**Name: Session:**

**Programming I**

**Class Design – Designing a Dice Game**

**Lab Exercise 12/18/2024**

In this game we will be implementing a dice game. The rules are as follows:

1. On your first roll of two die, if you get 7 or 11 you win
2. On your first roll of two die, if you get 2, 3, or 12 you lose
3. You then repeatedly roll the two die until you get the value of your initial roll in which case you win unless you roll a 7 at any time in which case you lose.

In this program we are going to make 3 Python files (die.py, player.py, and playDice.py)

The first file die.py will be a class definition of the Die class. It will contain a constructor, roll(), getValue(), and \_\_str\_\_() methods.

**#die.py**

from random import randint

class Die():

def \_\_init\_\_(self):

self.value = 1

def roll(self):

self.value = randint(1, 6)

def getValue(self):

return self.value

def \_\_str\_\_(self):

return str(self.value)

The second file player.py will be a class definition of the Player class. It will contain a constructor, getNumberOfRolls(), play(), and \_\_str\_\_ () methods. The most important method in this class is the play method which implements all of the game logic. Notice that the Player class must import the Die class. A player object will have 3 things: two dice and a list of rolls.

**# player.py**

from die import Die

class Player():

def \_\_init\_\_ (self):

self.die1 = Die()

self.die2 = Die()

self.rolls = [ ]

def getNumberOfRolls(self):

return len(self.rolls)

def play(self):

self.rolls = [ ]

self.die1.roll()

self.die2.roll()

(d1, d2) = (self.die1.getValue(), self.die2.getValue())

self.rolls.append((d1, d2))

initialSum = d1 + d2

if initialSum in (2, 3, 12):

return False

elif initialSum in (7, 11):

return True

while True:

self.die1.roll()

self.die2.roll()

(d1, d2) = (self.die1.getValue(), self.die2.getValue())

self.rolls.append((d1, d2))

sum = d1 + d2

if sum == 7:

return False

elif sum == initialSum:

return True

def \_\_str\_\_(self):

result = ""

for (d1, d2) in self.rolls:

result += str((d1, d2)) + "\t" + str(d1 + d2) + '\n'

return result

Finally we will create a third file; playDice.py. This file will have two functions that allow us to play one game or play many games. If we play many games, we will want to keep some statistics. We will obviously want to keep track of the number of wins and losses but also keep track of the number of rolls per win and rolls per loss. In addition to the playOneGame() and playManyGames() functions we will need some code to test everything.

**## playDice.py**

from player import Player

def playOneGame():

player = Player()

youWin = player.play()

print (player)

if youWin:

print ("You win")

else:

print ("You lose")

def playManyGames(number):

wins = 0

losses = 0

winRolls = 0

lossRolls = 0

player = Player()

for count in range(number):

hasWon = player.play()

rolls = player.getNumberOfRolls()

if hasWon:

wins += 1

winRolls += rolls

else:

losses += 1

lossRolls += rolls

if wins > 0:

avgWin = winRolls/wins

else:

print('you won 0 games')

if losses > 0:

avgLoss = lossRolls/losses

else:

print('you lost 0 games')

## Test code

playOneGame()

print ("\n\n\n")

number = int(input("Enter the number of games: "))

print ("Multi-game simulation of", number, "games")

playManyGames(number)

Once your game is working, print out your shell output, attach to this sheet, answer the following questions and turn in.

1. What are the data attributes in the Die class?
2. What are the data attributes in the Player class?
3. What data type is each dice roll (contained in the list self.rolls)
4. Why does the Die class not need to be imported into the playDice.py file?
5. What does the \_\_str\_\_ method return?
6. Why must the Player class import the Die class?
7. In the playManyGames function, what happens if wins or losses is 0? How would you fix this?

Now that we have implemented a Dice game, let’s re-write the program using animated graphics. In order to accomplish this, we will need to modify the Player class (player.py) and playdice.py. We will also create a rolldice module (rolldice.py) which contains the functionality for animation of the dice.

The Die class (die.py) will be unchanged.

**## player.py**

from die import Die

from rolldice import \*

from time import sleep

class Player():

def \_\_init\_\_ (self):

self.die1 = Die()

self.die2 = Die()

self.rolls = [ ]

def getNumberOfRolls(self):

return len(self.rolls)

def play(self, canvas):

self.rolls = [ ]

(d1, d2) = roll(canvas, True)

sleep(2)

self.rolls.append((d1, d2))

initialSum = d1 + d2

if initialSum in (2, 3, 12):

return False

elif initialSum in (7, 11):

return True

while True:

canvas.drawText(200, 150,"Your point is " + str(initialSum))

(d1, d2) = roll(canvas, False)

sleep(2)

self.rolls.append((d1, d2))

sum = d1 + d2

if sum == 7:

return False

elif sum == initialSum:

return True

def \_\_str\_\_(self):

result = ""

for (d1, d2) in self.rolls:

result += str((d1, d2)) + "\t" + str(d1 + d2) + '\n'

return result

#end of player class

**## rolldice.py**

from graphics import GraphicsWindow

from random import randint

from time import sleep

# Define a constant for die size and number of dice.

DIE\_SIZE = 60

NUM\_DICE = 2

message = ''

def roll(canvas, first) :

count = 0

global message

#Create animation of dice roll

while count < 7:

thisRoll = rollDice(canvas, DIE\_SIZE)

count += 1

sleep(0.1)

if first:

message = "Your point is " + str(thisRoll[0] + thisRoll[1])

return thisRoll

def configureWindow(winSize) :

win = GraphicsWindow(winSize, winSize)

canvas = win.canvas()

canvas.setBackground(0, 128, 0)

return canvas, win

def rollDice(canvas, size) :

roll = []

canvas.clear()

xOffset = size

yOffset = size

for die in range(NUM\_DICE) :

dieValue = randint(1, 6)

drawDie(canvas, xOffset, yOffset, size, dieValue)

canvas.drawText(100, 200, message)

if die == 2 :

xOffset = size \* 2

yOffset = size \* 3

else :

xOffset = xOffset + size \* 2

roll.append(dieValue)

return tuple(roll)

def drawDie(canvas, x, y, size, dieValue) :

# The size of the dot and positioning will be based on the size of the die.

dotSize = size / 5

offset1 = dotSize / 2

offset2 = dotSize / 2 \* 4

offset3 = dotSize / 2 \* 7

# Draw the rectangle for the die.

canvas.setFill("white")

canvas.setOutline("black")

canvas.setLineWidth(2)

canvas.drawRect(x, y, size, size)

# Set the color used for the dots.

canvas.setColor("black")

canvas.setLineWidth(1)

# Draw the center dot or middle row of dots, if needed.

if dieValue == 1 or dieValue == 3 or dieValue == 5 :

canvas.drawOval(x + offset2, y + offset2, dotSize, dotSize)

elif dieValue == 6 :

canvas.drawOval(x + offset1, y + offset2, dotSize, dotSize)

canvas.drawOval(x + offset3, y + offset2, dotSize, dotSize)

# Draw the upper-left and lower-right dots, if needed.

if dieValue >= 2 :

canvas.drawOval(x + offset1, y + offset1, dotSize, dotSize)

canvas.drawOval(x + offset3, y + offset3, dotSize, dotSize)

# Draw the lower-left and upper-right dots, if needed.

if dieValue >= 4 :

canvas.drawOval(x + offset1, y + offset3, dotSize, dotSize)

canvas.drawOval(x + offset3, y + offset1, dotSize, dotSize)

**## playDice.py**

from player import Player

from rolldice import \*

import time

DIE\_SIZE = 60

def playOneGame():

player = Player()

youWin = player.play(canvas)

print(player)

if youWin:

print( "You win")

canvas.drawText(200, 150,"You win")

else:

print ("You lose")

canvas.drawText(200, 150,"You lose")

time.sleep(5)

win.close()

## Test code

canvas, win = configureWindow(5 \* DIE\_SIZE)

playOneGame()