# CHAPTER :- 1
# INTRODUCTION

## 1.1 Introduction

The Python Snake Game stands as a timeless classic in the realm of arcade gaming, capturing the essence of simplicity, strategy, and addictive gameplay. In this report, we delve into the design and implementation of a Snake Game using the Python programming language and the Pygame library, showcasing the process of bringing this iconic game to life in a modern development environment.

At its core, the Snake Game revolves around the concept of guiding a snake across a bounded playing field, manoeuvring it to consume food pellets scattered throughout the environment. With each pellet consumed, the snake grows longer, presenting both opportunities and challenges as players strive to navigate an increasingly crowded and hazardous landscape.

## 1.2 About Snake game

Taneli Armanto's creation of the Snake game in 1997 indeed marked a significant milestone in mobile gaming history. It's fascinating how the concept was adapted from the arcade game "Blockade" and evolved into a mobile phenomenon. The simplicity yet addictive nature of the game contributed to its widespread popularity, especially after it was preloaded on Nokia devices in 1998. This move undoubtedly expanded its reach to a larger audience, contributing to a resurgence of interest in the Snake concept.

The recognition by Next Generation, ranking it number 41 on their "Top 100 Games of All Time" list in 1996, highlights its enduring appeal. The combination of quick reactions and strategic thinking required to excel at the game likely contributed to its enduring popularity.

It's intriguing to note that the Museum of Modern Art in New York City acknowledged the cultural significance of the Nokia port of Snake by expressing interest in adding it to their collection in the future, underscoring its impact on both gaming and popular culture. This recognition further solidifies its status as a timeless classic in the gaming world.

# CHAPTER :- 2
# TOOLS USED

## 2.1 PY-GAME Module

The Pygame library is an open-source module for Python designed to facilitate game and multimedia application development. It leverages the Simple DirectMedia Layer (SDL) for cross-platform compatibility, allowing developers to create games that run on various operating systems. Pygame abstracts away backend complexities, enabling users to focus on game logic and graphics. With Pygame, developers can handle graphics, sound, and user input easily, thanks to its comprehensive set of modules and event handling mechanisms. The library boasts a supportive community, offering ample resources like forums, documentation, and tutorials for both beginners and experienced developers. Overall, Pygame streamlines the game development process, empowering developers to create engaging games without the hassle of low-level programming.

## 2.2 TIME Module

The Python `time` module is a versatile tool that serves multiple purposes beyond just representing time. While it does offer various ways to handle time data, such as objects, numbers, and strings, its utility extends to other important functionalities in programming.

For instance, one common use of the `time` module is to introduce delays or pauses in code execution. This is particularly useful when you need to synchronize actions, wait for user input, or simulate real-time behavior in your programs. The `time.sleep()` function, for example, allows you to pause execution for a specified number of seconds, providing a simple and effective way to control the timing of your code's operations.

In summary, while the `time` module in Python certainly offers robust support for handling time-related data, its capabilities extend beyond mere time representation. From controlling code execution to measuring performance and facilitating time conversions, the `time` module proves to be a valuable asset for developers across a wide range of programming scenarios.

## 2.3 RANDOM Module

The Python **random** module is a fundamental component of the Python standard library, serving as a tool for generating pseudo-random numbers. It's important to note that while these numbers appear random, they're actually generated using deterministic algorithms known as pseudo-random number generators (PRNGs). Despite not being truly random, PRNGs are highly useful for a wide range of applications where randomness is required, such as simulations, games, cryptography, and statistical analysis.

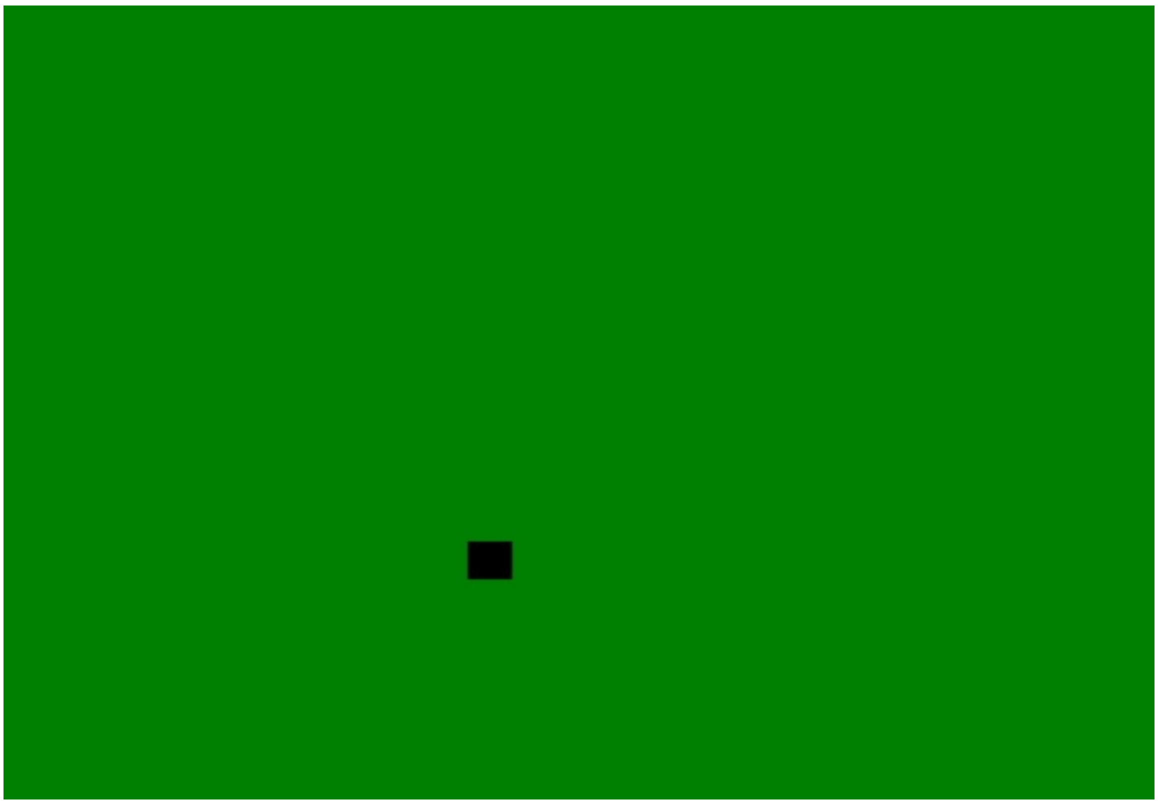
# CHAPTER :- 3
# VARIOUS SCREENSHOTS

## 3.1 FIRST STEP WAS TO CREATE THE GAME SCREEN

This step includes the orientation and size of output window. It also includes physical properties of title bar i.e. name of game, its color, etc. The background color of output window is applied here. Below is the generated screen with black as background color, landscape view, height*width as 600*800units.

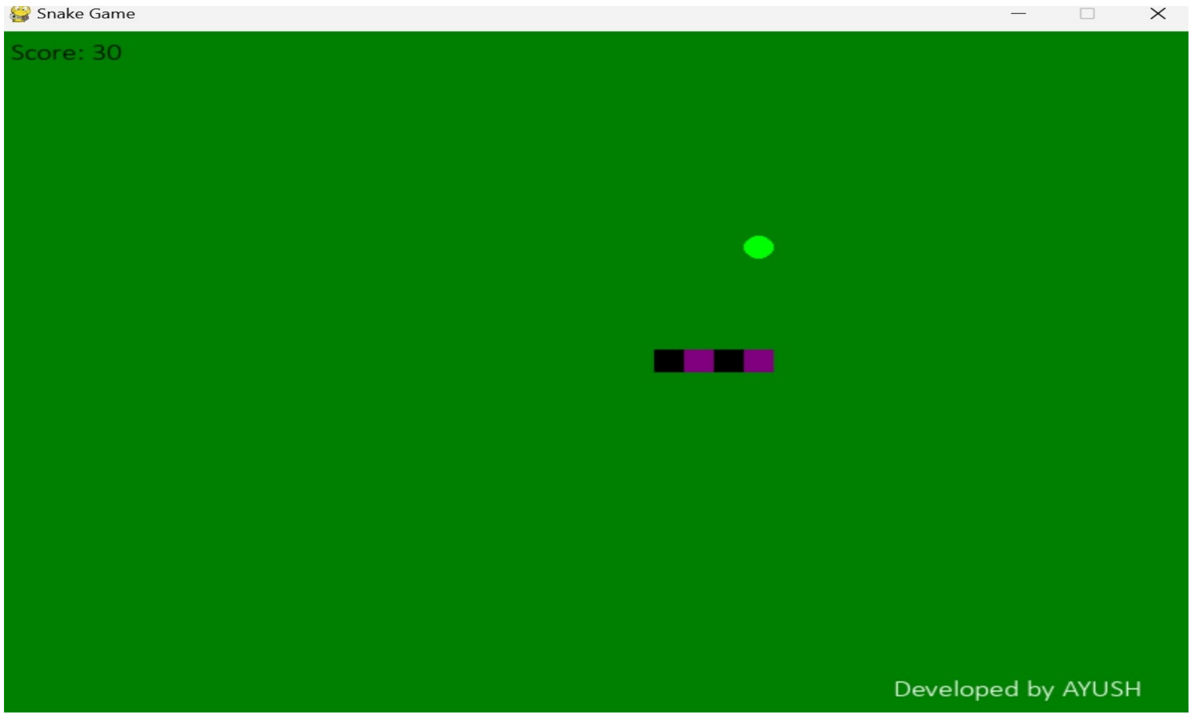
## 3.2 SECOND STEP WAS TO ADD THE SNAKE

In this step we have added snake and its navigations. The color of snake, its initial length and thickness, speed, etc are added in this step. The navigation properties are also generated during the implementation of this step. Here the blue color dot denotes the snake.



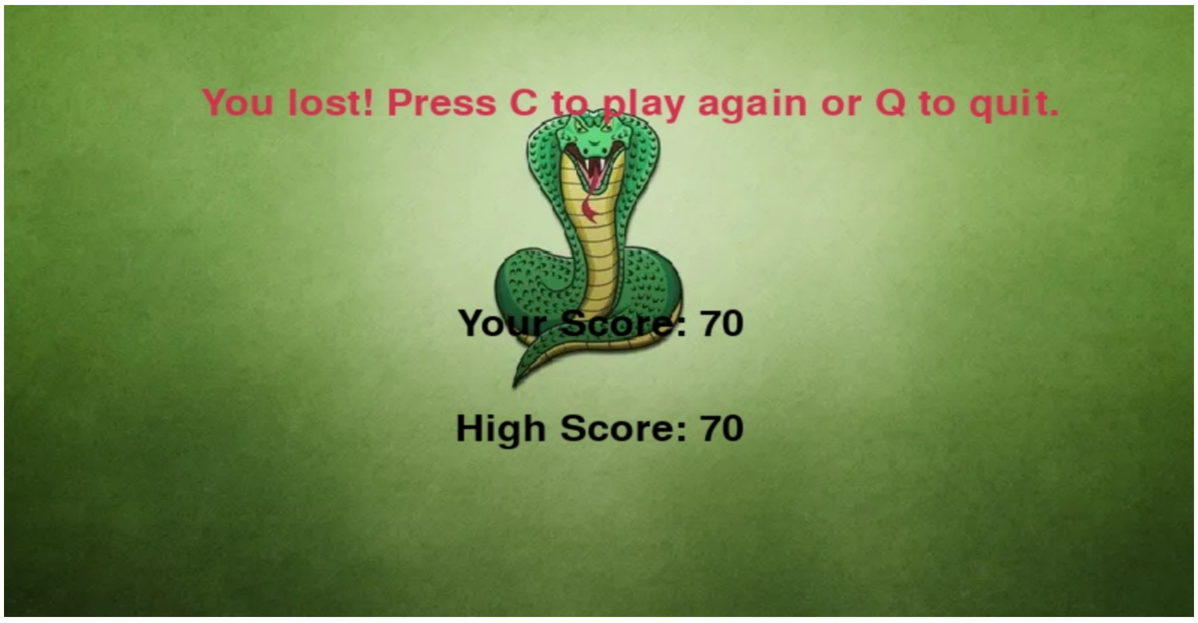
## 3.3 THIRD STEP WAS TO ADD THE FOOD

The eating element for snake is added in this step. Its physical properties are added here. Its random generation and placement is taken care. And the score is also maintained in this step because both food and score go hand in hand. As per eat score is awarded. Here in picture blue is food and black is snake.



## 3.4 FOURTH STEP WAS TO DISPLAY THE SCORE

In this step, when the game is over the points are displayed. This also includes the termination condition off the game. Here some shortcut keys are designed such as press p to play again or to start the game, press e to exit the screen. Here in the picture color yellow represents the score and the red one show the shortcut keys.



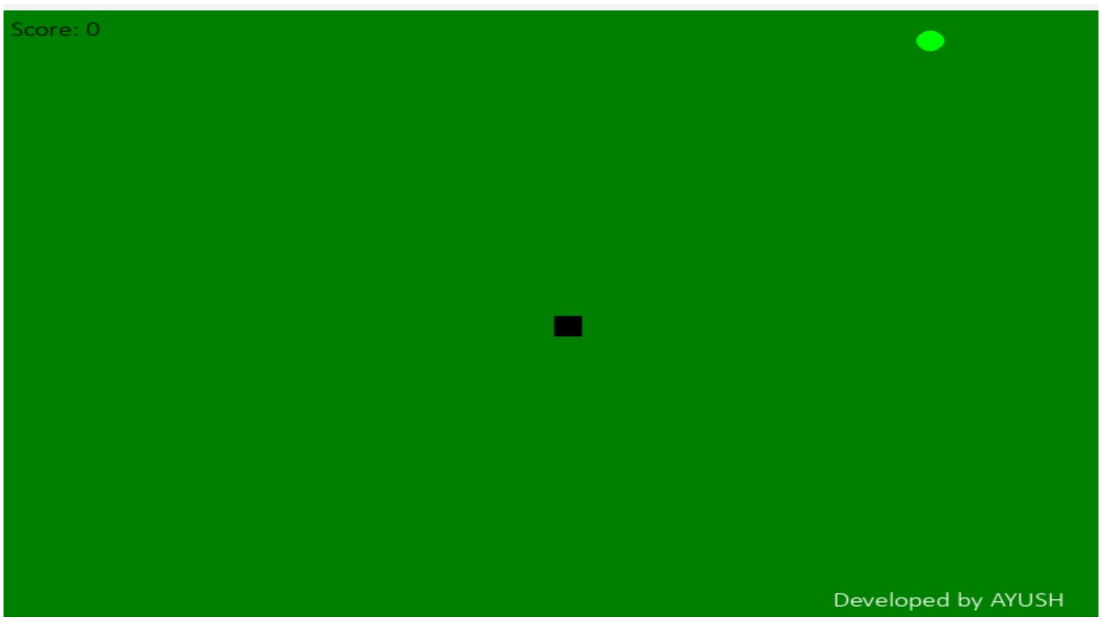
You lost! Press C to play again or Q to quit.

Your Score: 70

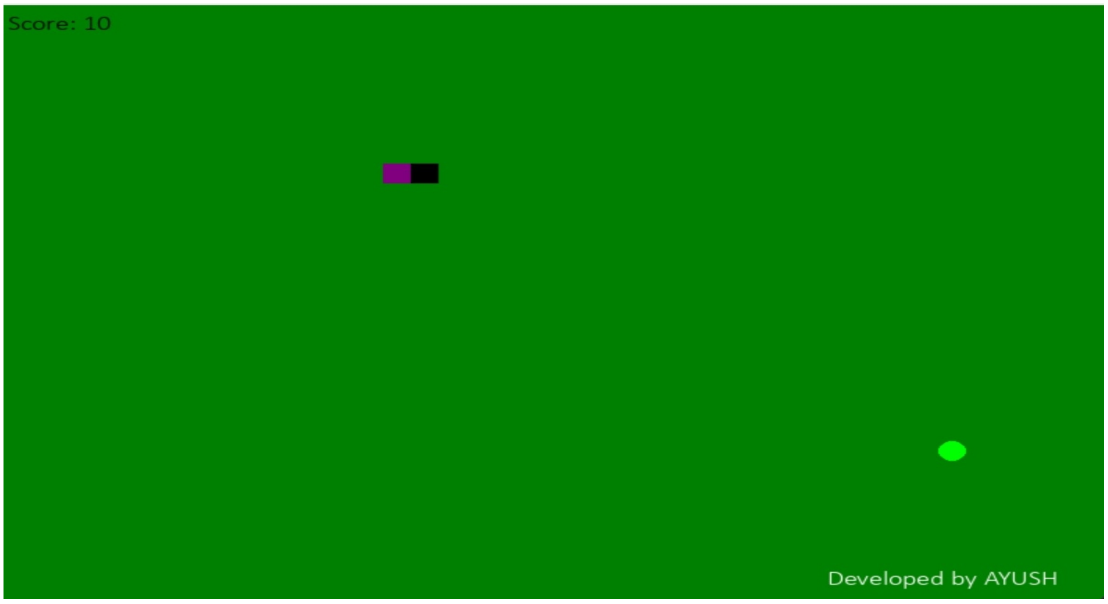
High Score: 70

## 3.5 WORKING SCREEN SHOTS OF GAME

SNAKES INITIAL LENGTH IS 1:



SNAKES LENGTH CHANGED WITH POINTS



above mentioned screenshots are of working game. In the first picture the length of snake was by default. That was the starting of game and the score was zero. Bar in the picture #2 the score was four and the length of the snake has been increased. If we play further the same mechanism will take place tell the termination condition that is if snake touches the wall or it's own body the game will stop and the score will be displayed.

# CHAPTER :- 4
# LOGIC OF THE GAME

## 4.1 Logic of the Game

There are multiple ways to implement a Snake game. I'll explain a classical implementation, and another modern implementation that uses modern programming languages features.

## 4.2 Implementation

The map here is represented using a 2D array of some sort, this is what the numbers mean in the map:

0 = empty

>0 = the snake

-1 = mouse

You can map additional entities to number as long as they're negative numbers, like walls or items

You're keeping track of the snake's x and y within the map, the direction it's going at, and the snake's length

Algorithm (repeated every second):

- Calculate x and y variables in the new direction (if direction is right, increase x by 1)
- If the new x and y are beyond boundaries, correct them.
- If value of new element at new x and y > 0, it's a game over (snakebit itself)
- Iterate over map, if element is > 0, increase it by 1. If it's better than snake's length, place a 0 instead.

  So in the map we showed earlier it'll look like this at this step, after the old 4 becomes 5, it'll be greater than the length, so it'll be cut off and be replaced with a 0.
- If value of new element at new x and y = -1, increase length by 1, that way next turn the length will be represented because the tail won't be cut off.
- Place 1 at new X and Y. So if the direction here was "right":
- Place a -1 somewhere random in the map if the mouse was eaten,use a while loop so that if the element at the position determined by the random x and y is >0, it generates another x and y until it finds an empty spot.

# CHAPTER :- 5 CONCLUSION AND REFERENCES

## 5.1 CONCLUSION

It is our teams hope that this document will be of huge help with understanding of our projects we have used  an approach which has proved beneficial in terms of our learning this language.

The coding of Snake was extremely difficult with many errors arising. Many systems had to be written numerous ways before a final working solution was found.  it is recommended that anyone who wishes to recreate this game starts simply when writing the code. It is advisable that they first perfect the snake movement controls before messing with the food generation. By taking the code in small sections, it is easier to get individual features to work. Building off this, use functions to contain each aspect of the game. Using functions made it easier to determine where errors were occurring when debugging the code. It also kept the code more organized.

## 5.2 REFERENCES

Geeks for Geeks: https://www.geeksforgeeks.org/create-a-snake-game-using-turtle-in-python/

Python Guides: https://pythonguides.com/snake-game-in-python/

Stack Overflow: https://stackoverflow.com/

Full Stack Python - Best Python Resources: https://www.fullstackpython.com/best-python-resources.html

# CHAPTER :- 6
# APPENDIX

## 6.1 CODE OF PROJECT

```
import pygame
import time
import random

pygame.init()
pygame.font.init()  # Initialize font module

# Load background music
pygame.mixer.music.load("D:\Subway_Surfers_Theme_V2-646327.mp3")
pygame.mixer.music.play(-1)

# Colors
white = (255, 255, 255)
yellow = (255, 255, 102)
black = (0, 0, 0)
red = (213, 50, 80)  # Reverting snake color back to its original color
green = (0, 255, 0)
blue = (50, 153, 213)
orange = (255, 165, 0)
purple = (128, 0, 128)
pink = (255, 192, 203)  # Changing background color from pink to green
bg_color = (0, 128, 0)  # New background color (green)
blink_colors = [(255, 0, 0), (255, 255, 0), (0, 255, 255)]

# Display
dis_width = 800
dis_height = 600
dis = pygame.display.set_mode((dis_width, dis_height))
pygame.display.set_caption('Snake Game')

# Load wallpapers
start_wallpaper = pygame.image.load("D:\close-up-snake-natural-habitat.jpg")
end_wallpaper = pygame.image.load("D:\cobra-snake-game-character-vbu0895huxqiek5p.jpg")
start_wallpaper = pygame.transform.scale(start_wallpaper, (dis_width, dis_height))
end_wallpaper = pygame.transform.scale(end_wallpaper, (dis_width, dis_height))

# Clock
clock = pygame.time.Clock()

# Snake and food size
snake_block = 20
snake_speed = 10

# Font Style
font_style = pygame.font.SysFont(None, 40)
small_font = pygame.font.SysFont(None, 20)
script_font = pygame.font.Font(pygame.font.match_font('calibri'), 20)

def text_objects(text, font):
    text_surface = font.render(text, True, black)
    return text_surface, text_surface.get_rect()
```

```
def message(msg, color, y_displace=0, x_pos=None, y_pos=None):
    mesg = font_style.render(msg, True, color)
    if x_pos is None or y_pos is None:
        dis.blit(mesg, [dis_width / 6, dis_height / 3 + y_displace])
    else:
        dis.blit(mesg, [x_pos - mesg.get_width() / 2, y_pos - mesg.get_height() / 2 + y_displace])

def pause_game():
    paused = True
    message("Paused. Press C to continue or Q to quit.", white, 10)
    pygame.display.update()
    while paused:
        for event in pygame.event.get():
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_c:
                    paused = False
                elif event.key == pygame.K_q:
                    pygame.quit()
                    quit()
        clock.tick(5)

def button(msg, x, y, w, h, ic, ac, action=None):
    mouse = pygame.mouse.get_pos()
    click = pygame.mouse.get_pressed()

    if x + w > mouse[0] > x and y + h > mouse[1] > y:
        pygame.draw.rect(dis, ac, (x, y, w, h))
        if click[0] == 1 and action is not None:
            action()
    else:
        pygame.draw.rect(dis, ic, (x, y, w, h))

    small_text = pygame.font.SysFont(None, 20)
    text_surf, text_rect = text_objects(msg, small_text)
    text_rect.center = ((x + (w / 2)), (y + (h / 2)))
    dis.blit(text_surf, text_rect)

def start_screen():
    intro = True
    while intro:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                quit()
            elif event.type == pygame.KEYDOWN:
                if event.key == pygame.K_RETURN:
                    intro = False

        dis.blit(start_wallpaper, (0, 0))
        message("Welcome to My Snake Game Everyone ", green, -100)
        message("Press Enter to start the game", red, 300)
        pygame.display.update()

def end_screen(score, high_score):
    outro = True
    while outro:
        dis.blit(end_wallpaper, (0,0))
        message("You lost! Press C to play again or Q to quit.", red)
        message("Your Score: " + str(score), black, 30, dis_width / 2, dis_height / 2 + 50)
        message("High Score: " + str(high_score), black, 60, dis_width / 2, dis_height / 2 + 100)
        pygame.display.update()

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                quit()
            elif event.type == pygame.KEYDOWN:
                if event.key == pygame.K_q:
                    pygame.quit()
                    quit()
                elif event.key == pygame.K_c:
                    return True  # Return True to restart the game loop

    return False

def gameLoop():
    start_screen()

    while True:  # Loop until the game is exited
        game_over = False
        game_close = False

        # Initial snake position
        x1 = dis_width / 2
        y1 = dis_height / 2

        # Movement changes
        x1_change = 0
        y1_change = 0

        # Snake body
        snake_List = []
        Length_of_snake = 1

        # Food position
        foodx = round(random.randrange(0, dis_width - snake_block) / 20.0) * 20.0
        foody = round(random.randrange(0, dis_height - snake_block) / 20.0) * 20.0
        blink = False
        blink_count = 0

        # Score
        score = 0
        high_score = 0  # Initialize high score

        while not game_over:

            while game_close:
                # If the end screen returns True, restart the game loop
                if end_screen(score, high_score):
                    gameLoop()  # Restart the game loop

            for event in pygame.event.get():
```

```
            if event.type == pygame.QUIT:
                game_over = True
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_LEFT and x1_change == 0:
                    x1_change = -snake_block
                    y1_change = 0
                elif event.key == pygame.K_RIGHT and x1_change == 0:
                    x1_change = snake_block
                    y1_change = 0
                elif event.key == pygame.K_UP and y1_change == 0:
                    y1_change = -snake_block
                    x1_change = 0
                elif event.key == pygame.K_DOWN and y1_change == 0:
                    y1_change = snake_block
                    x1_change = 0
                elif event.key == pygame.K_p:
                    pause_game()

        # Border crossing
        if x1 >= dis_width:
            x1 = 0
        elif x1 < 0:
            x1 = dis_width - snake_block
        elif y1 >= dis_height:
            y1 = 0
        elif y1 < 0:
            y1 = dis_height - snake_block

        x1 += x1_change
        y1 += y1_change
        dis.fill(bg_color)

        if blink:
             pygame.draw.circle(dis, random.choice(blink_colors), (int(foodx + snake_block / 2), int(foody +
snake_block / 2)), int(snake_block / 2))
            blink_count += 1
            if blink_count == 5:
                blink = False
                blink_count = 0
        else:
              pygame.draw.circle(dis, green, (int(foodx + snake_block / 2), int(foody + snake_block / 2)),
int(snake_block / 2))

        snake_head = []
        snake_head.append(x1)
        snake_head.append(y1)
        snake_List.append(snake_head)
        if len(snake_List) > Length_of_snake:
            del snake_List[0]

        for x in snake_List[:-1]:
            if x == snake_head:
                game_close = True

        # Snake drawing
        for idx, segment in enumerate(snake_List):
            color = black if idx % 2 == 0 else purple
            pygame.draw.rect(dis, color, [segment[0], segment[1], snake_block, snake_block])

        # Score display
        score_text = script_font.render("Score: " + str(score), True, black)
        dis.blit(score_text, [10, 10])

        # AYUSH font style
        ayush_text = script_font.render("Developed by AYUSH", True, white)
        dis.blit(ayush_text, [dis_width - 200, dis_height - 30])

        pygame.display.update()

        # Update high score
        if score > high_score:
            high_score = score

        # Food eating
        if x1 == foodx and y1 == foody:
            foodx = round(random.randrange(0, dis_width - snake_block) / 20.0) * 20.0
            foody = round(random.randrange(0, dis_height - snake_block) / 20.0) * 20.0
            Length_of_snake += 1
            score += 10  # Increment score when food is eaten
            blink = True

        clock.tick(snake_speed)

    # Display end screen after game over
    if not end_screen(score, high_score):
        break  # Exit the game loop if the player chooses to quit

pygame.quit()
quit()

gameLoop()
```