
    #           with each other. Return false if there
       is one circle in the
    #           deque interact with another circle in
       the deque.
    #  @return Returns true if all the circle does not
       interact. False if one
    #          circle interact with another.
    #  @exception EMPTY throws if the deque is empty
    @staticmethod
    def disjoint():
        s = Deq.s
        size = len(s)
        if size == 0:
            raise EMPTY("Deque is empty.")
        ## @var r store a list of boolean - false means
            two arbitrary circle
        #         in the deque intersects each other
        r = [not i.intersect(j) for j in s for i in s
           if i != j]
        return reduce(lambda x, y: x and y, r, True)


    ## @brief Calulate the total force in x-direction
       on first circle
```

```
    #  @details This method first takes in a partial
       gravitational force
    #           function to calculate the force acting
       on first circle by another
    #           circles in the deque (for example, s[0]
        and s[2]). Then the
    #           method calculates the x-comp force
       between two circles. This
    #           process continue untill we reach the
       end of the deque. The
    #           output is the sum of the x-comp force
       exerted by all circles
    #           in the deque on the first circle (s[0])
       .
    #  @param f a partial gravitational force function
    #  @return Returns the sum of the x-comp force
       acting on first circle
    #  @exception EMPTY throws if the deque is empty
    @staticmethod
    def sumFx(f):
        s = Deq.s
        size = len(s)
        if size == 0:
            raise EMPTY("Deque is empty.")
        c0 = s[0]
        ## @brief calculate the x-comp force between
           two circles
        #  @param c circle to be calculated
        #  @return Return the x-comp force acting on c0
            by c
        def fx(c):
            force = c.force(f)(c0)
            line_len = float((c0.connection(c).len()))
            x_comp = (c.cen().xcrd() - c0.cen().xcrd())
                / line_len
            return force * x_comp
        return reduce(lambda x,y: x+y, [fx(i) for i in
           s if i != c0], 0)
```

```
    ## @brief Calculate the total area of the circles
       in the deque
    #  @details This method returns the sum of all
       circles' area in deque.
    #           Overlap between circles is not
       considered in this case, which
    #           means the output generated maybe count
       for the area overlaped
    #           more than once.
    #  @return Returns total area of all circles in the
        deque
    #  @exception EMPTY throws if the deque is empty
    @staticmethod
    def totalArea():
        s = Deq.s
        size = len(s)
        if size == 0:
            raise EMPTY("Deque is empty.")
        ## @var r store the area of each circle in a
           list
        r = [c.area() for c in s]
        return reduce(lambda x, y: x+y, r, 0)


    ## @brief Calculate the average radius of circles
       in the deque
    #  @return Returns the average of all the circles'
       radius in the deque
    #  @exception EMPTY throws if the deque is empty
    @staticmethod
    def averageRadius():
        s = Deq.s
        size = len(s)
        if size == 0:
            raise EMPTY("Deque is empty.")
```

```
        return reduce(lambda x,y: x+y, [c.rad() for c
            in s], 0)/float(size)




## @brief raise exception when the sequence is full
class FULL(Exception):
    def __init__(self,value):
        self.value = value
    def __str__(self):
        return str(self.value)

## @brief raise exception when the sequence is empty
class EMPTY(Exception):
    def __init__(self,value):
        self.value = value
    def __str__(self):
        return str(self.value)
```

1.5 testCircleDeque.py

```
#name: Mingnan Su
#macID: sum1

## @file testCircleDeque.py
#  @author Mingnan SU
#  @brief Test pointADT, lineADT, circleADT, deque
#  @date 2/18/2017

import unittest
from pointADT import *
from lineADT import *
from circleADTp import *
from deque import *

## @brief Tests all the other modules in current folder
```

```
class CircleDequeTests(unittest.TestCase):

    def setUp(self):
        #point sits at (0,1)
        self.pt = PointT(0,1)

        #line connected by (0,0) and (0,1)
        self.ln = LineT(PointT(0,0), self.pt)

        #Circle center at (0,1) have radius 1
        self.ci = CircleT(self.pt, 1)

        #Deque init
        Deq.init()
        self.c1 = CircleT(PointT(2,0), 1)
        self.c2 = CircleT(PointT(-3,0), 1)
        self.c3 = CircleT(PointT(0,-3), 1.5)

    def tearDown(self):
        self.pt = None
        self.ln = None
        self.ci = None
        Deq.init()
        self.c1 = None
        self.c2 = None
        self.c3 = None

    ## @brief Normal case - PointT xcrd return xc value
    def test_point_xcrd(self):
        self.assertTrue(self.pt.xcrd() == 0)

    ## @brief Normal case - PointT ycrd return yc value
    def test_point_ycrd(self):
        self.assertTrue(self.pt.ycrd() == 1)

    ## @brief Normal case - PointT dist return distance
        between two points
    def test_point_dist(self):
```

```
        pt2 = PointT(0,3)
        # Distance from point (0,1) to (0,3) should be
            2
        self.assertAlmostEqual(self.pt.dist(pt2), 2.0)

    ## @brief Boundary case - PointT dist return 0 when
        two points overlap
    def test_point_dist_zero(self):
        pt2 = PointT(0,1)
        self.assertAlmostEqual(self.pt.dist(pt2), 0.0)

    ## @brief Normal case - PointT rot roates a point
       by given angle
    def test_point_rot(self):
        # point (0,1) rotates 90 degrees to left
        self.pt.rot(pi/2)
        # Now the point should sits at (-1,0)
        self.assertTrue(self.pt == PointT(-1,0))

    ## @brief Boundary case - PointT rot rotates 360
       degree
    def test_point_rot_noChange(self):
        # rotate 360 degrees to left
        self.pt.rot(2*pi)
        self.assertTrue(self.pt == PointT(0,1))

    ## @brief Boundary case - PointT rot rotates 0
       degree
    def test_point_rot_zero(self):
        # rotate 0 degrees to left
        self.pt.rot(0)
        self.assertTrue(self.pt == PointT(0,1))

    ## @brief Boundary case - PointT rot rotates
       negative degree
    def test_point_rot_neg(self):
        # rotate 90 degree to right
        self.pt.rot(-pi/2.0)
```

```
        self.assertTrue(self.pt == PointT(1,0))

    ## @brief Normal case - LineT beg returns beginning
        point
    def test_line_beg(self):
        self.assertTrue(self.ln.beg() == PointT(0,0))

    ## @brief Normal case - LineT end returns ending
       point
    def test_line_end(self):
        self.assertTrue(self.ln.end() == self.pt)

    ## @brief Normal case - LineT len returns the
       length of the line
    def test_line_len(self):
        # Length of the line from (0,0) to (0,1)
        self.assertAlmostEqual(self.ln.len(), 1.0)

    ## @brief Boundary case - LineT len return zero if
       two points connecting
    #                          the line is overlapped
    def test_line_len_zero(self):
        # Length of line from (0,1) to (0,1)
        self.assertAlmostEqual(LineT(self.pt,PointT
           (0,1)).len(), 0)

    ## @brief Normal case - LineT mdpt returns the
       middle point of the line
    def test_line_mdpt(self):
        # Middile point of the line from (0,0) to (0,1)
        self.assertTrue(self.ln.mdpt() == PointT(0,0.5)
           )


    ## @brief Boundary case - LineT mdpt if two points
       are the same
    #                          (length of the line is
       zero)
```

```
    def test_line_mdpt_same(self):
        # Middle point of line from (0,1) to (0,1)
        test_line = LineT(self.pt, PointT(0,1))
        # First check if the length is zero
        self.assertTrue(test_line.len() == 0)
        # Check middle point
        self.assertTrue(test_line.mdpt() == self.pt)

    ## @brief Normal case - LineT rot rotates a point
       by given angle
    def test_line_rot(self):
        # rotate 90 degrees to the left for the line
           from (0,0) to (0,1)
        self.ln.rot(pi/2)
        # Result line should be from (0,0) to (-1,0)
        self.assertTrue(self.ln == LineT(PointT(0,0),
           PointT(-1,0)))

    ## @brief Boundary case - LineT rot rotates 360
       degrees to the left
    def test_line_rot_noChange(self):
        self.ln.rot(2*pi)
        self.assertTrue(self.ln == LineT(PointT(0,0),
           self.pt))

    ## @brief Boundary case - LineT rot rotates 0
       degree
    def test_line_rot_zero(self):
        self.ln.rot(0)
        self.assertTrue(self.ln == LineT(PointT(0,0),
           self.pt))

    ## @brief Boundary case - LineT rot rotates
       negative degree
    def test_line_rot_neg(self):
        # rotate 90 degrees to the right
        self.ln.rot(-pi/2)
        # Result line should be from (0,0) to (1,0)
```

```
        self.assertTrue(self.ln == LineT(PointT(0,0),
            PointT(1,0)))

    ## @brief Normal case - CircleT cen return center
        point of the circle
    def test_circle_cen(self):
        self.assertTrue(self.ci.cen() == self.pt)

    ## @brief Normal case - CircleT rad return the
        radius of the circle
    def test_circle_rad(self):
        self.assertTrue(self.ci.rad() == 1)

    ## @brief Normal case - CircleT area return the
        area of the circle
    def test_circle_area(self):
        self.assertAlmostEqual(self.ci.area(),
            3.1415926535)

    ## @brief Normal case - CircleT intersect test if
        two circle intersects
    def test_circle_intersect_yes(self):
        # self.circle centered at (0,1) have radius 1
        # intersect with circle centered at (0,0) with
            radius 1
        c_test = CircleT(PointT(0,0), 1)
        self.assertTrue(self.ci.intersect(c_test))

    ## @brief Normal case - CircleT intersect test if
        two circle intersects
    def test_circle_intersect_no(self):
        # self.circle centered at (0,1) have radius 1
        # does not intersect with circle centered at
            (-2,-2) with radius 1
        c_test = CircleT(PointT(-2,-2), 1)
        self.assertFalse(self.ci.intersect(c_test))
```

```
    ## @brief Boundary case - CircleT intersect test if
        two nested circles
    #                          (One circle is inside the
        another) intersects
    def test_circle_intersect_nested(self):
        # self.circle centered at (0,1) have radius 1
        # c_test with the same center but have radius 2
        c_test = CircleT(self.pt, 2)
        self.assertTrue(self.ci.intersect(c_test))

    ## @brief Boundary case - CircleT intersect test if
        two circles just
    #                          touches each other at
       boundary
    def test_circle_intersect_touch(self):
        # self.circle centered at (0,1) have radius 1
        # c_test centered at (0,-1) have radius 1
        c_test = CircleT( PointT(0,-1), 1 )
        self.assertTrue(self.ci.intersect(c_test))

    ## @brief Normal case - CircleT connection test if
       two center points connects
    def test_circle_connection(self):
        # self.circle centered at (0,1) have radius 1
        # c_test centered at (0,0) have radius 1
        c_test = CircleT( PointT(0,0), 1 )
        connect_line = self.ci.connection(c_test)
        self.assertTrue( connect_line == LineT(self.pt,
            PointT(0,0)))

    ## @brief Normal case - CircleT force Assume f(x) =
        x
    #                       Test the function c.force(f
       )
    def test_circle_force(self):
        func = lambda x: x**2
        test_func = self.ci.force(func)
        # self.circle centered at (0,1) have radius 1
```

```
        # c_test centered at (0,0) have radius 1
        c_test = CircleT( PointT(0,0), 1 )

        # test_func will take in c_test, mutiply the
            area of self.ci and c_test

        # The line connecting centers should be from
            (0,0) to (0,1)
        # So we easily knows the length of the
            connected line is 1
        # Then we get f(1) = 1**2 = 1

        #Final result should be pi*pi*1 = pi**2
        self.assertAlmostEqual(test_func(c_test), pi
            **2)

    ## @brief Normal case - Deque init check if Deq.s
        is empty list
    def test_deque_init(self):
        self.assertTrue(Deq.s == [])

    ## @brief Normal case - Deque pushBack pushes to
        the end
    def test_deque_pushBack(self):
        c1 = self.c1
        c2 = self.c2
        c3 = self.c3
        Deq.pushBack(c1)
        Deq.pushBack(c2)
        Deq.pushBack(c3)
        self.assertTrue(Deq.s == [c1,c2,c3])

    ## @brief Exception case - Deque pushBack cannot
        push to full deque
    def test_deque_pushBack_full(self):
        for _ in range(20):
            Deq.pushBack(self.c1)
```

```
        #Now the deque is full with 20 circles
        with self.assertRaises(FULL):
            Deq.pushBack(self.c1)

    ## @brief Normal case - Deque pushFront pushes to
       the front
    def test_deque_pushFront(self):
        c1 = self.c1
        c2 = self.c2
        c3 = self.c3
        Deq.pushFront(c1)
        Deq.pushFront(c2)
        Deq.pushFront(c3)
        self.assertTrue(Deq.s == [c3, c2, c1])

    ## @brief Exception case - Deque pushFront cannot
       push to full deque
    def test_deque_pushFront_full(self):
        for _ in range(20):
            Deq.pushFront(self.c1)

        #Now the deque is full with 20 circles
        with self.assertRaises(FULL):
            Deq.pushFront(self.c1)

    ## @brief Normal case - Deque popBack pops the last
        element
    def test_deque_popBack(self):
        c1 = self.c1
        c2 = self.c2
        c3 = self.c3
        Deq.pushFront(c1)
        Deq.pushFront(c2)
        Deq.pushFront(c3)
        Deq.popBack()
        #[c3, c2, c1] omit last element
        self.assertTrue(Deq.s == [c3, c2])
```

```
    ## @brief Exception case - Deque popBack cannot pop
        an empty deque
    def test_deque_popBack_empty(self):
        with self.assertRaises(EMPTY):
            Deq.popBack()

    ## @brief Normal case - Deque popFront pops the
       first element
    def test_deque_popFront(self):
        c1 = self.c1
        c2 = self.c2
        c3 = self.c3
        Deq.pushFront(c1)
        Deq.pushFront(c2)
        Deq.pushFront(c3)
        Deq.popFront()
        #[c3,c2,c1] omit first element
        self.assertTrue(Deq.s == [c2, c1])

    ## @brief Exception case - Deque popFront cannot
       pop an empty deque
    def test_deque_popFront_empty(self):
        with self.assertRaises(EMPTY):
            Deq.popFront()

    ## @brief Normal case - Deque back returns the last
        element
    def test_deque_back(self):
        c1 = self.c1
        c2 = self.c2
        c3 = self.c3
        Deq.pushBack(c1)
        Deq.pushBack(c2)
        Deq.pushBack(c3)
        # return the last element of [c1, c2, c3]
        self.assertTrue(Deq.back() == c3)
```

```
    ## @brief Exception case - Deque back cannot return
        with empty deque
    def test_deque_back_empty(self):
        with self.assertRaises(EMPTY):
            Deq.back()

    ## @brief Normal case - Deque front returns the
       first element
    def test_deque_front(self):
        c1 = self.c1
        c2 = self.c2
        c3 = self.c3
        Deq.pushBack(c1)
        Deq.pushBack(c2)
        Deq.pushBack(c3)
        # return the first element of [c1, c2, c3]
        self.assertTrue(Deq.front() == c1)

    ## @brief Exception case - Dequq front cannot
       return with empty deque
    def test_deque_front_empty(self):
        with self.assertRaises(EMPTY):
            Deq.front()

    ## @brief Normal case - Deque size return the size
       of the deque
    def test_deque_size(self):
        c1 = self.c1
        c2 = self.c2
        c3 = self.c3
        Deq.pushBack(c1)
        Deq.pushBack(c2)
        Deq.pushBack(c3)
        self.assertTrue(Deq.size() == 3)

    ## @brief Boundary case - Deque size is zero after
       initilazing
    def test_deque_size_zero(self):
```

```
        self.assertTrue(Deq.size() == 0)

    ## @brief Normal case - Deque disjoint test if all
       circles do not touches
    def test_deque_disjoint_true(self):
        c1 = self.c1
        c2 = self.c2
        c3 = self.c3
        Deq.pushBack(c1)
        Deq.pushBack(c2)
        Deq.pushBack(c3)
        #c1,c2,c3 all separated and donot intersects
           with each other
        self.assertTrue(Deq.disjoint())

    ## @brief Normal case - Deque disjoint test if one
       pair of circles touches
    def test_deque_disjoint_false(self):
        c1 = self.c1
        #c1 and c2 touches each other
        c2 = CircleT(PointT(0,0), 1)
        c3 = self.c3
        Deq.pushBack(c1)
        Deq.pushBack(c2)
        Deq.pushBack(c3)
        self.assertFalse(Deq.disjoint())

    ## @brief Boundary case - Deque disjoint with only
       one circle in the deque
    def test_deque_disjoint_one(self):
        c1 = self.c1
        Deq.pushBack(c1)
        #If there is only one circle in the deque,
        #   c1 do not have any chance touches any other
            circle,
        #   so result should be true
        self.assertTrue(Deq.disjoint())
```

```
## @brief Exception case - Deque disjoint cannot
    test with empty deque
def test_deque_disjoint_empty(self):
    with self.assertRaises(EMPTY):
        Deq.disjoint()

## @brief Normal case - Deque sumFx calculates x-
    comp force acting on s[0]
def test_deque_sumFx(self):
    c1 = self.c1        #Center (2,0) Radius 1
    c2 = self.c2        #Center (-3,0) Radius 1
    c3 = self.ci        #Center (0,1) Radius 1
    Deq.pushBack(c1)
    Deq.pushBack(c2)
    Deq.pushBack(c3)
    # Assume f(x) = x
    func = lambda x: x
    # According to the specification, the result
       should be
    #       Fx(f,c2,c1) + Fx(f,c3,c1)
    #
    # Calculating Fx(f,c1,c2)
    #           = xcomp(c1.force(f)(c2), c2, c1)
    #           = xcomp(pi*pi*5, c2, c1)
    #           = pi*pi*5*(-5/5)
    #           = - 5 * (pi ** 2)
    #
    # Calculating Fx(f,c1,c3)
    #           = xcomp(c1.force(f)(c3), c3, c1)
    #           = xcomp(pi*pi*sqrt(5), c3, c1)
    #           = pi*pi*sqrt(5)*(-2/sqrt(5))
    #           = - 2 * (pi ** 2)
    expected = -5*pi**2 + -2*pi**2
    self.assertAlmostEquals(Deq.sumFx(func),
       expected)
```

```
    ## @brief Exception case - Deque sumFx cannot
       evaluate with empty deque
    def test_deque_sumFx_empty(self):
        with self.assertRaises(EMPTY):
            Deq.sumFx(lambda x:x)

    ## @brief Normal case - Deque totalArea calculates
       the sum of area
    def test_deque_totalArea(self):
        c1 = self.c1        #Center (2,0) Radius 1
        c2 = self.c2        #Center (-3,0) Radius 1
        c3 = self.c3        #Center (0,-3) Radius 1.5
        Deq.pushBack(c1)
        Deq.pushBack(c2)
        Deq.pushBack(c3)
        # total area should be
        expected = pi + pi + pi*1.5**2
        self.assertAlmostEquals(Deq.totalArea(),
           expected)

    ## @brief Exception case - Deque totalArea cannot
       evaluate with empty deque
    def test_deque_totalArea_empty(self):
        with self.assertRaises(EMPTY):
            Deq.totalArea()

    ## @brief Normal case - Deque averageRadius
       calculates average value of radius
    def test_deque_averageRadius(self):
        c1 = self.c1        #Center (2,0) Radius 1
        c2 = self.c2        #Center (-3,0) Radius 1
        c3 = self.c3        #Center (0,-3) Radius 1.5
        Deq.pushBack(c1)
        Deq.pushBack(c2)
        Deq.pushBack(c3)
        # average radius should be
        expected = (1+1+1.5) / 3.0
```

```
        self.assertAlmostEquals(Deq.averageRadius(),
            expected)

    ## @brief Exception case - Deque averageRadius
        cannot evaluate with empty deque
    def test_deque_averageRadius_empty(self):
        with self.assertRaises(EMPTY):
            Deq.averageRadius()


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

1.6 Makefile

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

SRC = src/testCircleDeque.py

.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
```

```
        rm src/*.pyc
```

Return to content page.

2 Partner Codes

2.1 CircleADT.py

```
#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())
```

Return to content page.

3 The result of testing my files

3.1 Compilation result

```
. . . . . .
----------------------------------------------------------------------
Ran 52 tests in 0.002s

OK
```

3.2 rational for test cases

For every method in an ADT, and for every access program in deque.py, I create a normal case for each procedure.

Then, I check for exceptions for every procedure which have exceptions stated in the specification.

Also, I check for the boundary cases.

- For pointADT, I check for the distance between two overlapped points, and check for rotation of a point with 0, 360, negative degrees.
- For lineADT, I check for the length and the middle point of the line with two overlapped points. I also check for the rotation of the line with 0, 360, negative degrees.

- For circleADT, I check for the intersection between two nested circle and two boundary touched circle.
- For deque, I check if the size is zero after we initialize a deque. And I check for the result of disjoint method with only one circlei in the deque.

Return to content page.

4 The result of testing partner's files

4.1 Compilation result

```
. . . . . .

Ran 52 tests in 0.002s

OK
```

4.2 the number of passed and failed test cases

There is 52 test cases in total, all passed.

4.3 details of failed test cases

All the test cases are passed, including the boundary cases I stated above.

4.4 Summary

All test passed. As shown in my partner's code conmment, he/she already considered the boundary cases I stated in the test file. The implementation of the programs also does not make any big differences between us. Hence all the test cases are passed.

Return to content page.

5 Discussion of test results

5.1 What I learned

This assignment gives us an opportunity to experience the real world programming implementation. According to the mathematical specification, we can learn how to translate the math expression into programming language. By going through the process of sepecification implementation, we get a better understanding of the advantages of module interface specification.

5.2 Any problem I found with:

1. My program
   In my program, I found a problem that my implementation of the specification is not as clear as possible. Some part of my code may be confusing to the reader who reads my code. Also, when I was planning and writing my code, I didn't take the performance into my consideration. Thus, the true implementation may result in bad performance.

2. Partner's module

In my partner's module, all the test cases pass through my test file. The true implementation do not have much difference between he/her and me. So the problem I found with my partner's module might be similar to me (as stated above). The only problem that he/her have but I do not have is about state varibles. I make the state variable in my ADT private to prevent unexpected outernal changes in ADT objects state variable.

3. the specification of the modules
   The specification of the modules might can be improved by omitting unnecessary methods. Also, every specification of the module in this assignment depends on one or more other modules, which means as long as one module is detected with error, abstracted object deque will crash.

5.3 formal MIS compare to informal sepecification

Formal module interface specification have better format so that the reader could easily understand what the program is going to achieve compared to informal specification. In assignment 1, we are given informal specification which present the requirement of the module in plain words. Without formal mathematical expression, some idea may end up being ambiguous or igored. On the other hand, the formal module interface specification in assignment 2 helps avoiding ambiguities and have a clear table to inform what programmer should implement.

5.4 testing framework (pyUnit) advantages

Testing framewrok, in this example pyUnit, helps programmer write a better test program. Some feature, such as setUp method, avoid code duplicate in a test program and speed up the programmer to write code.

Return to content page.

6 Specification for totalArea(), averageRadius()

6.1 totalArea()

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

6.2 averageRadius()

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

Return to content page.

7 Critique of Circle Module's interface

Determine if the exported access program provide is:

- consistent
  This module is consistent. The naming convetion (circle ADT represents a circle) matches common people's perspectives. A circle is located by a point and radius, which is also easy to understand.

- essential
  A method called intersect in circleADT, could be solved by using another method connection. The programmer can evaluate the length of

the connection, and then compare to the sum of two circle's radius, the comparison result could indicates if these two circles intersect. Hence, circleADT is not essential in some senses. However, intersect method do a lot of great job by simplifying the code, since we are going to implement deque.disjoint() after.

- general
  Circle module's interface could be considered as general since it is very easy to understand and mathces common senses. But intersect method ignore the boundary cases (when a small circle is inside the big circle, this module considered these two circle intersect), which might be hard for people to understand.

- minimal
  Circle module have two methods (rad and area) which both do not take any input and output a real number. This situation might opposes the idea of minimal. But other than that, this module do a great job on minimize the program.

- opaque
  In circle module's interface, it uses pointADT and lineADT. This specification hides the details of these two modules and implement them directly. For example, when we want to create a new circle, we only need to give a real number (radius) and a PointT (a point). What the PointT should look like is not stated in the module. Hence, circle module could be considered as opaque since it hides the information of the module it used.

Return to content page.

8 Output of Deq.disjoint() discussion

8.1 Output generated by specification (with explanation)

According to specification, the output should be false, cause we only have one circle in the deque, which means the condition will never meet.

8.2 Is output above make sense?

This answer does not make sense, since if there is only one circle in the deque, the only circle does not have any chance intersect with another circle. The output should be true.

8.3 Is output above the same as the result calculated by my code?

No. The result generated by the program is true.