import turtle

import random

# Constants

w = 1000

h = 600

food\_size = 15

initial\_delay = 100

min\_delay = 60

level\_increment = 10

score = 0

level = 1

delay = initial\_delay # Define delay here

# Define wall boundaries

left\_boundary = -w / 2 + 20

right\_boundary = w / 2 - 20

top\_boundary = h / 2 - 20

bottom\_boundary = -h / 2 + 20

offsets = {

"up": (0, 20),

"down": (0, -20),

"left": (-20, 0),

"right": (20, 0)

}

paused = False # Variable to track whether the game is paused or not

# Create a turtle for displaying the score

score\_display = turtle.Turtle()

score\_display.speed(0) # Set the drawing speed

score\_display.color("white") # Set the text color

score\_display.penup()

score\_display.hideturtle()

score\_display.goto(0, h / 2 - 40)

score\_display.write("Score: 0", align="center", font=("Courier", 24, "normal"))

# Define the game over screen

game\_over\_screen = turtle.Turtle()

game\_over\_screen.speed(0)

game\_over\_screen.color("white")

game\_over\_screen.penup()

game\_over\_screen.hideturtle()

game\_over\_screen.goto(0, 0)

# Constants for high score

high\_score\_file = "high\_score.txt"

high\_score = 0

# Function to read the high score from the file

def load\_high\_score():

global high\_score

try:

with open(high\_score\_file, "r") as file:

high\_score = int(file.read())

except FileNotFoundError:

high\_score = 0

# Function to save the high score to the file

def save\_high\_score():

with open(high\_score\_file, "w") as file:

file.write(str(high\_score))

# Function to update the high score if needed

def update\_high\_score():

global high\_score

if score > high\_score:

high\_score = score

save\_high\_score()

# Function to display the high score on the screen

def display\_high\_score():

score\_display.clear()

score\_display.write(f"Score: {score} High Score: {high\_score}", align="center", font=("Courier", 24, "normal"))

def display\_game\_over():

game\_over\_screen.clear()

game\_over\_screen.write("Game Over", align="center", font=("Courier", 36, "normal"))

game\_over\_screen.goto(0, -40)

game\_over\_screen.write(f"Score: {score}", align="center", font=("Courier", 24, "normal"))

game\_over\_screen.goto(0, -80)

game\_over\_screen.write("Press 'R' to Restart", align="center", font=("Courier", 18, "normal"))

def game\_over():

global snake, snake\_dir, food\_position, pen, score, level, delay

snake\_dir = "stop" # Stop snake movement

update\_high\_score() # Check and update the high score

display\_game\_over()

def update\_score():

score\_display.clear()

score\_display.write(f"Score: {score}", align="center", font=("Courier", 24, "normal"))

def reset():

global snake, snake\_dir, food\_position, pen, score, level, delay, paused

delay = initial\_delay # Reset delay

snake = [[0, 0], [0, 20], [0, 40], [0, 60], [0, 80]]

snake\_dir = "up"

food\_position = get\_random\_food\_position()

food.goto(food\_position)

move\_snake()

score = 0

level = 1

load\_high\_score()

update\_score() # Update the score display

display\_high\_score() # Display high score

pen.clear()

game\_over\_screen.clear() # Clear the game over screen

screen.update()

screen.ontimer(display\_high\_score, 2000) # Display high score for 2 seconds

paused = False # Reset the paused state

def move\_snake():

global snake, snake\_dir, food\_position, pen, score, level, delay, paused

if paused:

turtle.ontimer(move\_snake, 100)

return

new\_head = snake[-1].copy()

new\_head[0] = snake[-1][0] + offsets[snake\_dir][0]

new\_head[1] = snake[-1][1] + offsets[snake\_dir][1]

# Check if the snake hits the left or right boundaries

if new\_head[0] < left\_boundary or new\_head[0] > right\_boundary:

game\_over() # Call the game over function

return

# Wrap around the top and bottom boundaries

if new\_head[1] > top\_boundary:

new\_head[1] = bottom\_boundary

elif new\_head[1] < bottom\_boundary:

new\_head[1] = top\_boundary

if new\_head in snake[:-1]:

game\_over() # Call the game over function

return

snake.append(new\_head)

if not food\_collision():

snake.pop(0)

pen.clearstamps()

for segment in snake:

pen.goto(segment[0], segment[1])

pen.stamp()

screen.update()

if score % level\_increment == 0:

level += 1

delay = max(delay - 10, min\_delay)

turtle.ontimer(move\_snake, delay)

def food\_collision():

global food\_position, score

if get\_distance(snake[-1], food\_position) < 20:

food\_position = get\_random\_food\_position()

food.goto(food\_position)

score += 1

update\_score() # Update the score display

return True

return False

def get\_random\_food\_position():

while True:

x = random.randint(left\_boundary, right\_boundary)

y = random.randint(bottom\_boundary, top\_boundary)

if (x, y) not in snake:

return (x, y)

def get\_distance(pos1, pos2):

x1, y1 = pos1

x2, y2 = pos2

distance = ((y2 - y1) \*\* 2 + (x2 - x1) \*\* 2) \*\* 0.5

return distance

def go\_up():

global snake\_dir

if snake\_dir != "down":

snake\_dir = "up"

def go\_right():

global snake\_dir

if snake\_dir != "left":

snake\_dir = "right"

def go\_down():

global snake\_dir

if snake\_dir != "up":

snake\_dir = "down"

def go\_left():

global snake\_dir

if snake\_dir != "right":

snake\_dir = "left"

def toggle\_pause():

global paused

paused = not paused

if paused:

score\_display.clear()

score\_display.write("Game Paused", align="center", font=("Courier", 36, "normal"))

else:

score\_display.clear()

update\_score()

display\_high\_score()

# Create the screen

screen = turtle.Screen()

screen.setup(w, h)

screen.title("Snake")

screen.bgcolor("red")

screen.tracer(0)

# Create the pen and food

pen = turtle.Turtle("square")

pen.penup()

food = turtle.Turtle()

food.shape("circle")

food.color("blue")

food.shapesize(food\_size / 20)

food.penup()

# Register the event handlers

screen.listen()

screen.onkey(go\_up, "Up")

screen.onkey(go\_right, "Right")

screen.onkey(go\_down, "Down")

screen.onkey(go\_left, "Left")

screen.onkey(reset, "r") # Add a key to restart the game

screen.onkey(toggle\_pause, "p") # Add a key to pause/resume the game

reset()

# Start the game

turtle.done()