BINUS UNIVERSITY
BINUS INTERNATIONAL

Assignment Cover Letter

(Individual Work)

Student Information:	Surname	Given Names	Student ID Number
1.	**Putra**	**Bryan**	**2301890983**

Course Code	**: COMP6502**	**Course Name**	**: Introduction to Programming**
Class	**: L1AC**	**Name of Lecturer(s)**	**:Ida Bagus Kerthyayana Manuaba, S.T., Ph.D**
Major	**: CS**		
Title of Assignment (if any)	: NubSnake		
Type of Assignment	**: Final Project**		
Submission Pattern			
Due Date	**: 17-01-2020**	**Submission Date**	**: 17-01-2020**

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. Brenda Spears

"Nub Snake"

Name : Bryan Putra

ID : 2301890983

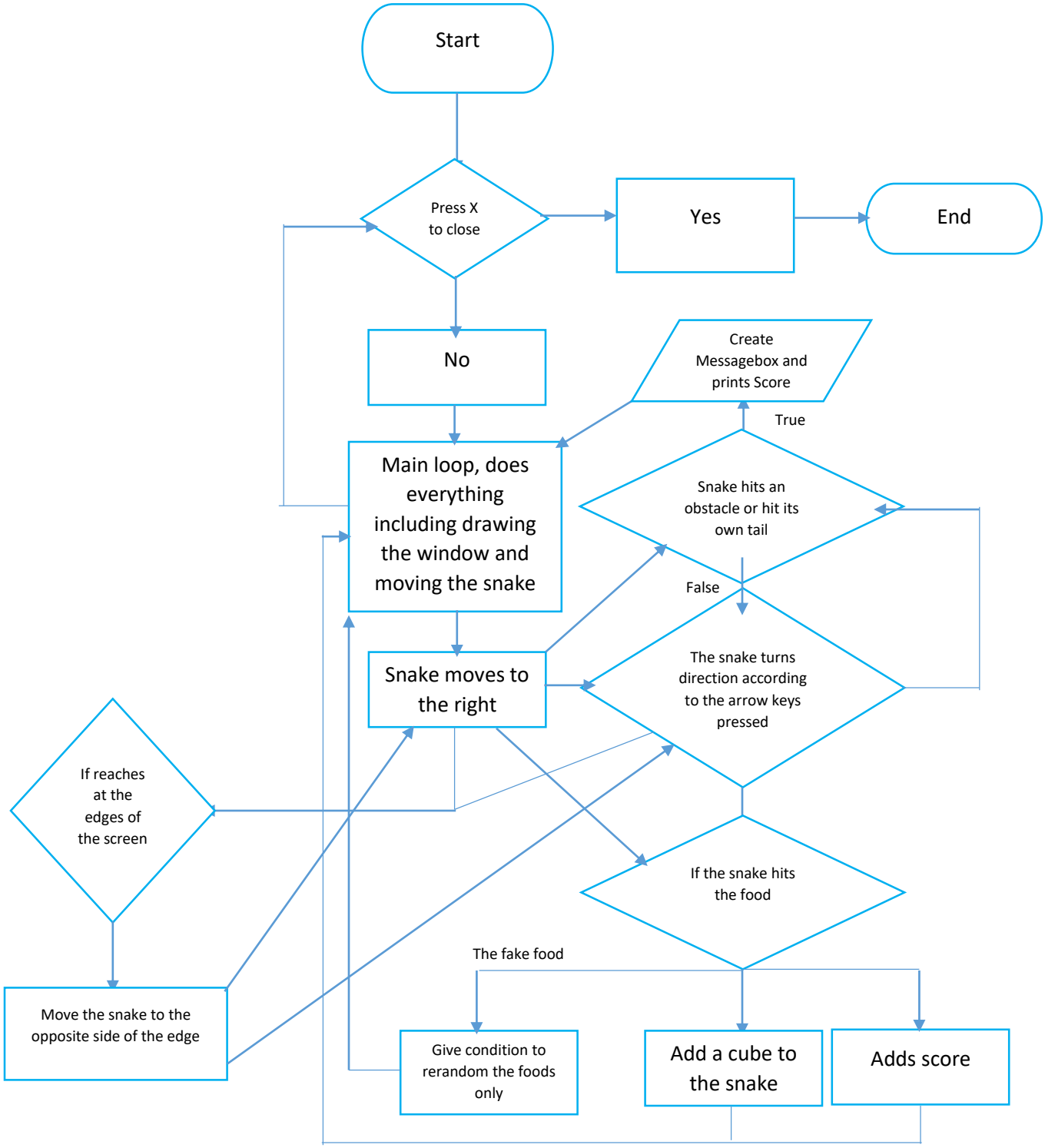
I. Project Specification

NubSnake is a snake game that has been here for a long time, it uses cubes and moves according to each of its positions, and unlike any normal snake games there are some small obstacles there and not just 1 food but 3 foods, 2 of them are fakes and only 1 of them is the real food which will make the snake longer

When I was younger, I used to play a lot of old phone games after studying. I had a private tutor who comes to my house according to the schedule to teach me, and he always lend me his phone to play games after I finished his questions, the better my score is the longer I get to play with his phone. While thinking about my project I remembered this and decided to make one of the games that I used to play which is the snake game, and I hope that people will feel nostalgic playing this game because it is pretty old. But rather than making it like the normal game I added a few twists to it to make it maybe more fun for some people.

II. Solution Design

Flowchart



III. What is happening in the code

```
import random
import pygame
import tkinter as tk
from tkinter import messagebox
```

these are the library that are used to make the code happen

```
class Cube(object):
    rows = 20
    w = 500

    def __init__(self, start, dirnx=1, dirny=0, color=(0, 0, 255)): # dirnx is
direction x
        self.position = start
        self.dirnx = 1 # start with the snake moving
        self.dirny = 0
        self.color = color

    def move(self, dirnx, dirny):
        self.dirnx = dirnx          # direction x
        self.dirny = dirny          # direction y
        self.position = (self.position[0] + self.dirnx, self.position[1] +self.dirny)
# this is to move the self.position[0] which is the x of the cube by adding the
direction that is given by pressing the key

    def draw(self, surface, eyes=False):
        distance = self.w // self.rows  # which is 25
        i = self.position[0]            # row
        j = self.position[1]            # column
        pygame.draw.rect(surface, self.color, (i * distance + 1, j * distance + 1,
distance - 2, distance - 2)) #distance -2 is the area, i * distance + 1 is x y
        if eyes:
            center = distance // 2 # the center of the cube
            radius = 3 # radius of the eye
            eye1 = (i * distance + center - radius, j * distance + 8) # i * distance
is the position x + center - radius is the position of the circle inside the cube
            eye2 = (i * distance + distance - radius * 2, j * distance + 8) # same as
top but the second eye
            pygame.draw.circle(surface, (0, 0, 0), eye1, radius) # draws the eye
            pygame.draw.circle(surface, (0, 0, 0), eye2, radius) # draws the second
eye
```

So the Cube is a class which are gonna be used for the cubes in the snake.
The def init is to initialize the attributes which are given. The def move function is for moving the cube, the draw function is for drawing the Cube.

```
class Snake(object):
    body = []       #list of cubes that will be the body
    turns = {}      #this is where i store the positions of the head of the snake and
why is below

    def __init__(self, color, position):
```

```
        self.color = color
        self.head = Cube(position)      #this is so we know the head's position at
all times
        self.body.append(self.head)     #append the head to the first list of the
body
        self.dirnx = 1                  #so when the game starts it immediately moves
first before the given command not just stay still
        self.dirny = 0

    def move(self):
        for event in pygame.event.get():        # so we can close the game
            if event.type == pygame.QUIT:
                pygame.quit()

            keys = pygame.key.get_pressed()
            for key in keys:
                if keys[pygame.K_LEFT]:     # if the left key is pressed then it
moves left by moving the x -1 and y 0
                    self.dirnx = -1
                    self.dirny = 0
                    self.turns[self.head.position[:]] = [self.dirnx, self.dirny] # to
remember the way the head turns so that the tail can also turn after the main head
has turned directions
                    # so the key is the current position of the head of the snake,
and it is equal to what direction it is turning to
                    # so when we turn it creates and adds it to the turns list
                elif keys[pygame.K_RIGHT]:
                    self.dirnx = 1          # if the right key is pressed then it
moves left by adding the x 1 and y 0
                    self.dirny = 0
                    self.turns[self.head.position[:]] = [self.dirnx, self.dirny] #
the same as the top

                elif keys[pygame.K_UP]:     # if the up key is pressed then it moves
left by moving the x 0 and y -1
                    self.dirnx = 0
                    self.dirny = -1
                    self.turns[self.head.position[:]] = [self.dirnx, self.dirny]
                elif keys[pygame.K_DOWN]:   # if the down key is pressed then it
moves left by moving the x 0 and y -1
                    self.dirnx = 0
                    self.dirny = 1
                    self.turns[self.head.position[:]] = [self.dirnx, self.dirny]

        for i, c in enumerate(self.body):   #look through the list of positions that
we have on the snake, i is the index and c is cube
            pos = c.position[:] # for each cube, grabs the position copy all the
elements in the position index
            if pos in self.turns:# if the position is in the turns dictionary
                turn = self.turns[pos]# the turn that we chose the turn list at the
index
                c.move(turn [0], turn[1])# give the cube the direction x and y after
we pressed a direction button
                if i == len(self.body) - 1:#if we are in the last cube
                    self.turns.pop(pos)# if we dont remove the turn the turn will
```

```
activate when we hit the position that was in our last tail
            else:
                if c.dirnx == -1 and c.position[0] <= 0: #if snake is moving left and
the position x is less than or equal to 0
                    c.position = (c.rows - 1, c.position[1]) #change the position to
the right side of the screen
                elif c.dirnx == 1 and c.position[0] >= c.rows - 1:
                    c.position = (0, c.position[1])
                elif c.dirny == 1 and c.position[1] >= c.rows - 1:
                    c.position = (c.position[0], 0)
                elif c.dirny == -1 and c.position[1] <= 0:
                    c.position = (c.position[0], c.rows - 1)
                else:
                    c.move(c.dirnx, c.dirny) #if it is not on the edge of each x y it
will continue moving just as it is told

    def reset(self, position): # to reset everything back to where and when it
started
        self.head = Cube(position)
        self.body = []
        self.body.append(self.head)
        self.turns = {}
        self.dirnx = 1
        self.dirny = 0

    def addCube(self):
        tail = self.body[-1]
        dx = tail.dirnx
        dy = tail.dirny
        if dx == 1 and dy == 0:     # if the last cube is moving to the right
            self.body.append(Cube((tail.position[0] - 1, tail.position[1]))) # adds a
cube 1 less than the current position of the last tail
        elif dx == -1 and dy == 0:
            self.body.append(Cube((tail.position[0] + 1, tail.position[1]))) #
basically same as above but in different positions
        elif dx == 0 and dy == 1:
            self.body.append(Cube((tail.position[0], tail.position[1] - 1)))
        elif dx == 0 and dy == -1:
            self.body.append(Cube((tail.position[0], tail.position[1] + 1)))
        self.body[-1].dirnx = dx    #for the tail to follow its head again
        self.body[-1].dirny = dy


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

So the above is the Snake class which is the snake that we are going to move. So the snake is made out of Cubes from the class above. But these cubes we have to keep an eye of their position constantly because they are moving. The function def move() is a

function made for the snake to move. So if I press the left button it will move the direction by changing the x and y. So if its moving left the x will be -1 because how pygame works is that the more you are moving left, the x will keep being substracted. While the y is 0 because its only moving horizontally which is left. The code below it is basically the same just different directions. Also there is an empty list called body and a dictionary called turns I declared earlier. Each time we press the key, we add a key value into the turns dictionary which is the self.head.position[:] and give it a value which is the direction x and direction y resulting from pressing the key. So when we turn it by pressing a key, it creates and adds to the turns dictionary/list.

Then in the for i, c in enumerate…… so for every index, cube in the body(get the index and the cube from the self.body list), for each cube grab the position in the variable "p". if the p is in the turns dict/list, the turn will be equal to the turns list at the index, then cube.move with the turn[0],turn[1] which is the direction x and y so it knows where to move. Then to finish it off if the index is len(self.body)-1 which is the length of the body -1 is the last cube on the snake, we remove the turn, because if we leave it, anytime we hit that position on the screen regardless if the snake is turning there or not, we will automatically change directions if we don't remove it from the dict/list.

But if it's not in the dict/list, we still have to move it because its constantly moving. We do that by writing the code in the else which probably will be easy enough to understand by reading the comments above in the code.

The def reset() function is for resetting the snake after it meets certain conditions like losing the game so that the snake went back to where it started.

The addCube() functionis basically to add cubes to the snake. So how it works depends on the direction it is turning. If its turning left which the horizontal direction of the tail is right, it adds a cube 1 less than current position of the last tail.

The def() draw is to draw the snake, i added a condition so that if the index is 0 which is the head, it will draw eyes, if its not then it will just draw regular cubes

```
def drawGrid(w, rows, surface):
    sizeBetween = w // rows  # how big each squares in the grid
    x = 0
    y = 0
    for i in range(rows):
        x = x + sizeBetween
        y = y+ sizeBetween
        pygame.draw.line(surface, (255, 255, 255), (x, 0), (x, w)) #draw column (x,0)
biar gambar ga geser ke bawah and (x,w) draw till the end of the y
        pygame.draw.line(surface, (255, 255, 255), (0, y), (w, y)) #draw rows (0,y)
biar gambar ga keser ke kanan and (w,y) draw till the end of the x
def redrawWindow(surface): # this function is for drawing everything in this code
    global rows, width, s
    screen.fill((0, 0, 0))
    s.draw(surface)
    food.draw(surface)
```

```
    food2.draw(surface)
    food3.draw(surface)
    for i in obstacles:
        i.draw(surface)
    drawGrid(width, rows, surface)
    pygame.display.update()


def randomFood(rows, s):
    positions = s.body

    while True:
        x = random.randrange(rows)
        y = random.randrange(rows)
        if len(list(filter(lambda c: c.position == (x,y), positions))) > 0:
            # if the length of the list of the filtered list of positions(the snake
body's), if the list of positions is the same as the randomly generated (x,y) then it
continue the loop until it finds the position which doesnt touch the snake
            # the > 0 means that if there really is something in the len() it means
that it is true and it returned a list
            continue
        else:
            break
    return (x, y)

def message_box(subject, content):
    root = tk.Tk()
    root.attributes("-topmost", True)
    root.withdraw()

    messagebox.showinfo(subject, content)
    try:
        root.destroy()
    except:
        pass

def createObstacles(obsAmount):
    newObstacles = []
    for i in range(obsAmount):
        newObstacles.append(Cube(randomFood(rows, s), color=(255, 255, 255)))
    return newObstacles
```

So these are functions that I wrote outside of the class. The first one is the drawGrid function which has 3 parameters: w, rows, surface. What it does is draw lines thorough the x and y to make squares like a grid. Then the redrawWindow(surface), is basically drawing everything that is present on the screen.

The def randomFood is to spawn food or any cube randomly in the grid. So I added positions is the list of the snake cubes or its body, and made a while loop. The loop spawns random in the position in the rows, if the length of the list of the filtered function is more than 0, it will skip and continue until it finds the position that is not the same as the snake body so that when we play the thing that we want to spawn doesn't spawn inside the snake's body. The filter is for returning a True value from a iterable with the help of the function. So in this case if the position of the cube is equal to the randomized

position with the iterable being positions(which is the body of the snake) then it will return a value which later I turn it to a list and using len I made it to int so if its bigger than zero we know that it exists and then we use continue so that it will loop through it again and again until it randomizes in a position which isn't in the body of the snake.

The def message_box() is basically just a function to write a message. I made this so that if a condition is met the message box will appear like what I wrote in the function.

The createObstacles() function is actually a function I created to simplify what I made earlier in my code which is the obstacles so that the code less spacious. So I made an empty list and then for i in range(obsAmount), we append a cube generated by the randomFood function and return its value. So if we made the obsAmount parameter 10 it will get 10 in the list above.

```
def main():
    global screen, width, rows, s, food, obstacle, food2, food3, position, obstacles
    obstacleAmount = 10
    width = 500
    rows = 20       # total of squares that are used in the game (empty cubes)
    screen = pygame.display.set_mode((width, width))
    pygame.display.set_caption("Nub Snake")
    s = Snake((0, 0, 255), (10, 10))
    food = Cube(randomFood(rows, s), color=(220, 20, 60))
    food2 = Cube(randomFood(rows, s), color=(255, 0, 0))
    food3 = Cube(randomFood(rows, s), color=(255, 0, 0))

    obstacles = createObstacles(obstacleAmount)

    clock = pygame.time.Clock()
    while True:
        clock.tick(10)              # to get 10 fps rather than more i tried if more
than that its too fast
        s.move()                    # call the move function for the snake to move
        if s.body[0].position == food.position:    # if the s.body[0].position which
is the head of the snake is the same as the food position which means that when the
head hits the food
            s.addCube()                            # call the addCube function for
the snake which is for the snake to grow longer as it eats food
            food = Cube(randomFood(rows, s), color=(220, 20, 60))# to add in an
another food after a food has been eaten in random cube
            food2 = Cube(randomFood(rows, s), color=(255, 0, 0))
            food3 = Cube(randomFood(rows, s), color=(255, 0, 0))
            obstacles = createObstacles(obstacleAmount)

        if s.body[0].position == food2.position:
            food = Cube(randomFood(rows, s), color=(220, 20, 60))
            food2 = Cube(randomFood(rows, s), color=(255, 0, 0))
            food3 = Cube(randomFood(rows, s), color=(255, 0, 0))

        if s.body[0].position == food3.position:
            food = Cube(randomFood(rows, s), color=(220, 20, 60))
            food2 = Cube(randomFood(rows, s), color=(255, 0, 0))
            food3 = Cube(randomFood(rows, s), color=(255, 0, 0)) for i in obstacles:
# for every things in the obstacles list
```

```
    if s.body[0].position == i.position: # if the head position is equal to the
position of things in the obstacles list which are the obstacles
        print("Score: " + str(len(s.body))) # print the score, the score is the
length of the snake's body
        message_box("You Lost!", "Try Again") # make a message box
        s.reset((10, 10)) # resets the snake in the position 10 10 which is in the
middle
        food = Cube(randomFood(rows, s), color=(220, 20, 60))
        food2 = Cube(randomFood(rows, s), color=(255, 0, 0))
        food3 = Cube(randomFood(rows, s), color=(255, 0, 0))
        obstacles = createObstacles(obstacleAmount)
    redrawWindow(screen) # redraw the whole thing

for x in range(len(s.body)): #loop through every cube in the snake body
    if s.body[x].position in list(map(lambda z:z.position, s.body[x + 1:])): #if the
position is in a list of all the position
        print("Score: " + str(len(s.body)))
        message_box("You Lost!", "Try Again")
        s.reset((10, 10))
        food = Cube(randomFood(rows, s), color=(220, 20, 60))# to add in an another
food after a food has been eaten in random cube
        food2 = Cube(randomFood(rows, s), color=(255, 0, 0))
        food3 = Cube(randomFood(rows, s), color=(255, 0, 0))
        obstacles = createObstacles(obstacleAmount) # rerandom obstacles
        break
redrawWindow(screen) # redraw the whole thing
```

The one above is my main loop. First I globaled all the variables that I use in my functions or in my main loop. Set the obstacle amount to 10 if you want it to be harder just add in more. Setting the screen, title and made objects which is the snake, the 3 foods, and the obstacles.

In the While loop, clock tick is to get 10 fps so that the game won't run too fast, try it on 60 fps and see what happens lol. And I move the snake by calling the function from my snake class. Then I add a few conditions regarding on how this game works.

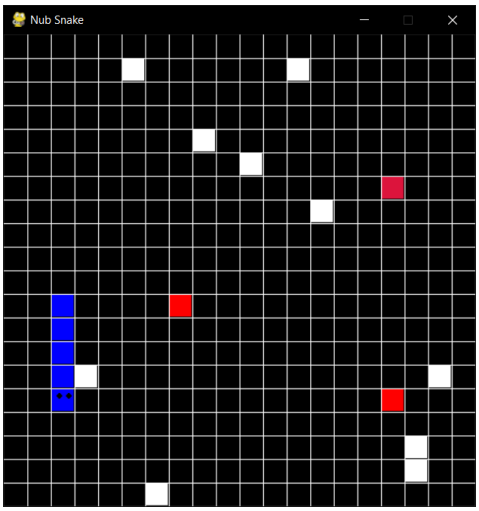
If the head position is equal to the food position which is the original food which will make you longer, I use the addCube function from the snake class to make the snake longer after it eats a food, then rerandomize the foods and obstacles. If it eats the other food which is the fake ones, it will only rerandom the food only but the obstacles still stay at the same positions.

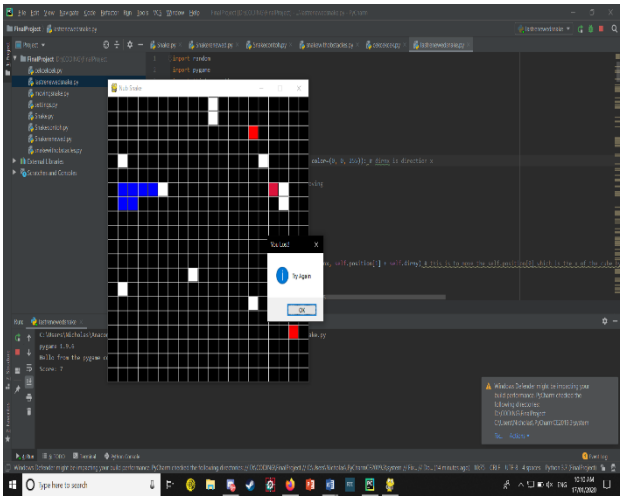
For every thing in the obstacles list which is all the obstacles, if the head position is equal to the things(obstacles)'s position, it will print out the score which is the length of the snake's body, make a message box and reset the snake with its position which is 10 10 which is in the middle of the screen, and rerandomize and redraw the whole thing.

The next one is basically the same just different condition, after looping through every cube in the snake body, if the position is in the position, to make it more understandable : when you play the game, if u move directly to your own body after having 3 or more scores, you will lose.

IV. How it works and evidence

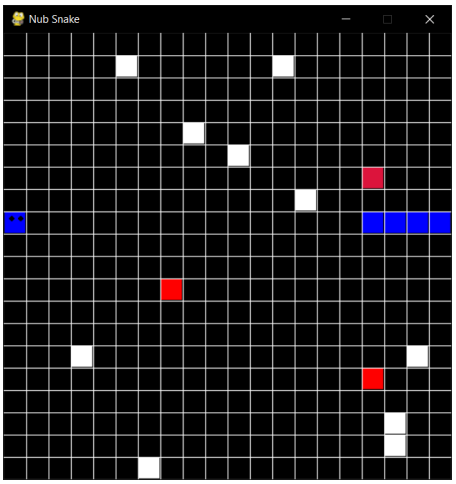
So when you play the program it opens a snake game. At the start the snake is already constantly moving by its own to the right direction so then you just need to press the arrow keys. Press up to go up, press left to go left, press right to go right, press down to go down. So there are 3 foods in the game, 2 of them are fakes, and 1 of them is the real food which will add your score and make your snake longer. They are all identical but I made the real one different so that people can compare it to the fake ones. There are also 10 obstacles but they are only cubes so that it will not be that hard. If you hit them you lose.

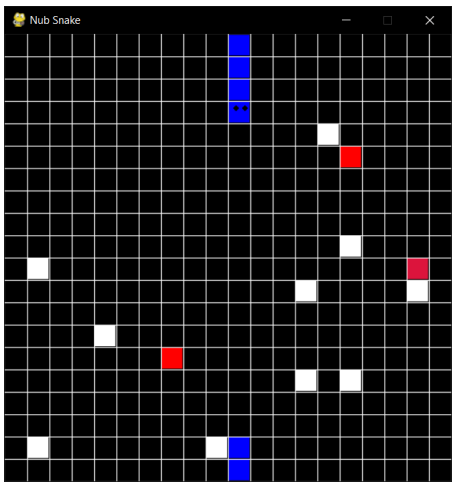


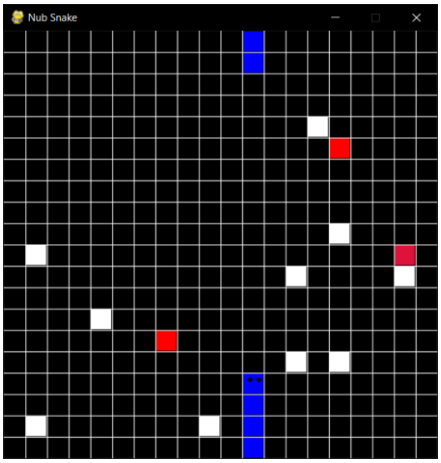


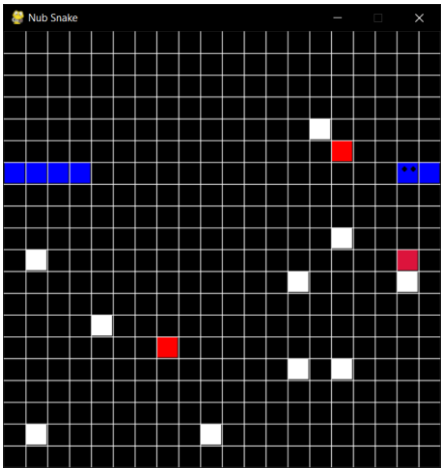
Losing will reset the game and the snake and re-random the positions of the foods and obstacles. If you get the real food it will also randomize all foods and obstacles again, but if you get the fake one it will only randomize the foods, the obstacles wont re-random because if it does it will be a bit too confusing to play the game.

If your snake moves to the end of the screen, you won't lose. Instead, you will be moved to the opposite side of the end of the screen. So if you move to the right till the end of the screen you will see yourself appearing at the start of the left side of the screen.

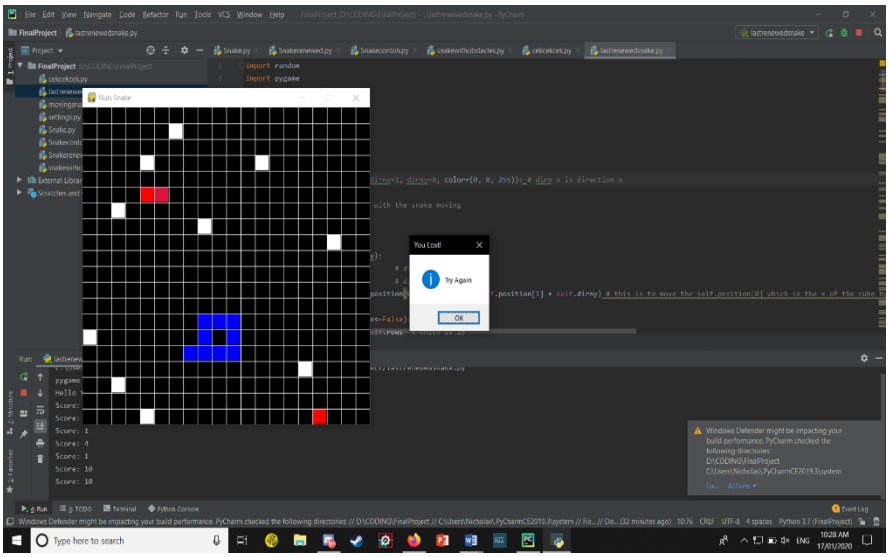


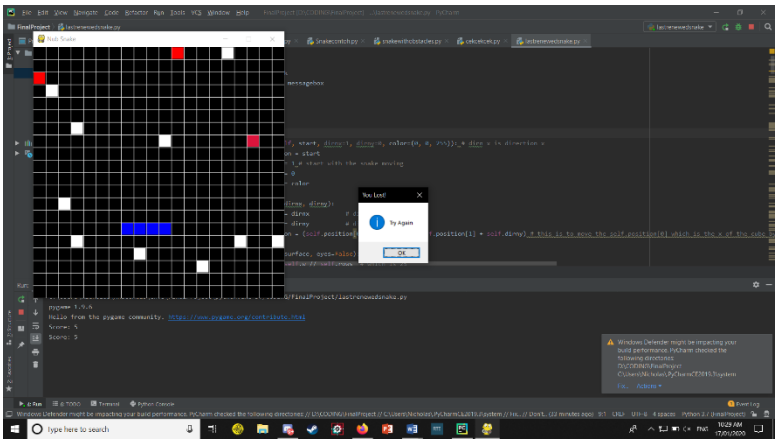






Players also can't turn to their own tail, if they do, the game will think that you just ate your own body and thus, you will lose if you do turn the opposite way from the direction you are currently turning to. And you shouldn't eat your own tail, you will also lose because of that, so the longer your snake is the harder the game will get. You will know your score after you lose, every time you lose it will print out your score in the IDE.





V. Problems that have been overcomed

At first I thought this project wouldn't be too hard because I didn't know how to implement it in codes. After I watched on how it works on the codes through the internet i actually thought that maybe I can't do this, but in the end I tried to stay focused and just do it. The position concept of this game I got confused early on but as I read and play it over and over again after each errors I managed to pull it off.
Not just early on though as I progressed through this project there is a bug which I really couldn't do it for probably like 3 days, so I searched for inspiration on the internet even though my code was different, I got the concept from it using functions that I've never used before.
From doing this project, I learned a lot from my first experience, like even though errors come after errors and so on, we just have to keep doing it, and this sharpened my feeling in coding even though I'm pretty new in programming

VI. Resources

https://www.youtube.com/watch?v=i6xMBig-pP4 this is for me to learn pygame generally
https://www.youtube.com/results?search_query=pygame+snake+tutorial a bit of inspiration from this video so I know how the game works

VII. Source Code

```
VIII.import random
      import pygame
      import tkinter as tk
      from tkinter import messagebox

      class Cube(object):
          rows = 20
          w = 500

          def __init__(self, start, dirnx=1, dirny=0, color=(0, 0, 255)): # dirnx
      is direction x
              self.position = start
              self.dirnx = 1 # start with the snake moving
              self.dirny = 0
              self.color = color

          def move(self, dirnx, dirny):
              self.dirnx = dirnx          # direction x
              self.dirny = dirny          # direction y
              self.position = (self.position[0] + self.dirnx, self.position[1] +
      self.dirny) # this is to move the self.position[0] which is the x of the
      cube by adding the direction that is given by pressing the key

          def draw(self, surface, eyes=False):
              distance = self.w // self.rows  # which is 25
              i = self.position[0]            # row ( the x )
```

```
        j = self.position[1]            # column ( the y )
        pygame.draw.rect(surface, self.color, (i * distance + 1, j *
distance + 1, distance - 2, distance - 2)) #distance -2 is the area, i *
distance + 1 is x y
        if eyes:
            center = distance // 2 # the center of the cube
            radius = 3 # radius of the eye
            eye1 = (i * distance + center - radius, j * distance + 8) # i *
distance is the position x + center - radius is the position of the circle
inside the cube
            eye2 = (i * distance + distance - radius * 2, j * distance + 8)
# same as top but the second eye
            pygame.draw.circle(surface, (0, 0, 0), eye1, radius) # draws
the eye
            pygame.draw.circle(surface, (0, 0, 0), eye2, radius) # draws
the second eye
    # def drawobstacle(self, surface):
    #     distance = self.w // self.rows
    #     distancerect = (self.w // self.rows) * 2  <------ NEVER MIND THIS
CODE I FAILED
    #     k = self.position[0]
    #     l = self.position[1]
    #     pygame.draw.rect(surface, self.color, (k * distance + 1, l *
distance + 1, distancerect // 2, distancerect * 1.5 - 2))


class Snake(object):
    body = []       #list of cubes that will be the body
    turns = {}      #this is where i store the positions of the head of the
snake and why is below

    def __init__(self, color, position):
        self.color = color
        self.head = Cube(position)      #this is so we know the head's
position at all times
        self.body.append(self.head)     #append the head to the first list
of the body
        self.dirnx = 1                  #so when the game starts it
immediately moves first before the given command not just stay still
        self.dirny = 0

    def move(self):
        for event in pygame.event.get():        # so we can close the game
            if event.type == pygame.QUIT:
                pygame.quit()

            keys = pygame.key.get_pressed()
            for key in keys:
                if keys[pygame.K_LEFT]:     # if the left key is pressed
then it moves left by moving the x -1 and y 0
                    self.dirnx = -1
                    self.dirny = 0
                    self.turns[self.head.position[:]] = [self.dirnx,
self.dirny] # to remember the way the head turns so that the tail can also
turn after the main head has turned directions
```

```
                    # so the key is the current position of the head of the
snake, and it is equal to what direction it is turning to
                    # so when we turn it creates and adds it to the turns
list
                elif keys[pygame.K_RIGHT]:
                    self.dirnx = 1          # if the right key is pressed
then it moves left by adding the x 1 and y 0
                    self.dirny = 0
                    self.turns[self.head.position[:]] = [self.dirnx,
self.dirny] # the same as the top

                elif keys[pygame.K_UP]:     # if the up key is pressed then
it moves left by moving the x 0 and y -1
                    self.dirnx = 0
                    self.dirny = -1
                    self.turns[self.head.position[:]] = [self.dirnx,
self.dirny]
                elif keys[pygame.K_DOWN]:   # if the down key is pressed
then it moves left by moving the x 0 and y -1
                    self.dirnx = 0
                    self.dirny = 1
                    self.turns[self.head.position[:]] = [self.dirnx,
self.dirny]

        for i, c in enumerate(self.body):   #look through the list of
positions that we have on the snake, i is the index and c is cube
            pos = c.position[:] # for each cube, grabs the position copy
all the elements in the position index
            if pos in self.turns:# if the position is in the turns
dictionary
                turn = self.turns[pos]# the turn that we chose the turn
list at the index
                c.move(turn [0], turn[1])# give the cube the direction x
and y after we pressed a direction button
                if i == len(self.body) - 1:# if we are in the last cube
                    self.turns.pop(pos)# if we dont remove the turn the
turn will activate when we hit the position that was in our last tail
            else:
                if c.dirnx == -1 and c.position[0] <= 0: #if snake is
moving left and the position x is less than or equal to 0
                    c.position = (c.rows - 1, c.position[1]) #change the
position to the right side of the screen
                elif c.dirnx == 1 and c.position[0] >= c.rows - 1:
                    c.position = (0, c.position[1])
                elif c.dirny == 1 and c.position[1] >= c.rows - 1:
                    c.position = (c.position[0], 0)
                elif c.dirny == -1 and c.position[1] <= 0:
                    c.position = (c.position[0], c.rows - 1)
                else:
                    c.move(c.dirnx, c.dirny) #if it is not on the edge of
each x y it will continue moving just as it is told

    def reset(self, position): # to reset everything back to where and when
it started
        self.head = Cube(position)
```

```
            self.body = []
            self.body.append(self.head)
            self.turns = {}
            self.dirnx = 1
            self.dirny = 0

        def addCube(self):
            tail = self.body[-1]
            dx = tail.dirnx
            dy = tail.dirny
            if dx == 1 and dy == 0:     # if the last cube is moving to the
    right
                self.body.append(Cube((tail.position[0] - 1,
    tail.position[1]))) # adds a cube 1 less than the current position of the
    last tail
            elif dx == -1 and dy == 0:
                self.body.append(Cube((tail.position[0] + 1,
    tail.position[1]))) # basically same as above but in different positions
            elif dx == 0 and dy == 1:
                self.body.append(Cube((tail.position[0], tail.position[1] -
    1)))
            elif dx == 0 and dy == -1:
                self.body.append(Cube((tail.position[0], tail.position[1] +
    1)))
            self.body[-1].dirnx = dx    #for the tail to follow its head again
            self.body[-1].dirny = dy


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

    def drawGrid(w, rows, surface):
        sizeBetween = w // rows  # how big each squares in the grid
        x = 0
        y = 0
        for i in range(rows):
            x = x + sizeBetween
            y = y+ sizeBetween
            pygame.draw.line(surface, (255, 255, 255), (x, 0), (x, w)) #draw
    column (x,0) biar gambar ga geser ke bawah and (x,w) draw till the end of
    the y
            pygame.draw.line(surface, (255, 255, 255), (0, y), (w, y)) #draw
    rows (0,y) biar gambar ga keser ke kanan and (w,y) draw till the end of the
    x
    def redrawWindow(surface): # this function is for drawing everything in
    this code
        global rows, width, s
        screen.fill((0, 0, 0))
        s.draw(surface)
```

```
        food.draw(surface)
        food2.draw(surface)
        food3.draw(surface)
        for i in obstacles:
            i.draw(surface)
        drawGrid(width, rows, surface)
        pygame.display.update()


    def randomFood(rows, s):
        positions = s.body

        while True:
            x = random.randrange(rows) # randomize
            y = random.randrange(rows) # randomize
            if len(list(filter(lambda c: c.position == (x,y), positions))) > 0:
                # if the length of the list of the filtered list of
    positions(the snake body's), if the list of positions is the same as the
    randomly generated (x,y) then it continue the loop until it finds the
    position which doesnt touch the snake
                # the > 0 means that if there really is something in the len()
    it means that it is true and it returned a list
                continue
            else:
                break
        return (x, y)

    def message_box(subject, content):
        root = tk.Tk()
        root.attributes("-topmost", True) # shows the text on the top
        root.withdraw()

        messagebox.showinfo(subject, content)
        try:
            root.destroy()
        except:
            pass

    def createObstacles(obsAmount):
        newObstacles = []
        for i in range(obsAmount):
            newObstacles.append(Cube(randomFood(rows, s), color=(255, 255,
    255)))
        return newObstacles

    def main():
        global screen, width, rows, s, food, obstacle, food2, food3, position,
    obstacles
        obstacleAmount = 10
        width = 500
        rows = 20       # squares that are used in the game in a single line so
    the total should be 20x20 = 400 squares (empty cubes)
        screen = pygame.display.set_mode((width, width)) # set the screen
        pygame.display.set_caption("Nub Snake") # write the title
        s = Snake((0, 0, 255), (10, 10)) # make the snake object
```

```
    food = Cube(randomFood(rows, s), color=(220, 20, 60)) # food object
    food2 = Cube(randomFood(rows, s), color=(255, 0, 0))
    food3 = Cube(randomFood(rows, s), color=(255, 0, 0))
    obstacles = createObstacles(obstacleAmount)
    clock = pygame.time.Clock() # this is for in the while loop
    while True:
        clock.tick(10)              # to get 10 fps rather than more i
tried if more than that its too fast
        s.move()                    # call the move function for the snake
to move
        if s.body[0].position == food.position:    # if the
s.body[0].position which is the head of the snake is the same as the food
position which means that when the head hits the food
            s.addCube()                            # call the addCube
function for the snake which is for the snake to grow longer as it eats
food
            food = Cube(randomFood(rows, s), color=(220, 20, 60))# to add
in an another food after a food has been eaten in random cube
            food2 = Cube(randomFood(rows, s), color=(255, 0, 0))
            food3 = Cube(randomFood(rows, s), color=(255, 0, 0))
            obstacles = createObstacles(obstacleAmount) # to rerandom the
obstacles's position

        if s.body[0].position == food2.position:  # if the head hits the
food2 it will rerandom again but the obstacles wont be randomed only the
food
            food = Cube(randomFood(rows, s), color=(220, 20, 60))
            food2 = Cube(randomFood(rows, s), color=(255, 0, 0))
            food3 = Cube(randomFood(rows, s), color=(255, 0, 0))

        if s.body[0].position == food3.position:  # same as food2
            food = Cube(randomFood(rows, s), color=(220, 20, 60))
            food2 = Cube(randomFood(rows, s), color=(255, 0, 0))
            food3 = Cube(randomFood(rows, s), color=(255, 0, 0))

        for i in obstacles: # for every things in the obstacles list
            if s.body[0].position == i.position: # if the head position is
equal to the position of things in the obstacles list which are the
obstacles
                print("Score: " + str(len(s.body))) # print the score, the
score is the length of the snake's body
                message_box("You Lost!", "Try Again") # make a message box
                s.reset((10, 10)) # resets the snake in the position 10 10
which is in the middle
                food = Cube(randomFood(rows, s), color=(220, 20, 60))
                food2 = Cube(randomFood(rows, s), color=(255, 0, 0))
                food3 = Cube(randomFood(rows, s), color=(255, 0, 0))
                obstacles = createObstacles(obstacleAmount)
            redrawWindow(screen) # redraw the whole thing

        for x in range(len(s.body)): #loop through every cube in the snake
body
            if s.body[x].position in list(map(lambda z:z.position, s.body[x
+ 1:])): #if the position is in a list of all the position
                print("Score: " + str(len(s.body)))
```

```
                    message_box("You Lost!", "Try Again")
                    s.reset((10, 10))
                    food = Cube(randomFood(rows, s), color=(220, 20, 60))# to
    add in an another food after a food has been eaten in random cube
                    food2 = Cube(randomFood(rows, s), color=(255, 0, 0))
                    food3 = Cube(randomFood(rows, s), color=(255, 0, 0))
                    obstacles = createObstacles(obstacleAmount) # rerandom
    obstacles
                    break
            redrawWindow(screen) # redraw the whole thing

    main()
```