SNAKE GAME USING PYTHON

LIBRARY USED –

TURTLE

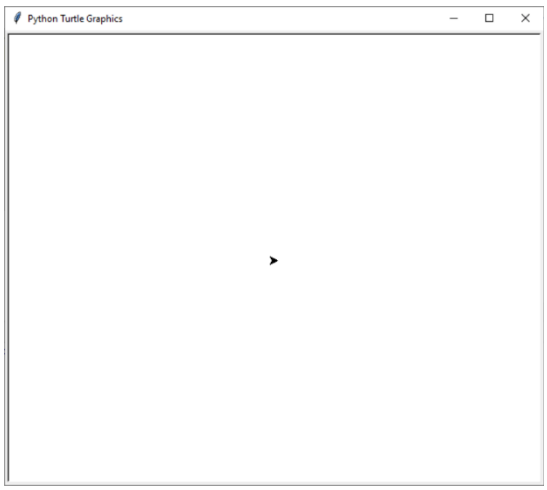
"Turtle" is a Python feature like a drawing board, which lets us command a turtle to draw all over it! We can use functions like turtle.forward(…) and turtle.right(…) which can move the turtle around.

I. How to use turtle library –

We can use turtle library by importing it using **import turtle** command. Now we can access all the functions and methods.

II. Now we need to create a dedicated window where we carry out each drawing command.

s = turtle.getscreen()



III. Now we can use the following functions of turtle library to perform different tasks.

- **forward(*distance*) or turtle.fd(*distance*) -** It moves the turtle in the forward direction by a certain distance. It takes one parameter **distance,** which can be an integer or float.
- **back(distance)** or **turtle.bk** or **turtle.backward(distance) -** This method moves the turtle in the opposite direction the turtle is headed. It doesn't change the turtle heading.
- **right(angle)** or **turtle.rt(angle) -** This method moves the turtle right by ***angle*** units.
- **left(angle)** or **turtle.lt(angle) -** This method turn the turtle left by ***angle*** units.
- **goto(*x, y=None*) or turtle.setpos(*x, y=None*) turtle.setposition(*x, y=None*) -** This method is used to move the turtle in the other area on the screen. It takes the two coordinates **- x and y**.
- **turtle.xcor( ) -** Return the turtle's **x** coordinate.
- **turtle.ycor( ) -** Return the turtle's **y** coordinate.
- **turtle.pendown() or turtle.pd() or turtle.down() -** Pull the pen down – drawing when moving.
- **turtle.penup() or turtle.pu() or turtle.up() -** Pull the pen up – no drawing when moving.
- **turtle.color(**args*) -** Return or set pencolor and fillcolor.
- **turtle.begin_fill() -** To be called just before drawing a shape to be filled.
- **turtle.end_fill() -** Fill the shape drawn after the last call to begin_fill()
- **turtle.hideturtle() or turtle.ht() -** Make the turtle invisible. It's a good idea to do this while you're in the middle of doing some complex drawing, because hiding the turtle speeds up the drawing observably.
- **turtle.showturtle() or turtle.st() -** Make the turtle visible.
- **turtle.shape(*name*) -** Set turtle shape to shape with given *name* or, if name is not given, return name of current

shape. Shape with *name* must exist in the TurtleScreen's shape dictionary. Initially there are the following polygon shapes: "arrow", "turtle", "circle", "square", "triangle", "classic".

- **turtle.onclick(*fun, btn=1, add=None*)** - Bind *fun* to mouse-click events on this turtle. If *fun* is None, existing bindings are removed. Example for the anonymous turtle.

- **turtle.getscreen()** - Return the TurtleScreen object the turtle is drawing on. TurtleScreen methods can then be called for that object.

- **turtle.bgcolor(**args*)** - Set or return background color of the TurtleScreen.

- **turtle.delay(*delay=None*)** - Set or return the drawing *delay* in milliseconds. The longer the drawing delay, the slower the animation.

- **turtle.update()** - Perform a TurtleScreen update. To be used when tracer is turned off.

- **turtle.onkeypress(*fun, key=None*)** - Bind *fun* to key-press event of key if key is given, or to any key-press-event if no key is given. Remark: in order to be able to register key-events, TurtleScreen must have focus.

- **turtle.mainloop()** - Starts event loop - calling Tkinter's mainloop function. Must be the last statement in a turtle graphics program. Must *not* be used if a script is run from within IDLE in -n mode (No subprocess) - for interactive use of turtle graphics.

IV. Steps and Code for the game

Step 1: We will be importing modules into the program and giving default values for the game.

```
import turtle
import time
import random

delay = 0.1
score = 0
high_score = 0
```

Step 2: Now, we will be creating the display of the game, i.e, the window screen for the game where we will create the head of the snake and food for the snake in the game and displaying the scores at the header of the game.

```
# Creating a window screen
wn = turtle.Screen()
wn.title("Snake Game")
wn.bgcolor("blue")

# the width and height can be put as user's choice
wn.setup(width=600, height=600)
wn.tracer(0)


# head of the snake
head = turtle.Turtle()
head.shape("square")
head.color("white")
head.penup()
head.goto(0, 0)
head.direction = "Stop"


# food in the game
food = turtle.Turtle()
colors = random.choice(['red', 'green', 'black'])
shapes = random.choice(['square', 'triangle', 'circle'])
food.speed(0)
food.shape(shapes)
food.color(colors)
food.penup()
food.goto(0, 100)


pen = turtle.Turtle()
```

```
pen.speed(0)
pen.shape("square")
pen.color("white")
pen.penup()
pen.hideturtle()
pen.goto(0, 250)
pen.write("Score : 0  High Score : 0", align="center",
          font=("candara", 24, "bold"))
```

Step 3: Now, we will be validating the key for the snake's movements. By clicking the keywords normally used for gaming 'w', 'a', 's' and 'd', we can operate the snake's movements around the screen.

```
# assigning key directions
def goup():
    if head.direction != "down":
        head.direction = "up"


def godown():
    if head.direction != "up":
        head.direction = "down"


def goleft():
    if head.direction != "right":
        head.direction = "left"


def goright():
    if head.direction != "left":
        head.direction = "right"


def move():
    if head.direction == "up":
        y = head.ycor()
        head.sety(y+20)
    if head.direction == "down":
        y = head.ycor()
        head.sety(y-20)
    if head.direction == "left":
        x = head.xcor()
        head.setx(x-20)
    if head.direction == "right":
        x = head.xcor()
        head.setx(x+20)
```

```
wn.listen()
wn.onkeypress(goup, "w")
wn.onkeypress(godown, "s")
wn.onkeypress(goleft, "a")
wn.onkeypress(goright, "d")
```

Step 4: Now, lastly, we will create the gameplay where the following will be happening:

- The snake will grow its body when the snake eats the fruits.
- Giving color to the snake's tail.
- After the fruit is eaten, the score will be counted.
- Checking for the snake's head collisions with the body or the wall of the window screen.
- Restarting the game automatically from the start after the collision.
- The new shape and color of the fruit will be introduced every time the window is restarted.
- The score will be returned to zero and a high score will be retained until the window is not closed.

```
segments = []

# Main Gameplay
while True:
    wn.update()
    if head.xcor() > 290 or head.xcor() < -290 or head.ycor() > 290 or head.ycor() < -290:
        time.sleep(1)
        head.goto(0, 0)
        head.direction = "Stop"
        colors = random.choice(['red', 'blue', 'green'])
        shapes = random.choice(['square', 'circle'])
        for segment in segments:
            segment.goto(1000, 1000)
        segments.clear()
        score = 0
        delay = 0.1
        pen.clear()
        pen.write("Score : {} High Score : {} ".format(
            score, high_score), align="center", font=("candara", 24, "bold"))
    if head.distance(food) < 20:
        x = random.randint(-270, 270)
        y = random.randint(-270, 270)
        food.goto(x, y)

        # Adding segment
```

```
        new_segment = turtle.Turtle()
        new_segment.speed(0)
        new_segment.shape("square")
        new_segment.color("orange")  # tail colour
        new_segment.penup()
        segments.append(new_segment)
        delay -= 0.001
        score += 10
        if score > high_score:
            high_score = score
        pen.clear()
        pen.write("Score : {} High Score : {} ".format(
            score, high_score), align="center", font=("candara", 24, "bold"))
    # Checking for head collisions with body segments
    for index in range(len(segments)-1, 0, -1):
        x = segments[index-1].xcor()
        y = segments[index-1].ycor()
        segments[index].goto(x, y)
    if len(segments) > 0:
        x = head.xcor()
        y = head.ycor()
        segments[0].goto(x, y)
    move()
    for segment in segments:
        if segment.distance(head) < 20:
            time.sleep(1)
            head.goto(0, 0)
            head.direction = "stop"
            colors = random.choice(['red', 'blue', 'green'])
            shapes = random.choice(['square', 'circle'])
            for segment in segments:
                segment.goto(1000, 1000)
            segment.clear()

            score = 0
            delay = 0.1
            pen.clear()
            pen.write("Score : {} High Score : {} ".format(
                score, high_score), align="center", font=("candara", 24, "bold"))
    time.sleep(delay)

wn.mainloop()
```