Python Coursework

Name: Kartik Nair, **MISIS:** M00697094

Name: Firas Ahmed, **MISIS:** M00738274

About the project:

For our final coursework for this module we decided to recreate the very popular game **Snake**. The reason for choosing this game specifically is because of it's simplicity. The first iteration of the snake game was in 1976 for simple 8-bit arcade machines. It's simple graphics and controls made it perfect for low resolutions, which is why we chose it for this project. We decided to use git & GitHub (our repo is public) to be able to work together on two different machines while keeping our code in sync. It also helped us easily debug when issues came up because we could check previous versions easily to see what exactly was causing the problem. The code is highly documented & follows PEP 8 (The official Python style guide) for highly readable & understandable code.

Project code with comments:

```
import opc
import random
import copy
from tkinter import *

'''
Helper function to easily
generate tuples based on
colour name
'''


def random_px():
    return (random.randint(0, 255),
            random.randint(0, 255),
            random.randint(0, 255))


def px(color):
    if color == "red":
        return (255, 0, 0)
    elif color == "green":
        return (0, 255, 0)
    elif color == "blue":
        return (0, 0, 255)
    elif color == "black":
        return (0, 0, 0)
    elif color == "white":
        return (255, 255, 255)


'''
Window class for the
```

```
tkinter window and to
manage the controls
'''


class Window:
    def __init__(self):
        self.root = Tk()
        self.root.geometry("300x300")
        self.root.title("Coursework")
        self.root.bind("<Key>", self.key)

        # Creating a frame like this makes fluid grids possible
        frame = Frame(self.root)

        # Boilerplate to get Frame working
        Grid.rowconfigure(self.root, 0, weight=1)
        Grid.columnconfigure(self.root, 0, weight=1)
        frame.grid(row=0, column=0, sticky=N+S+E+W)

        # Create grid using frame
        grid = Frame(frame)
        grid.grid(sticky=N+S+E+W, column=0, row=0, columnspan=2)

        '''
        Now we can create and add
        the buttons to the frame.
        Also note the use of lambda
        functions below to get dynamic
        parameters in a callback
        '''
        upBtn = Button(frame, text="Up",
                       command=lambda: self.btn_event("up"))
        upBtn.grid(column=1, row=0, sticky=N+S+E+W)

        ltBtn = Button(frame, text="Left",
                       command=lambda: self.btn_event("left"))
        ltBtn.grid(column=0, row=1, sticky=N+S+E+W)

        rtBtn = Button(frame, text="Right",
                       command=lambda: self.btn_event("right"))
        rtBtn.grid(column=2, row=1, sticky=N+S+E+W)

        dnBtn = Button(frame, text="Down",
                       command=lambda: self.btn_event("down"))
        dnBtn.grid(column=1, row=2, sticky=N+S+E+W)

        restartBtn = Button(frame, text="Restart",
                            command=self.restart)
        restartBtn.grid(column=1, row=1, sticky=N+S+E+W)

        for x in range(3):
            Grid.columnconfigure(frame, x, weight=1)

        for y in range(3):
            Grid.rowconfigure(frame, y, weight=1)

    # Wrapper for Tkinter's mainloop
```

```python
    def mainloop(self):
        self.root.mainloop()

    def key(self, event):
        direction = event.keysym.lower()

        # Make sure direction is valid
        if direction == "up" or \
                direction == "down" or \
                direction == "left" or \
                direction == "right":
            snake.set_direction(direction)

    def restart(self):
        screen_arr = [[px("black") for i in range(60)] for j in range(6)]
        apple.x = 0
        apple.y = 0
        snake.body = [Point(57, 0), Point(58, 0), Point(59, 0)]
        snake.set_direction("left")

    def btn_event(self, direction):
        snake.set_direction(direction)


'''
Screen class to abstract
away the opc methods
'''


class Screen:
    def __init__(self, screen=[]):
        self.screen = screen
        self.client = opc.Client('localhost:7890')

    def render(self, screen):
        self.client.put_pixels(screen)
        self.client.put_pixels(screen)


'''
Just a simple class
for a point with an
x & y position
'''


class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __eq__(self, other):
        return self.x == other.x \
            and self.y == other.y

    def __hash__(self):
        return hash(('x', self.x, 'y', self.y))
```

```
'''
The class for the
player & snake
'''


class Snake:
    def __init__(self, direction='left'):
        self.body = [Point(57, 0), Point(58, 0), Point(59, 0)]
        self.direction = direction
        self.grow = False

    def show(self, screen):
        for point in self.body:
            screen[point.y][point.x] = px("red")

    def eat(self, food, screen):
        # Remove the food from the screen
        screen[food.y][food.x] = px("black")
        # Setting grow to true will make the snake grow in the next frame
        self.grow = True

    def set_direction(self, direction):
        # Make sure player is not allowed to go backwards
        if self.direction == "right" and direction == "left":
            self.direction = "right"
        elif self.direction == "left" and direction == "right":
            self.direction = "left"
        elif self.direction == "up" and direction == "down":
            self.direction = "up"
        elif self.direction == "down" and direction == "up":
            self.direction = "down"
        else:
            self.direction = direction

    def move(self, screen):
        # Deepcopy is required because of the Point object
        newSnake = copy.deepcopy(self.body)

        # Duplicate the previous snakes head
        newHead = newSnake[0]

        # Then move it to the next position
        if self.direction == "right":
            newHead.x += 1
        elif self.direction == "left":
            newHead.x -= 1
        elif self.direction == "up":
            newHead.y -= 1
        elif self.direction == "down":
            newHead.y += 1

        # Make sure user wraps around walls
        if newHead.x >= 60:
            newHead.x = 0
        elif newHead.x <= -1:
            newHead.x = 59

        if newHead.y >= 6:
            newHead.y = 0

        elif newHead.y <= -1:
            newHead.y = 5

        # If snake is not growing remove the last piece
        if not self.grow:
            toClear = self.body.pop()
            screen[toClear.y][toClear.x] = px("black")

        # Add back the new head
        self.body.insert(0, newHead)
        self.grow = False  # Reset the growing


'''
Read the hardcoded game over screen
array from the txt file and then evaluate
that text as a 2d array so we can
subscript it later
'''

f = open("./game-over-screen.txt", "r")
game_over_txt = ""

if f.mode == 'r':
    contents = f.read()
    game_over_txt += contents

game_over_txt = eval(game_over_txt)


'''
Update is the function
that will be called every
frame change
'''



def game_over_screen(screen):
    screen = game_over_txt

    # Make the pixels at the two edges flicker randomly
    for y in range(6):
        for x in range(60):
            if x < 9 or x > 52:
                screen[y][x] = random_px()

    return screen


def update():
    # Set the game_over variable to False by default
    game_over = False
```

```python
    # Reset the screen to black
    screen_arr = [[px("black") for i in range(60)] for j in range(6)]

    # Actually check if the game is over
    if len(set(snake.body)) != len(snake.body):
        game_over = True

    # If the game is over render out random pixels
    if game_over:
        screen_arr = game_over_screen(screen_arr)
        screen.render([j for sub in screen_arr for j in sub])
        window.root.after(frame_rate, update)
    else:
        # Add the snake to the screen
        snake.show(screen_arr)

        # Add an apple to the screen as well
        screen_arr[apple.y][apple.x] = px("green")

        # Check if snake has eaten apple
        if snake.body[0] == apple:
            # Call eat on the snake
            snake.eat(apple, screen_arr)

            '''
            Choose a new random position
            for the apple making sure not
            to intersect with the snakes position
            '''
            while True:
                apple.x = random.randint(0, 59)
                apple.y = random.randint(0, 5)

                inBody = False

                for piece in snake.body:
                    if piece == apple:
                        inBody = True

                if inBody:
                    continue
                else:
                    break

        # Actually render out the screen
        screen.render([j for sub in screen_arr for j in sub])

        # Move the snake to the next position
        snake.move(screen_arr)

        # Call update() recursively every 100 ms
        window.root.after(frame_rate, update)


window = Window()
screen = Screen()
snake = Snake()
apple = Point(0, 0)
```

```
frame_rate = 1000 // 10

# Initial call to update()
window.root.after(frame_rate, update)
window.mainloop()
```

Preview of end result

As this is a document the preview is a static image: