Odd Semester (2023)

BINUS UNIVERSITY INTERNATIONAL

Assignment Cover Letter

(Individual Work)

Student Information:		Surname	Given Names	Student ID Number
	1.	Thiodoris	Dyllan	2301939020

Course Code	**: COMP6502**	**Course Name**	**: Introduction to Programming**
Class	**: L1CC**	**Name of Lecturer(s)**	**: Ida Bagus Kerthyayana Manuaba**
Major	**: CS**		
Title of Assignment	: Simple snake game		
Type of Assignment	**: Final Project**		
Submission Pattern			
Due Date	**: 17-01-20**	**Submission Date**	**:**

The assignment should meet the below requirements.

1. Assignment (hard copy) is required to be submitted on clean paper, and (soft copy) as per lecturer's instructions.
2. Soft copy assignment also requires the signed (hardcopy) submission of this form, which automatically validates the softcopy submission.
3. The above information is complete and legible.
4. Compiled pages are firmly stapled.
5. Assignment has been copied (soft copy and hard copy) for each student ahead of the submission.

Plagiarism/Cheating

BiNus International seriously regards all forms of plagiarism, cheating and collusion as academic offenses which may result in severe penalties, including loss/drop of marks, course/class discontinuity and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

Declaration of Originality

By signing this assignment, I understand, accept and consent to BiNus International terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.

Signature of Student: (Name of Student)

1. Dyllan Thiodoris

"Snake Game"

Name : Dyllan Thiodoris

ID : 2301939020

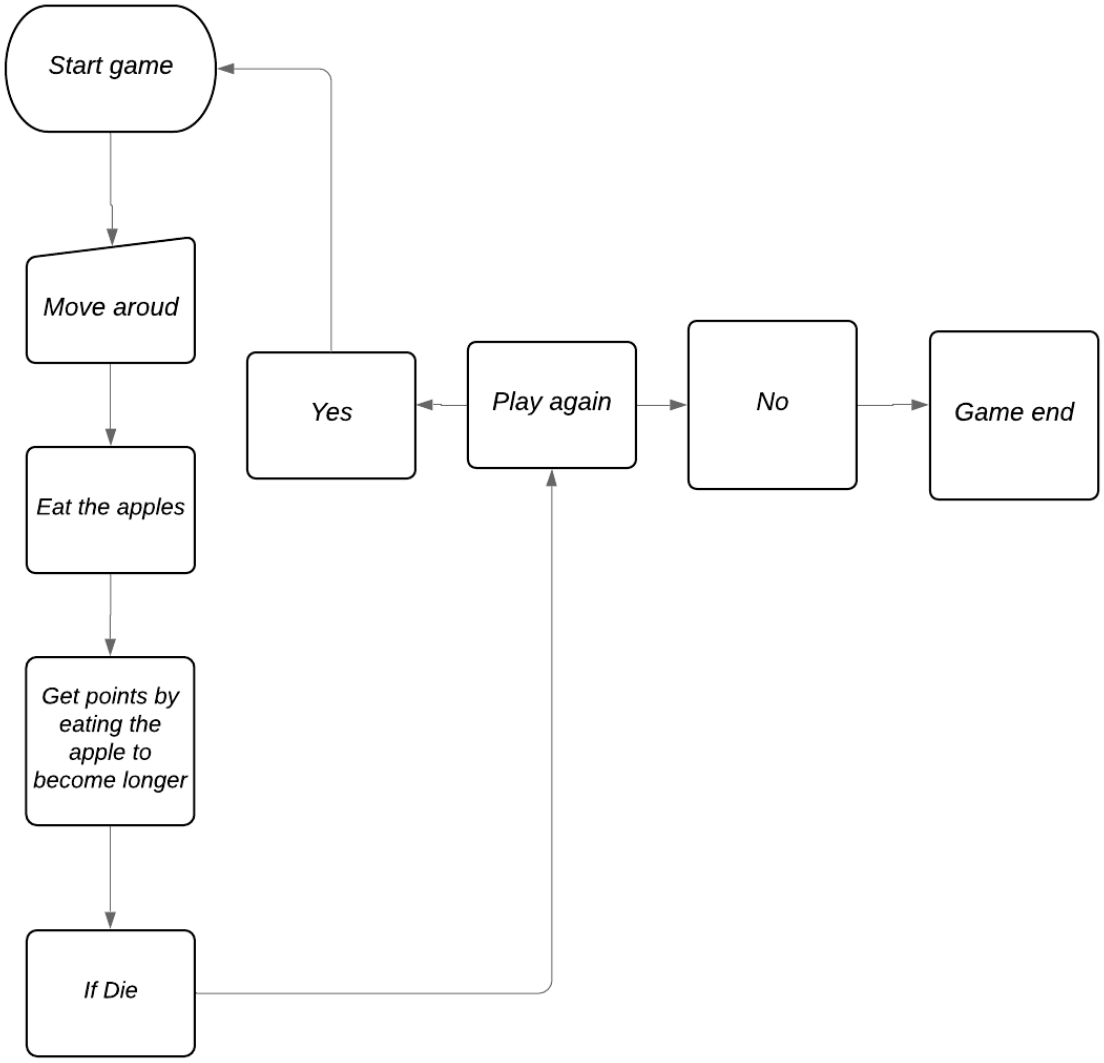
I. Description

The function of this program:

The purpose of this program is to make people play older games and convince them that old games are still really fun to play.

II.a. Design/Plan

Project's Hierarchy Chart



II.b. Explananinng Some Of The Function Inside the Class

FinalCode.py

- **def move(self):**

  it moves the snake in any direction the arrow key is pointing at

- **def getHead(self):**

  get the head of the snake

- **def grow(self):**

  it grows every time its eats and apple

- **def getX(self):**

  gets the x coordinate

- **def getY(self):**

  gets the y coordinate

- **def setX(self):**

  sets the x coordinate

- **def setY(self):**

  sets the y coordinate

- **def respawnApple(apple ,index ,sx , sy):**

  respawns the apple

III. Lessons Things that Have Been Learned

```
def respawnApple(apples, index, sx, sy):
    radius = math.sqrt((SCREEN_WIDTH / 2 * SCREEN_WIDTH / 2 + SCREEN_HEIGHT / 2 *
SCREEN_HEIGHT / 2)) / 2
    angle = 999
    while (angle > radius):
        angle = random.uniform(0, 800) * math.pi * 2
        x = SCREEN_WIDTH / 2 + radius * math.cos(angle)
        y = SCREEN_HEIGHT / 2 + radius * math.sin(angle)
        if (x == sx and y == sy):
```

```
            continue
    newApple = Apple(x, y, 1)
    apples[index] = newApple
```

I learned that you could have the apple to respawn at an area that is completely different.

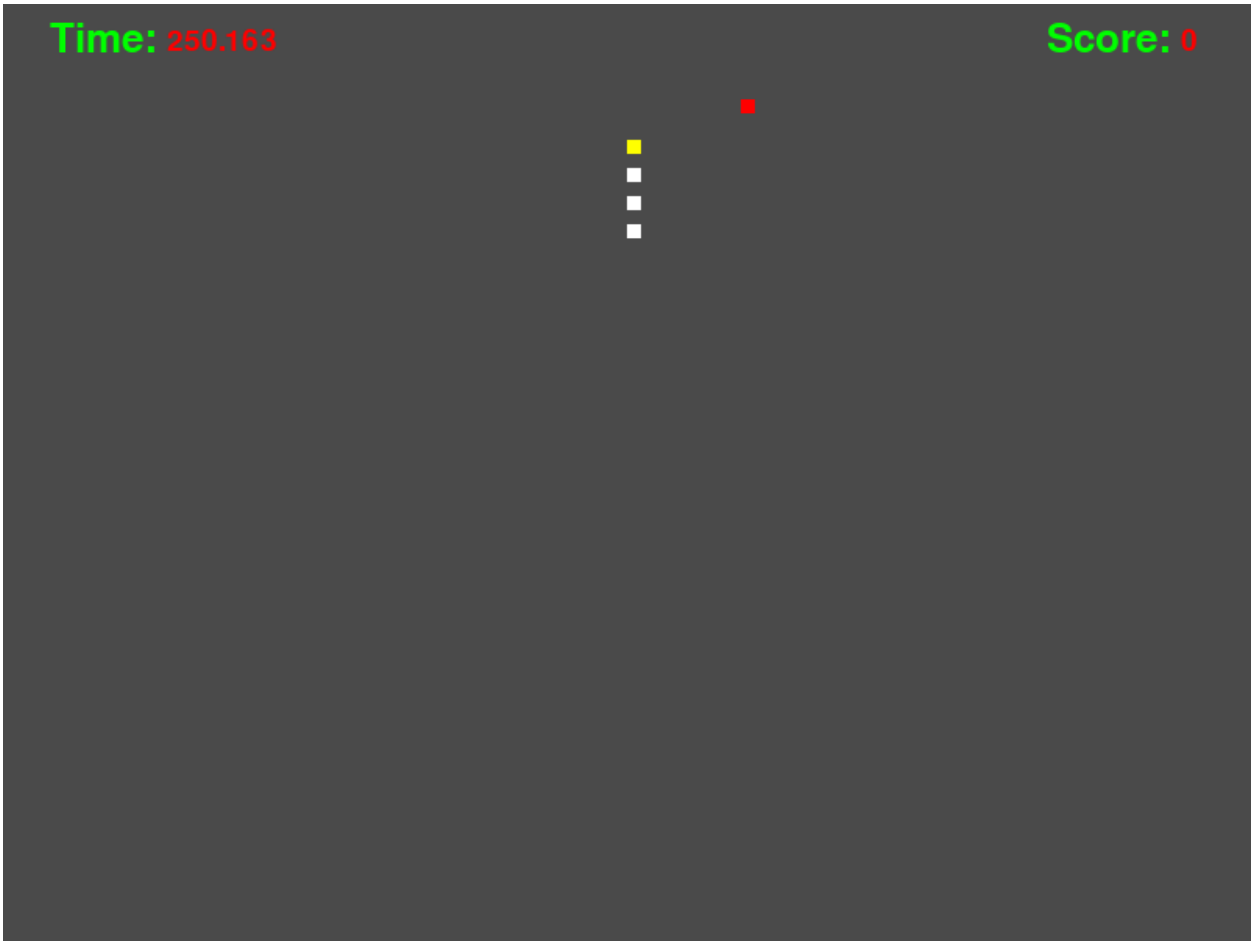
```
def drawScore(score):
    score_numb = score_numb_font.render(str(score), 1, pygame.Color("red"))
    screen.blit(score_msg, (SCREEN_WIDTH - score_msg_size[0] - 60, 10))
    screen.blit(score_numb, (SCREEN_WIDTH - 45, 14))
```

I also learned how you could make score board and put it into real-time in the game

```
def drawGameTime(gameTime):
    game_time = score_font.render("Time:", 1, pygame.Color("green"))
    game_time_numb = score_numb_font.render(str(gameTime / 1000), 1,
pygame.Color("red"))
    screen.blit(game_time, (30, 10))
    screen.blit(game_time_numb, (105, 14))
```

The timer is also a thing that I learned in the past days

Screenshots:



Resources :

- https://gist.github.com/someoneigna/5022021 (background code along with the algorithms)

V. Source Code

```
import
sys,os,pygame,r
andom,math
                    pygame.init()
                    pygame.display.set_caption("Nsnake v1.0")
                    pygame.font.init()
                    random.seed()
                    #Global constant definitions
                    SPEED = 0.36
                    SNAKE_SIZE = 9
                    APPLE_SIZE = SNAKE_SIZE
```

```
SEPARATION = 10
SCREEN_HEIGHT = 600
SCREEN_WIDTH = 800
FPS = 25
KEY = {"UP":1,"DOWN":2,"LEFT":3,"RIGHT":4}
#Screen initialization
screen =
pygame.display.set_mode((SCREEN_WIDTH,SCREEN_HEIGHT),pygame.HWSURFACE)
#Resources
score_font = pygame.font.Font(None,38)
score_numb_font = pygame.font.Font(None,28)
game_over_font = pygame.font.Font(None,46)
play_again_font = score_numb_font
score_msg = score_font.render("Score:",1,pygame.Color("green"))
score_msg_size = score_font.size("Score")
#icon = pygame.image.load("nsnake32.png")
#pygame.display.set_icon(icon)
background_color = pygame.Color(74,74,74)
black = pygame.Color(0,0,0)
#Clock
gameClock = pygame.time.Clock()
def checkCollision(posA,As,posB,Bs):
    #As size of a | Bs size of B
    if(posA.x   < posB.x+Bs and posA.x+As > posB.x and posA.y < posB.y + Bs
and posA.y+As > posB.y):
        return True
    return False
def checkLimits(entity):
    if(entity.x > SCREEN_WIDTH):
        entity.x = SNAKE_SIZE
    if(entity.x < 0):
        entity.x = SCREEN_WIDTH - SNAKE_SIZE
    if(entity.y > SCREEN_HEIGHT):
        entity.y = SNAKE_SIZE
    if(entity.y < 0):
        entity.y = SCREEN_HEIGHT - SNAKE_SIZE

class Apple:
    def __init__(self,x,y,state):
        self.x = x
        self.y = y
        self.state = state
        self.color = pygame.color.Color("red")
    def draw(self,screen):
```

```
pygame.draw.rect(screen,self.color,(self.x,self.y,APPLE_SIZE,APPLE_SIZE),0)

class Segment:
    def __init__(self,x,y):
        self.x = x
        self.y = y
        self.direction = KEY["UP"]
        self.color = "white"

class Snake:
    def __init__(self,x,y):
        self.x = x
        self.y = y
        self.direction = KEY["UP"]
        self.stack = []

        self.stack.append(self)

        blackBox = Segment(self.x,self.y + SEPARATION)
        blackBox.direction = KEY["UP"]
        blackBox.color = "NULL"
        self.stack.append(blackBox)


    def move(self):
        last_element = len(self.stack)-1
        while(last_element != 0):
            self.stack[last_element].direction = self.stack[last_element-
1].direction
            self.stack[last_element].x = self.stack[last_element-1].x
            self.stack[last_element].y = self.stack[last_element-1].y
            last_element-=1
        if(len(self.stack)<2):
            last_segment = self
        else:
            last_segment = self.stack.pop(last_element)
        last_segment.direction = self.stack[0].direction
        if(self.stack[0].direction ==KEY["UP"]):
            last_segment.y = self.stack[0].y - (SPEED * FPS)
        elif(self.stack[0].direction == KEY["DOWN"]):
            last_segment.y = self.stack[0].y + (SPEED * FPS)
        elif(self.stack[0].direction ==KEY["LEFT"]):
            last_segment.x = self.stack[0].x - (SPEED * FPS)
```

```
        elif(self.stack[0].direction == KEY["RIGHT"]):
            last_segment.x = self.stack[0].x + (SPEED * FPS)
        self.stack.insert(0,last_segment)
    def getHead(self):
        return(self.stack[0])

    def grow(self):
        last_element = len(self.stack)-1
        self.stack[last_element].direction =
self.stack[last_element].direction
        if(self.stack[last_element].direction == KEY["UP"]):
            newSegment = Seg-
ment(self.stack[last_element].x,self.stack[last_element].y-SNAKE_SIZE)
            blackBox = Segment(newSegment.x,newSegment.y-SEPARATION)

        elif(self.stack[last_element].direction == KEY["DOWN"]):
            newSegment = Seg-
ment(self.stack[last_element].x,self.stack[last_element].y+SNAKE_SIZE)
            blackBox = Segment(newSegment.x,newSegment.y+SEPARATION)

        elif(self.stack[last_element].direction == KEY["LEFT"]):
            newSegment = Segment(self.stack[last_element].x-
SNAKE_SIZE,self.stack[last_element].y)
            blackBox = Segment(newSegment.x-SEPARATION,newSegment.y)

        elif(self.stack[last_element].direction == KEY["RIGHT"]):
            newSegment = Seg-
ment(self.stack[last_element].x+SNAKE_SIZE,self.stack[last_element].y)
            blackBox = Segment(newSegment.x+SEPARATION,newSegment.y)

        blackBox.color = "NULL"
        self.stack.append(newSegment)
        self.stack.append(blackBox)

    def iterateSegments(self,delta):
        pass

    def setDirection(self,direction):
        if(self.direction == KEY["RIGHT"] and direction == KEY["LEFT"] or
self.direction == KEY["LEFT"] and direction == KEY["RIGHT"]):
            pass
        elif(self.direction == KEY["UP"] and direction == KEY["DOWN"] or
self.direction == KEY["DOWN"] and direction == KEY["UP"]):
            pass
```

```
            else:
                self.direction = direction

        def get_rect(self):
            rect = (self.x,self.y)
            return rect

        def getX(self):
            return self.x

        def getY(self):
            return self.y

        def setX(self,x):
            self.x = x

        def setY(self,y):
            self.y = y

        def checkCrash(self):
            counter = 1
            while(counter < len(self.stack)-1):

if(checkCollision(self.stack[0],SNAKE_SIZE,self.stack[counter],SNAKE_SIZE)a
nd self.stack[counter].color != "NULL"):
                    return True
                counter+=1
            return False

        def draw(self,screen):

pygame.draw.rect(screen,pygame.color.Color("yellow"),(self.stack[0].x,self.
stack[0].y,SNAKE_SIZE,SNAKE_SIZE),0)
            counter = 1
            while(counter < len(self.stack)):
                if(self.stack[counter].color == "NULL"):
                    counter+=1
                    continue

pygame.draw.rect(screen,pygame.color.Color("white"),(self.stack[counter].x,
self.stack[counter].y,SNAKE_SIZE,SNAKE_SIZE),0)
                counter+=1
```

```
def getKey():
        for event in pygame.event.get():
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_UP:
                    return KEY["UP"]
                elif event.key == pygame.K_DOWN:
                    return KEY["DOWN"]
                elif event.key == pygame.K_LEFT:
                    return KEY["LEFT"]
                elif event.key == pygame.K_RIGHT:
                    return KEY["RIGHT"]
                elif event.key == pygame.K_ESCAPE:
                    return "exit"
                elif event.key == pygame.K_y:
                    return "yes"
                elif event.key == pygame.K_n:
                    return "no"
            if event.type == pygame.QUIT:
                sys.exit()
def respawnApple(apples,index,sx,sy):
    radius = math.sqrt((SCREEN_WIDTH/2*SCREEN_WIDTH/2  +
SCREEN_HEIGHT/2*SCREEN_HEIGHT/2))/2
    angle = 999
    while(angle > radius):
        angle = random.uniform(0,800)*math.pi*2
        x = SCREEN_WIDTH/2 + radius * math.cos(angle)
        y = SCREEN_HEIGHT/2 + radius * math.sin(angle)
        if(x == sx and y == sy):
            continue
    newApple = Apple(x,y,1)
    apples[index] = newApple

def respawnApples(apples,quantity,sx,sy):
    counter = 0
    del apples[:]
    radius = math.sqrt((SCREEN_WIDTH/2*SCREEN_WIDTH/2  +
SCREEN_HEIGHT/2*SCREEN_HEIGHT/2))/2
    angle = 999
    while(counter < quantity):
        while(angle > radius):
            angle = random.uniform(0,800)*math.pi*2
            x = SCREEN_WIDTH/2 + radius * math.cos(angle)
            y = SCREEN_HEIGHT/2 + radius * math.sin(angle)
            if( (x-APPLE_SIZE == sx or x+APPLE_SIZE == sx) and (y-
```

```
APPLE_SIZE == sy or y+APPLE_SIZE == sy) or radius - angle <= 10):
                continue
        apples.append(Apple(x,y,1))
        angle = 999
        counter+=1

def endGame():
    message = game_over_font.render("Game Over",1,pygame.Color("white"))
    message_play_again = play_again_font.render("Play Again?
Y/N",1,pygame.Color("green"))
    screen.blit(message,(320,240))
    screen.blit(message_play_again,(320+12,240+40))
    pygame.display.flip()
    pygame.display.update()

    myKey = getKey()
    while(myKey != "exit"):
        if(myKey == "yes"):
            main()
        elif(myKey == "no"):
            break
        myKey = getKey()
        gameClock.tick(FPS)
    sys.exit()
def drawScore(score):
    score_numb = score_numb_font.render(str(score),1,pygame.Color("red"))
    screen.blit(score_msg, (SCREEN_WIDTH-score_msg_size[0]-60,10) )
    screen.blit(score_numb,(SCREEN_WIDTH - 45,14))

def drawGameTime(gameTime):
    game_time = score_font.render("Time:",1,pygame.Color("green"))
    game_time_numb =
score_numb_font.render(str(gameTime/1000),1,pygame.Color("red"))
    screen.blit(game_time,(30,10))
    screen.blit(game_time_numb,(105,14))

def exitScreen():
    pass
def main():
    score = 0
    #Snake initialization
    mySnake = Snake(SCREEN_WIDTH/2,SCREEN_HEIGHT/2)
    mySnake.setDirection(KEY["UP"])
    mySnake.move()
```

```
    start_segments=3
    while(start_segments>0):
        mySnake.grow()
        mySnake.move()
        start_segments-=1
    #Apples
    max_apples = 1
    eaten_apple = False
    apples = [Ap-
ple(random.randint(60,SCREEN_WIDTH),random.randint(60,SCREEN_HEIGHT),1)]
    respawnApples(apples,max_apples,mySnake.x,mySnake.y)

    startTime = pygame.time.get_ticks()
    endgame = 0

    while(endgame!=1):
        gameClock.tick(FPS)
        #Input
        keyPress = getKey()
        if keyPress == "exit":
            endgame = 1

        #Collision check
        checkLimits(mySnake)
        if(mySnake.checkCrash()== True):
            endGame()

        for myApple in apples:
            if(myApple.state == 1):

if(checkCollision(mySnake.getHead(),SNAKE_SIZE,myApple,APPLE_SIZE)==True):
                    mySnake.grow()
                    myApple.state = 0
                    score+=5
                    eaten_apple=True

        #Position Update
        if(keyPress):
            mySnake.setDirection(keyPress)
        mySnake.move()



        #Respawning apples
```

```
            if(eaten_apple == True):
                eaten_apple = False
                respawnApple(apples,0,mySnake.getHead().x,mySnake.getHead().y)
            #Drawing
            screen.fill(background_color)
            for myApple in apples:
                if(myApple.state == 1):
                    myApple.draw(screen)

            mySnake.draw(screen)
            drawScore(score)
            gameTime = pygame.time.get_ticks() - startTime
            drawGameTime(gameTime)

            pygame.display.flip()
            pygame.display.update()




main()
```