from Tkinter import * # ALL VISUAL EQUIPMENT
import random # FOR RANDOM BEGINNINGS

WIDTH = 400 # OF SCREEN IN PIXELS
HEIGHT = 400 # OF SCREEN IN PIXELS
FLOOR = 0 # BOTTOM OF CONTAINER
BALLS = 2 # IN SIMULATION
WALL = 50 # FROM SIDE IN PIXELS
WALL_FORCE = 400 # ACCELERATION PER MOVE
SPEED_LIMIT = 3000 # FOR ball VELOCITY
BALL_RADIUS = 5 # FOR ballS IN PIXELS
OFFSET_START = 20 # FROM WALL IN PIXELS
FRAMES_PER_SEC = 40 # SCREEN UPDATE RATE

def main(): # Start the program.
    initialise() mainloop()

def initialise(): # Setup simulation variables.
    global active
    active = False
    build_balls()
    build_graph()

    def build_graph(): # Build GUI environment.
        global graph, left, top
        root = Tk()
        root.resizable(False, False)
        root.title('Balls')
        left = (root.winfo_screenwidth() - WIDTH) / 2
        top = (root.winfo_screenheight() - HEIGHT) / 2
        root.geometry('%dx%d+%d+%d' % (WIDTH, HEIGHT, left, top))
        root.bind_all('<Escape>', lambda event: event.widget.quit())
        root.bind('Configure', window_move)
        graph = Canvas(root, width=WIDTH, height=HEIGHT, background='white')
        graph.after(1000 / FRAMES_PER_SEC, update)
        graph.after(1000, activate)
        graph.pack()

        def lowerHeight(event):
            global HEIGHT
            HEIGHT -= 50
            print "lowered to ", HEIGHT

        def raiseHeight(event):
            global HEIGHT
            HEIGHT += 50
            print "raised to ", HEIGHT

        def activate():
            # Active window_move event.
            global active
            active = True

        def window_move(event):
            global left, top
            if active:
                diff = TwoD(left - event.x, top - event.y)

    def update(): # Main simulation loop.
        draw()
        move()

    def draw():
        graph.delete(ALL)
        # Draw sides.
        graph.create_rectangle((0, 0, WALL - BALL_RADIUS, HEIGHT), fill='light green')
        graph.create_rectangle((WIDTH - WALL + BALL_RADIUS, 0, WIDTH, HEIGHT), fill='light green')
        # Draw floor.
        y = HEIGHT - WALL + BALL_RADIUS + 2
        graph.create_line((WALL - BALL_RADIUS, y, WIDTH - WALL + BALL_RADIUS, y), fill='blue', width=3)
    # Draw all balls.

    def move():
        # Move all balls.
        for ball in balls:
            x1 = ball.position.x - BALL_RADIUS
            y1 = ball.position.y - BALL_RADIUS
            x2 = ball.position.x + BALL_RADIUS
            y2 = ball.position.y + BALL_RADIUS
            graph.create_oval((x1, y1, x2, y2), fill='red')
        graph.update()

    def simulate_wall(ball):
        # Create viewing boundaries.
        if ball.position.x < WALL:
            ball.velocity.x += WALL_FORCE
        elif ball.position.x > WIDTH - WALL:
            ball.velocity.x -= WALL_FORCE
        if ball.position.y >= HEIGHT - WALL:
            ball.velocity.y *= -1
            ball.position.y = HEIGHT - WALL

    def simulate_gravity(ball):
        # Create a pull.
        ball.velocity.y += 50

    def simulate_friction(ball):
        # Slow velocity down.
        ball.velocity = .9925

    def limit_speed(ball):
        # Limit ball speed.
        if ball.velocity.mag() > SPEED_LIMIT:
            ball.velocity = ball.velocity.mag() / SPEED_LIMIT

    for ball in balls:
        if HEIGHT - WALL - 2 < ball.position.y and top > event.y:
            ball.velocity.y = (-1000 * (top - event.y))
        ball.position += diff
        left, top = event.x, event.y

    for ball in balls:
        build_balls()
# TWO DIMENSIONAL VECTOR CLASS

class TwoD:
    def __init__(self, x, y):
        self.x = float(x)
        self.y = float(y)

    def __repr__(self):
        return 'TwoD(%s, %s)' % (self.x, self.y)

    def __add__(self, other):
        return TwoD(self.x + other.x, self.y + other.y)

    def __sub__(self, other):
        return TwoD(self.x - other.x, self.y - other.y)

    def __mul__(self, other):
        return TwoD(self.x * other, self.y * other)

    def __div__(self, other):
        return TwoD(self.x / other

    def __iadd__(self, other):
        self.x += other.x
        self.y += other.y
        return self

    def __isub__(self, other):
        self.x -= other.x
        self.y -= other.y
        return self

    def __imul__(self, other):
        self.x *= other
        self.y *= other
        return self

    def __idiv__(self, other):
        self.x /= other
        self.y /= other
        return self

    def mag(self):
        return ((self.x ** 2) + (self.y ** 2)) ** 0.5

# BALL IMPLEMENTATION CLASS

class Ball:
    def __init__(self, width, height, offset, move_divider):
        self.velocity = TwoD(0, 0)
        self.position = TwoD(*(-offset if random.randint(0, 1) else width + offset, random.randint(1, height)))
        self.move_divider = move_divider * 5

    def update_velocity(self, balls):
        vector = TwoD(0, 0)
        for ball in balls:
            if ball is not self:
                if (self.position - ball.position).mag() < (BALL_RADIUS * 2.5):
                    vector = (ball.position - self.position) / vector.mag() / vector.mag()

        self.velocity += vector * self.velocity
        self.position += self.move / self.move_divider

    def move(self):
        self.velocity += self.__temp
        limit_speed(self)
        self.position += self.velocity / self.move_divider

# Execute the simulation.
if __name__ == '__main__':
    main()