**AI Game Documentation**

**Aim:** To create a Bouncing Ball Game.

**Algorithm Used:**

Physics simulation calculates the ball’s position and movement based on velocity and acceleration.

Collision detection identifies when and where the ball collides with the screen boundaries (or other objects) and adjusts the ball's velocity to simulate bouncing behavior.

**Program:**

from tkinter import \*

import time

import random

root = Tk()

root.title("Bounce Ball")

root.geometry("500x570")

root.resizable(0, 0)

root.wm\_attributes("-topmost", 1)

canvas = Canvas(root, width=500, height=500, bd=0, highlightthickness=0, highlightbackground="Red", bg="Black")

canvas.pack(padx=10, pady=10)

score = Label(height=50, width=80, text="Score: 00", font="Consolas 14 bold")

score.pack(side="left")

root.update()

class Ball:

    def \_\_init\_\_(self, canvas, color, paddle, bricks, score):

        self.bricks = bricks

        self.canvas = canvas

        self.paddle = paddle

        self.score = score

        self.bottom\_hit = False

        self.hit = 0

        self.id = canvas.create\_oval(10, 10, 25, 25, fill=color, width=1)

        self.canvas.move(self.id, 230, 461)

        start = [4, 3.8, 3.6, 3.4, 3.2, 3, 2.8, 2.6]

        random.shuffle(start)

        self.x = start[0]

        self.y = -start[0]

        self.canvas.move(self.id, self.x, self.y)

        self.canvas\_height = canvas.winfo\_height()

        self.canvas\_width = canvas.winfo\_width()

    def brick\_hit(self, pos):

        for brick\_line in self.bricks:

            for brick in brick\_line:

                brick\_pos = self.canvas.coords(brick.id)

                try:

                    if pos[2] >= brick\_pos[0] and pos[0] <= brick\_pos[2]:

                        if pos[3] >= brick\_pos[1] and pos[1] <= brick\_pos[3]:

                            canvas.bell()

                            self.hit += 1

                            self.score.configure(text="Score: " + str(self.hit))

                            self.canvas.delete(brick.id)

                            return True

                except:

                    continue

        return False

    def paddle\_hit(self, pos):

        paddle\_pos = self.canvas.coords(self.paddle.id)

        if pos[2] >= paddle\_pos[0] and pos[0] <= paddle\_pos[2]:

            if pos[3] >= paddle\_pos[1] and pos[1] <= paddle\_pos[3]:

                return True

        return False

    def draw(self):

        self.canvas.move(self.id, self.x, self.y)

        pos = self.canvas.coords(self.id)

        start = [4, 3.8, 3.6, 3.4, 3.2, 3, 2.8, 2.6]

        random.shuffle(start)

        if self.brick\_hit(pos):

            self.y = start[0]

        if pos[1] <= 0:

            self.y = start[0]

        if pos[3] >= self.canvas\_height:

            self.bottom\_hit = True

        if pos[0] <= 0:

            self.x = start[0]

        if pos[2] >= self.canvas\_width:

            self.x = -start[0]

        if self.paddle\_hit(pos):

            self.y = -start[0]

class Paddle:

    def \_\_init\_\_(self, canvas, color):

        self.canvas = canvas

        self.id = canvas.create\_rectangle(0, 0, 100, 10, fill=color)

        self.canvas.move(self.id, 200, 485)

        self.x = 0

        self.pausec = 0

        self.canvas\_width = canvas.winfo\_width()

        self.canvas.bind\_all("<Left>", self.turn\_left)

        self.canvas.bind\_all("<Right>", self.turn\_right)

        self.canvas.bind\_all("<space>", self.pauser)

    def draw(self):

        pos = self.canvas.coords(self.id)

        if pos[0] + self.x <= 0:

            self.x = 0

        if pos[2] + self.x >= self.canvas\_width:

            self.x = 0

        self.canvas.move(self.id, self.x, 0)

    def turn\_left(self, event):

        self.x = -3.5

    def turn\_right(self, event):

        self.x = 3.5

    def pauser(self, event):

        self.pausec += 1

        if self.pausec == 2:

            self.pausec = 0

class Bricks:

    def \_\_init\_\_(self, canvas, color):

        self.canvas = canvas

        self.id = canvas.create\_oval(5, 5, 25, 25, fill=color, width=2)

playing = False

def start\_game(event):

    global playing

    if not playing:

        playing = True

        score.configure(text="Score: 00")

        canvas.delete("all")

        BALL\_COLOR = ["red", "yellow", "white", "green"]

        BRICK\_COLOR = ["PeachPuff3", "dark slate gray", "rosy brown", "light goldenrod yellow", "turquoise3", "salmon",

                       "light steel blue", "dark khaki", "pale violet red", "orchid", "tan", "MistyRose2",

                       "DodgerBlue4", "wheat2", "RosyBrown2", "bisque3", "DarkSeaGreen1"]

        random.shuffle(BALL\_COLOR)

        paddle = Paddle(canvas, "blue")

        bricks = []

        for i in range(0, 5):

            b = []

            for j in range(0, 19):

                random.shuffle(BRICK\_COLOR)

                tmp = Bricks(canvas, BRICK\_COLOR[0])

                b.append(tmp)

            bricks.append(b)

        for i in range(0, 5):

            for j in range(0, 19):

                canvas.move(bricks[i][j].id, 25 \* j, 25 \* i)

        ball = Ball(canvas, BALL\_COLOR[0], paddle, bricks, score)

        root.update\_idletasks()

        root.update()

        time.sleep(1)

        while True:

            if paddle.pausec != 1:

                try:

                    canvas.delete(m)

                except NameError:

                    pass

                if not ball.bottom\_hit:

                    ball.draw()

                    paddle.draw()

                    root.update\_idletasks()

                    root.update()

                    time.sleep(0.01)

                    if ball.hit == 95:

                        canvas.create\_text(250, 250, text="YOU WON !!", fill="yellow", font="Consolas 24 ")

                        root.update\_idletasks()

                        root.update()

                        playing = False

                        break

                else:

                    canvas.create\_text(250, 250, text="GAME OVER!!", fill="red", font="Consolas 24 ")

                    root.update\_idletasks()

                    root.update()

                    playing = False

                    break

            else:

                try:

                    m = canvas.create\_text(250, 250, text="PAUSE!!", fill="green", font="Consolas 24 ")

                except NameError:

                    pass

                root.update\_idletasks()

                root.update()

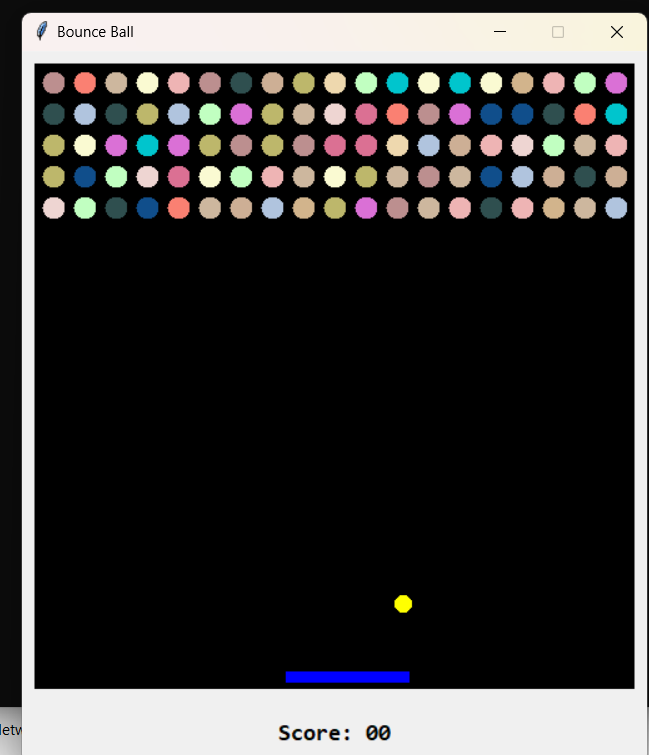
root.bind\_all("<Return>", start\_game)

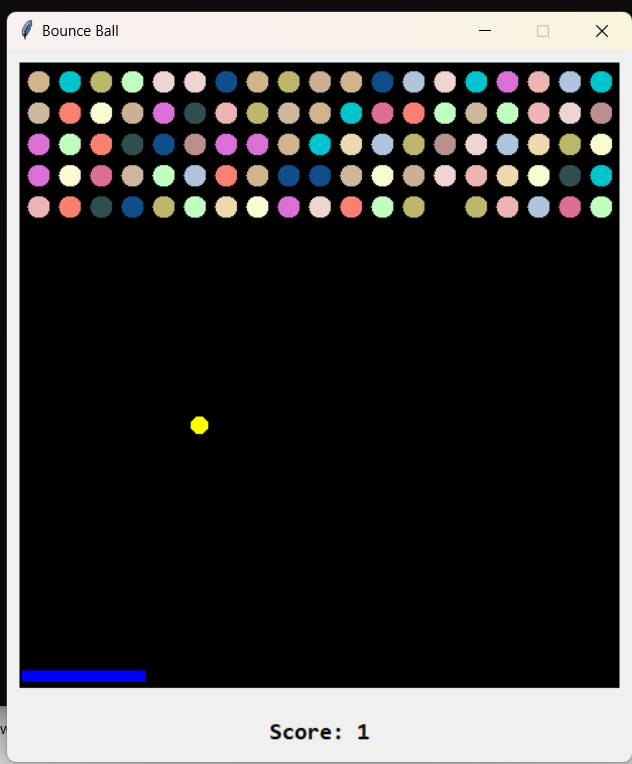
canvas.create\_text(250, 250, text="Press Enter to start Game!!", fill="red", font="Consolas 18")

root.mainloop()

**Output:**

****

****

****

****

**Conclusion**: The program to create a Bouncing Ball Game has been created and executed successfully.