Project Report

Gadtardi Pratama Wongkaren 2301929480

Project: Snake Game using Python

Problem:

A snake game implementing the material taught in class.

Solution:

Although there are many possible methods to create a snake game using the pygame module in python, this particular code uses most of the material learnt in class as well as a heavy and demonstrative use of Object-Oriented Programming.

Reference

Snake game taken from freecodecamp.org

Video: https://www.youtube.com/watch?v=CD4qAhfFuLo

Original un-annotated code: https://pastebin.com/embed_js/jB6k06hG

Project Explanation

The project creates a snake game using python's pygame module. The game works by creating a "snake" that increases in length every time it eats a snack which is randomly generated on the level. If the "head" of the snake touches its own body, then the game is over. The objective of the game is to have the longest body of the snake as possible without the head of the snake 'eating itself'.

Project Components

1. **The project uses the random, pygame, tkinter modules for python**

```
#possibly to invoke random module from math module but not necessary in the current version of python
import math
#imports randomizer module from python library
import random
#imports pygame python game-making module
import pygame
#imports tkinter (graphical user interface module for python) and assigns calling name as 'tk'
import tkinter as tk
#imports messagebox module from tkinter to create message boxes
from tkinter import messagebox
```

2. The primary function and loop of the program

```
#defining the primary function and loop of the game
def main():
    #setting global variables
    global width, rows, s, snack
    #setting width of window by pixels
    width = 500
    #dividing the pixels into number of rows
    rows = 20
    #setting the length and width of the pygame window
    win = pygame.display.set_mode((width, width))
    #establishing snake color, central starting position by using 's' as an instance of the snake class
    s = snake((255,0,0), (10,10))
    #establishing snack as an instance of cube class
    snack = cube(randomSnack(rows, s), color=(0,255,0))
    #flag is condition while program is running for flow control
    flag = True

    #setting the rate of the game so it doesnt run too fast
    #using pygame internal tick rate module
    clock = pygame.time.Clock()
   #while game running:
    while flag:
        #set delay to 50 miliseconds
        pygame.time.delay(50)
        #iterate at 10 frames per second
        clock.tick(10)
        #establishes movement of snake as defined by move() function
        s.move()
        #if snake body same as position of snack (i.e. eats a snack)
        if s.body[0].pos == snack.pos:
            #adds a cube to snake body by calling addCube() function
            s.addCube()
            #defines snack as an instance of cube class with randomSnack function, has green color
            snack = cube(randomSnack(rows, s), color=(0,255,0))

        for x in range(len(s.body)):
            #if head of the snake touches its own tail, ends the game
            if s.body[x].pos in list(map(lambda z:z.pos,s.body[x+1:])):
                #prints final length of body as game score in console by measuring-
                #number of cubes in the list that is body attribute of instance 's'
                print('Score: ', len(s.body))
                #puts in the following messages in the message box
                message_box('You Lost!', 'Play again...')
                #resets to center position
                s.reset((10,10))
                #breaks the loop so the game ends
                break


        redrawWindow(win)


    pass


#runs the game by calling the primary function and loop of the game
main()
```

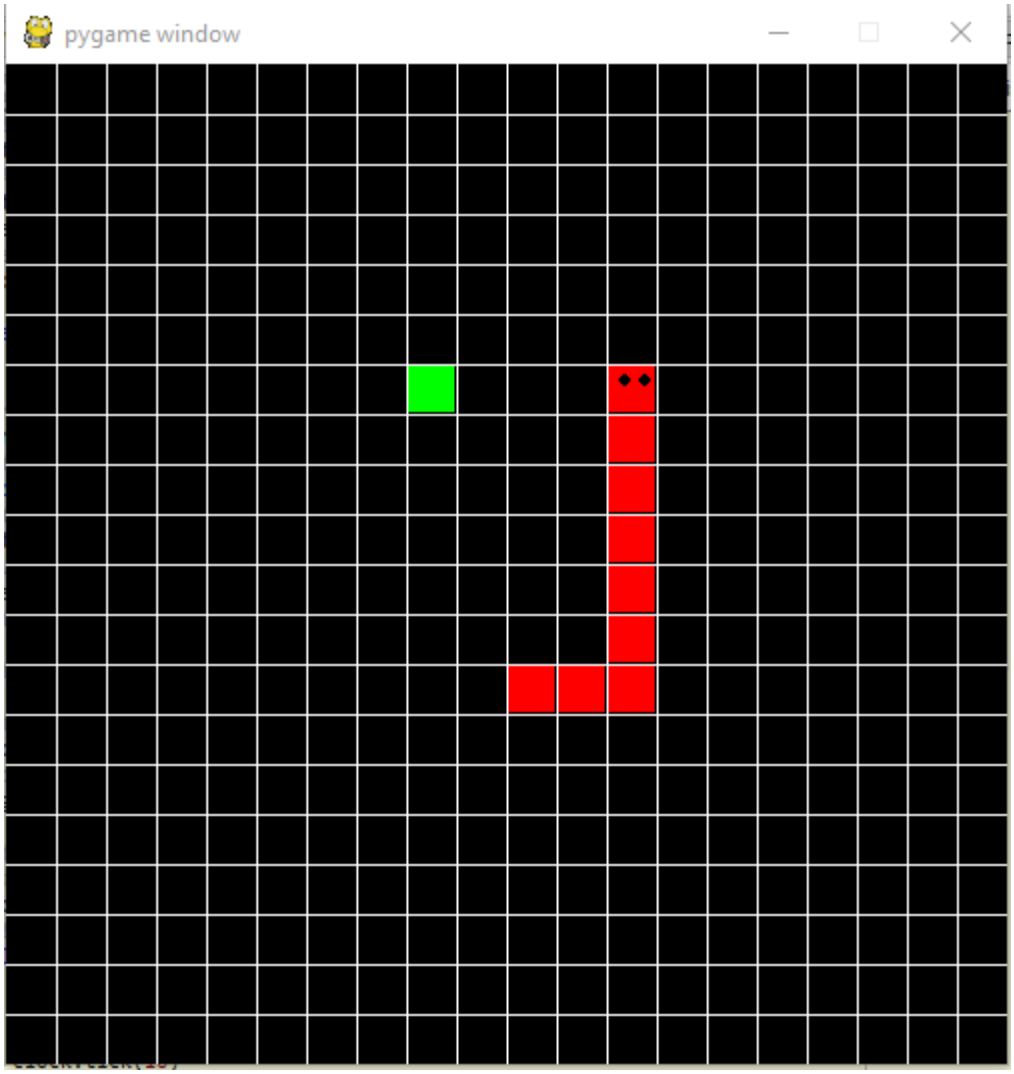
3. **The two main objects in the program are snake, and cube. The snake object is composed of cube objects.**

```
#creates the cube object which is the building block of the snake and whose size is for the grid
class cube(object):
    #number of rows divided from number of pixels (see main() function below)
    rows = 20
    #width of square of the window in pixels (see main() function below)
    w = 500
    #initializes the snake head, gives it red color
    def __init__(self,start,dirnx=1,dirny=0,color=(255,0,0)):
        self.pos = start
        self.dirnx = 1
        self.dirny = 0
        self.color = color

    #function to make the movement (add direction to position)
    def move(self, dirnx, dirny):
        self.dirnx = dirnx
        self.dirny = dirny
        self.pos = (self.pos[0] + self.dirnx, self.pos[1] + self.dirny)

    #method to establish that x,y coordinates correspond to grid, not pixels
    #eyes=False because (see draw() method below)
    def draw(self, surface, eyes=False):
        #dis(distance) is pixels of width divided by rows
        dis = self.w // self.rows
        #so x value of grid
        i = self.pos[0]
        #so y value of grid
        j = self.pos[1]

        #uses pygame to draw the square (-2 pixels to not cover white lines)
        pygame.draw.rect(surface, self.color, (i*dis+1,j*dis+1, dis-2, dis-2))
        if eyes:
            #drawing the eyes
            centre = dis//2
            radius = 3
            circleMiddle = (i*dis+centre-radius,j*dis+8)
            circleMiddle2 = (i*dis + dis -radius*2, j*dis+8)
            pygame.draw.circle(surface, (0,0,0), circleMiddle, radius)
            pygame.draw.circle(surface, (0,0,0), circleMiddle2, radius)
```

```python
#creates snake object class
class snake(object):
    #in preparation to make the snake's body a list of cubes
    body = []
    turns = {}
    #setting class attributes
    def __init__(self, color, pos):
        self.color = color
        #snake head is a cube at a certain position
        self.head = cube(pos)
        #snake body is a list of cubes appended to the head
        self.body.append(self.head)
        #defines movement direction for snake (remembering 0,0 is top left of pygame window)
        self.dirnx = 0
        self.dirny = 1

    def move(self):
        #script to make sure the game quits when we want to close it
        #tells python to loop through all different event 'types' available in pygame module
        for event in pygame.event.get():
            #if pygame 'quit' event triggered
            if event.type == pygame.QUIT:
                #then quit the game
                pygame.quit()

            #script to define movement of snake
            #'calls up' the list of keys in pygame as keys and checks if they get pressed
            keys = pygame.key.get_pressed()
            #if a certain key in 'keys' is pressed:
            for key in keys:
                #if left directional key is pressed:
                if keys[pygame.K_LEFT]:
                    #movement direction towards left
                    self.dirnx = -1
                    #one value equals zero to make sure snake is only moving in one direction
                    self.dirny = 0
                    #trick using dictionary key-value assignment to make turn at current head position
                    self.turns[self.head.pos[:]] = [self.dirnx, self.dirny]
                #elif so no multiple directions
                elif keys[pygame.K_RIGHT]:
                    self.dirnx = 1
                    self.dirny = 0
                    self.turns[self.head.pos[:]] = [self.dirnx, self.dirny]

                elif keys[pygame.K_UP]:
                    self.dirnx = 0
                    #in pygame the higher the y value the more downwards the coordinates go
                    self.dirny = -1
                    self.turns[self.head.pos[:]] = [self.dirnx, self.dirny]

                elif keys[pygame.K_DOWN]:
                    self.dirnx = 0
                    self.dirny = 1
                    self.turns[self.head.pos[:]] = [self.dirnx, self.dirny]

        for i, c in enumerate(self.body):
            #position of cube
            p = c.pos[:]
            if p in self.turns:
```

```
        for i, c in enumerate(self.body):
            #position of cube
            p = c.pos[:]
            if p in self.turns:
                turn = self.turns[p]
                c.move(turn[0],turn[1])
                #so the snake doesnt turn in the same direction if it hits the coordinate a turn was initiated
                if i == len(self.body)-1:
                    self.turns.pop(p)

            #script to make sure if snake reaches edge of screen it pops out the other side
            else:
                #if snake is moving leftwards and x postion equal/less than 0, move to rightest row, same y val
                if c.dirnx == -1 and c.pos[0] <= 0: c.pos = (c.rows-1, c.pos[1])
                #rightwards
                elif c.dirnx == 1 and c.pos[0] >= c.rows-1: c.pos = (0,c.pos[1])
                #downwards
                elif c.dirny == 1 and c.pos[1] >= c.rows-1: c.pos = (c.pos[0], 0)
                #upwards
                elif c.dirny == -1 and c.pos[1] <= 0: c.pos = (c.pos[0],c.rows-1)
                #if snake isnt at edge of screen or turning, keep moving in its present direction
                else: c.move(c.dirnx,c.dirny)

    #when resetting the game
    def reset(self, pos):
        self.head = cube(pos)
        #clears body tail and position
        self.body = []
        self.body.append(self.head)
        self.turns = {}
        self.dirnx = 0
        self.dirny = 1

#defining the 'tail' of the snake
    def addCube(self):
        #adding tail behind the head
        tail = self.body[-1]
        dx, dy = tail.dirnx, tail.dirny

        #if body moving right, add tail to the left
        if dx == 1 and dy == 0:
            self.body.append(cube((tail.pos[0]-1,tail.pos[1])))
        elif dx == -1 and dy == 0:
            self.body.append(cube((tail.pos[0]+1,tail.pos[1])))
        elif dx == 0 and dy == 1:
            self.body.append(cube((tail.pos[0],tail.pos[1]-1)))
        elif dx == 0 and dy == -1:
            self.body.append(cube((tail.pos[0],tail.pos[1]+1)))

        #make the tail move and follow direction of body
        self.body[-1].dirnx = dx
        self.body[-1].dirny = dy


    def draw(self, surface):
        for i, c in enumerate(self.body):
            #if first cube in list (the head):
            if i ==0:
                #draw eyes
                c.draw(surface, True)
            else:
                c.draw(surface)
```

Screenshot of program at work:



Shortcomings of code:

If snake has a tail already and is told to move in the direction of its own tail, the game will end. Possible solutions include clause where if snake told to move in direction of itself, command will be ignored.