Main.py

```
# LUDOMANIA #
from tkinter import *
from tkinter import messagebox
from PIL import Image,ImageTk
import random
import time
from random import randint, choice


class Ludomania:
    def show_hint(self):
        possible_moves = []
    # Calculate possible moves for the current player
        for coin in self.robo_store:
        # Check if the coin can move forward
         if self.temp[coin-1] + self.move_red_counter <= 106:
            possible_moves.append((coin, self.temp[coin-1] + self.move_red_counter))
        # Check if the coin can capture an opponent's coin
        for opponent_coin in self.blue_coin_position:
            if self.temp[coin-1] + self.move_red_counter == opponent_coin:
                possible_moves.append((coin, opponent_coin))


    # Display the two possible moves to the user
            if len(possible_moves) >= 2:
                move1 = possible_moves[0]
                move2 = possible_moves[1]
            messagebox.showinfo("Hint", f"Possible moves: {move1} or {move2}")
        else:
            messagebox.showinfo("Hint", "No possible moves found")
```

```
    def __init__(self,
root,six_side_block,five_side_block,four_side_block,three_side_block,two_side_block,one_side_blo
ck):
        self.window = root
        # Make canvas
        self.make_canvas = Canvas(self.window, bg="#FFFFFF", width=1000, height=630)
        self.make_canvas.pack(fill=BOTH,expand=1)




        # Make some containers to store data
        self.made_red_coin = []
        self.made_grn_coin = []
        self.made_pink_coin = []
        self.made_blue_coin = []

        self.red_number_label = []
        self.grn_number_label = []
        self.pink_number_label = []
        self.blue_number_label = []

        self.block_value_predict = []
        self.total_people_play = []

        # Ludo block all side image store
        self.block_number_side = [one_side_block, two_side_block, three_side_block, four_side_block,
five_side_block, six_side_block]

        # Use for store specific position of all coins
```

```
self.red_coord_store = [-1, -1, -1, -1]
self.grn_coord_store = [-1, -1, -1, -1]
self.pink_coord_store = [-1, -1, -1, -1]
self.blue_coord_store = [-1, -1, -1, -1]

self.red_coin_position = [0, 1, 2, 3]
self.grn_coin_position = [0, 1, 2, 3]
self.pink_coin_position = [0, 1, 2, 3]
self.blue_coin_position = [0, 1, 2, 3]

for index in range(len(self.red_coin_position)):# Specific coin position set to -1 by default
    self.red_coin_position[index] = -1
    self.grn_coin_position[index] = -1
    self.pink_coin_position[index] = -1
    self.blue_coin_position[index] = -1

# Number to room to be traverse by specific color coin, store in that variable
self.move_red_counter = 0
self.move_grn_counter = 0
self.move_pink_counter = 0
self.move_blue_counter = 0

self.take_permission = 0
self.six_with_overlap = 0

self.red_store_active = 0
self.blue_store_active = 0
self.pink_store_active = 0
self.grn_store_active = 0

self.six_counter = 0
```

```
        self.time_for = -1

        # Some variables initializes with None
        self.right_star = None
        self.down_star = None
        self.left_star = None
        self.up_star = None

        # Robo Control
        self.robo_prem = 0
        self.count_robo_stage_from_start = 0
        self.robo_store = []

        # By default some function call
        self.board_set_up()

        self.instruction_btn_red()
        self.instruction_btn_blue()
        self.instruction_btn_pink()
        self.instruction_btn_grn()

        self.take_initial_control()



    def board_set_up(self):
        # Cover Box made
        self.make_canvas.create_rectangle(100, 15, 100 + (40 * 15), 15 + (40 * 15), width=6, fill="white")

        # Square box
        self.make_canvas.create_rectangle(100, 15, 100+240, 15+240, width=3, fill="red")# left up large
square
```

```
        self.make_canvas.create_rectangle(100, (15+240)+(40*3), 100+240, (15+240)+(40*3)+(40*6), width=3, fill="#073763")# left down large square

        self.make_canvas.create_rectangle(340+(40*3), 15, 340+(40*3)+(40*6), 15+240, width=3, fill="#8fce73")# right up large square

        self.make_canvas.create_rectangle(340+(40*3), (15+240)+(40*3), 340+(40*3)+(40*6), (15+240)+(40*3)+(40*6), width=3, fill="pink")# right down large square


        # Left 3 box(In white region)

        self.make_canvas.create_rectangle(100, (15+240), 100+240, (15+240)+40, width=3)

        self.make_canvas.create_rectangle(100+40, (15 + 240)+40, 100 + 240, (15 + 240) + 40+40, width=3, fill="#F00000")

        self.make_canvas.create_rectangle(100, (15 + 240)+80, 100 + 240, (15 + 240) + 80+40, width=3)


        # right 3 box(In white region)

        self.make_canvas.create_rectangle(100+240, 15, 100 + 240+40, 15 + (40*6), width=3)

        self.make_canvas.create_rectangle(100+240+40, 15+40, 100+240+80, 15 + (40*6), width=3, fill="#8fce73")

        self.make_canvas.create_rectangle(100+240+80, 15, 100 + 240+80+40, 15 + (40*6), width=3)


        # up 3 box(In white region)

        self.make_canvas.create_rectangle(340+(40*3), 15+240, 340+(40*3)+(40*6), 15+240+40, width=3)

        self.make_canvas.create_rectangle(340+(40*3), 15+240+40, 340+(40*3)+(40*6)-40, 15+240+80, width=3, fill="pink")

        self.make_canvas.create_rectangle(340+(40*3), 15+240+80, 340+(40*3)+(40*6), 15+240+120, width=3)


        # down 3 box(In white region)

        self.make_canvas.create_rectangle(100, (15 + 240)+(40*3), 100 + 240+40, (15 + 240)+(40*3)+(40*6), width=3)

        self.make_canvas.create_rectangle(100+240+40, (15 + 240)+(40*3), 100 + 240+40+40, (15 + 240)+(40*3)+(40*6)-40, width=3, fill="#073763")

        self.make_canvas.create_rectangle(100 + 240+40+40, (15 + 240)+(40*3), 100 + 240+40+40+40, (15 + 240)+(40*3)+(40*6), width=3)
```

```
# All left separation line
start_x = 100 + 40
start_y = 15 + 240
end_x = 100 + 40
end_y = 15 + 240 + (40 * 3)
for _ in range(5):
    self.make_canvas.create_line(start_x, start_y, end_x, end_y, width=3)
    start_x+=40
    end_x+= 40

# All right separation line
start_x = 100+240+(40*3)+40
start_y = 15 + 240
end_x = 100+240+(40*3)+40
end_y = 15 + 240 + (40 * 3)
for _ in range(5):
    self.make_canvas.create_line(start_x, start_y, end_x, end_y, width=3)
    start_x += 40
    end_x += 40

# All up separation done
start_x = 100+240
start_y = 15+40
end_x = 100+240+(40*3)
end_y = 15+40
for _ in range(5):
    self.make_canvas.create_line(start_x, start_y, end_x, end_y, width=3)
    start_y += 40
    end_y += 40

# All down separation done
```

```
    start_x = 100 + 240
    start_y = 15 + (40*6)+(40*3)+40
    end_x = 100 + 240 + (40 * 3)
    end_y = 15 + (40*6)+(40*3)+40
    for _ in range(5):
        self.make_canvas.create_line(start_x, start_y, end_x, end_y, width=3)
        start_y += 40
        end_y += 40

    # Square box(Coins containers) white region make
    self.make_canvas.create_rectangle(100+20, 15+40-20, 100 + 40 + 60 + 40 +60+20, 15+40+40+40+100-20, width=3, fill="white")
    self.make_canvas.create_rectangle(340+(40*3)+40 - 20, 15 + 40-20, 340+(40*3)+40 + 60 + 40 + 40+20+20, 15+40+40+40+100-20, width=3, fill="white")
    self.make_canvas.create_rectangle(100+20, 340+80-20+15, 100 + 40 + 60 + 40 +60+20, 340+80+60+40+40+20+15, width=3, fill="white")
    self.make_canvas.create_rectangle(340+(40*3)+40 - 20, 340 + 80 - 20+15, 340+(40*3)+40 + 60 + 40 + 40+20+20, 340 + 80 + 60 + 40 + 40 + 20+15, width=3, fill="white")

    # Left up square inside box made
    self.make_canvas.create_rectangle(100+40, 15+40, 100+40+40, 15+40+40, width=3, fill="red")
    self.make_canvas.create_rectangle(100+40+60+60, 15 + 40, 100+40+60+40+60, 15 + 40 + 40, width=3, fill="red")
    self.make_canvas.create_rectangle(100 + 40, 15 + 40+100, 100 + 40 + 40, 15 + 40 + 40+100, width=3, fill="red")
    self.make_canvas.create_rectangle(100 + 40 + 60 + 60, 15 + 40+100, 100 + 40 + 60 + 40 +60, 15 + 40 + 40+100, width=3, fill="red")

    # Right up square inside box made
    self.make_canvas.create_rectangle(340+(40*3)+40, 15 + 40, 340+(40*3)+40 + 40, 15 + 40 + 40, width=3, fill="#8fce73")
```

```
        self.make_canvas.create_rectangle(340+(40*3)+40+ 60 + 40+20, 15 + 40, 340+(40*3)+40 + 60 + 40 + 40+20, 15 + 40 + 40, width=3, fill="#8fce73") #green

        self.make_canvas.create_rectangle(340+(40*3)+40, 15 + 40 + 100, 340+(40*3)+40 + 40, 15 + 40 + 40 + 100, width=3, fill="#8fce73")

        self.make_canvas.create_rectangle(340+(40*3)+40+ 60 + 40+20, 15 + 40 + 100, 340+(40*3)+40 + 60 + 40 + 40+20, 15 + 40 + 40 + 100, width=3, fill="#8fce73")



        # Left down square inside box made

        self.make_canvas.create_rectangle(100 + 40, 340+80+15, 100 + 40 + 40, 340+80+40+15, width=3, fill="#073763")  #blue

        self.make_canvas.create_rectangle(100 + 40 + 60 + 40+20, 340+80+15, 100 + 40 + 60 + 40 + 40+20, 340+80+40+15, width=3, fill="#073763")

        self.make_canvas.create_rectangle(100 + 40, 340+80+60+40+15, 100 + 40 + 40, 340+80+60+40+40+15, width=3, fill="#073763")

        self.make_canvas.create_rectangle(100 + 40 + 60 + 40+20, 340+80+60+40+15, 100 + 40 + 60 + 40 + 40+20, 340+80+60+40+40+15, width=3, fill="#073763")



        # Right down square inside box made

        self.make_canvas.create_rectangle(340 + (40 * 3) + 40, 340+80+15, 340 + (40 * 3) + 40 + 40, 340+80+40+15, width=3, fill="pink")

        self.make_canvas.create_rectangle(340 + (40 * 3) + 40 + 60 + 40+20, 340+80+15, 340 + (40 * 3) + 40 + 60 + 40 + 40+20, 340+80+40+15, width=3, fill="pink")

        self.make_canvas.create_rectangle(340 + (40 * 3) + 40, 340+80+60+40+15, 340 + (40 * 3) + 40 + 40,340+80+60+40+40+15, width=3, fill="pink")

        self.make_canvas.create_rectangle(340 + (40 * 3) + 40 + 60 + 40+20, 340+80+60+40+15,340 + (40 * 3) + 40 + 60 + 40 + 40+20, 340+80+60+40+40+15, width=3, fill="pink")



        # blue start position

        self.make_canvas.create_rectangle(100+240,340+(40*5)-5,100+240+40,340+(40*6)-5,fill="#073763",width=3) #green

        # Red start position

        self.make_canvas.create_rectangle(100 + 40, 15+(40*6), 100 +40 + 40, 15+(40*6)+40, fill="red", width=3)

        # Green start position
```

```
self.make_canvas.create_rectangle(100 + (40*8), 15 + 40, 100 +(40*9), 15 + 40+ 40, fill="#8fce73", width=3) #blue

# pink start position

self.make_canvas.create_rectangle(100 + (40 * 6)+(40*3)+(40*4), 15 + (40*8), 100 + (40 * 6)+(40*3)+(40*5), 15 + (40*9), fill="pink", width=3)


# Traingle in middle depending upon the vertex(1,2,3)

self.make_canvas.create_polygon(100+240, 15+240, 100+240+60, 15+240+60, 100+240, 15+240+(40*3), width=3,fill="red",outline="black")

self.make_canvas.create_polygon(100 + 240+(40*3), 15 + 240, 100 + 240 + 60, 15 + 240 + 60, 100 + 240+(40*3), 15 + 240 + (40 * 3), width=3, fill="pink",outline="black")

self.make_canvas.create_polygon(100 + 240, 15 + 240, 100 + 240 + 60, 15 + 240 + 60, 100 + 240 + (40 * 3), 15 + 240, width=3, fill="#8fce73",outline="black")

self.make_canvas.create_polygon(100 + 240, 15 + 240+(40*3), 100 + 240 + 60, 15 + 240 + 60, 100 + 240 + (40 * 3), 15 + 240+(40*3), width=3, fill="#073763",outline="black") #green


# Make coin for red left up block(top left , bottom right corner)

red_1_coin = self.make_canvas.create_oval(100+40, 15+40, 100+40+40, 15+40+40, width=3, fill="red", outline="black")

red_2_coin = self.make_canvas.create_oval(100+40+60+60, 15 + 40, 100+40+60+60+40, 15 + 40 + 40, width=3, fill="red", outline="black")

red_3_coin = self.make_canvas.create_oval(100 + 40 + 60 + 60, 15 + 40 + 100, 100 + 40 + 60 + 60 + 40, 15 + 40 + 40 + 100, width=3, fill="red", outline="black")

red_4_coin = self.make_canvas.create_oval(100 + 40, 15 + 40+100, 100 + 40 + 40, 15 + 40 + 40+100, width=3,fill="red", outline="black")

self.made_red_coin.append(red_1_coin)

self.made_red_coin.append(red_2_coin)

self.made_red_coin.append(red_3_coin)

self.made_red_coin.append(red_4_coin)


# Make coin under number label for red left up block

red_1_label = Label(self.make_canvas, text="1", font=("Arial", 15, "bold"), bg="red", fg="black")

red_1_label.place(x=100 + 40 + 10, y=15 + 40 + 5)

red_2_label = Label(self.make_canvas, text="2", font=("Arial", 15, "bold"), bg="red", fg="black")
```

```
        red_2_label.place(x=100 + 40 + 60 + 60 + 10, y=15 + 40 + 5)

        red_3_label = Label(self.make_canvas, text="3", font=("Arial", 15, "bold"), bg="red", fg="black")

        red_3_label.place(x=100 + 40 + 60 + 60 + 10, y=15 + 40 + 100 + 5)

        red_4_label = Label(self.make_canvas, text="4", font=("Arial", 15, "bold"), bg="red", fg="black")

        red_4_label.place(x=100 + 40 + 10, y=15 + 40 + 100 + 5)

        self.red_number_label.append(red_1_label)

        self.red_number_label.append(red_2_label)

        self.red_number_label.append(red_3_label)

        self.red_number_label.append(red_4_label)


        # Make coin for green right up block

        grn_1_coin = self.make_canvas.create_oval(340+(40*3)+40, 15 + 40, 340+(40*3)+40 + 40, 15 +
40 + 40, width=3, fill="#8fce73", outline="black")

        grn_2_coin = self.make_canvas.create_oval(340+(40*3)+40+ 60 + 40+20, 15 + 40,
340+(40*3)+40 + 60 + 40 + 40+20, 15 + 40 + 40, width=3, fill="#8fce73", outline="black")

        grn_3_coin = self.make_canvas.create_oval(340 + (40 * 3) + 40 + 60 + 40 + 20, 15 + 40 + 100, 340
+ (40 * 3) + 40 + 60 + 40 + 40 + 20, 15 + 40 + 40 + 100, width=3, fill="#8fce73", outline="black")

        grn_4_coin = self.make_canvas.create_oval(340+(40*3)+40, 15 + 40 + 100, 340+(40*3)+40 + 40,
15 + 40 + 40 + 100, width=3, fill="#8fce73", outline="black")

        self.made_grn_coin.append(grn_1_coin)

        self.made_grn_coin.append(grn_2_coin)

        self.made_grn_coin.append(grn_3_coin)

        self.made_grn_coin.append(grn_4_coin)


        # Make coin under number label for green right up block

        grn_1_label = Label(self.make_canvas, text="1", font=("Arial", 15, "bold"), bg="#8fce73",
fg="black")

        grn_1_label.place(x=340 + (40 * 3) + 40 + 10, y=15 + 40 + 5)

        grn_2_label = Label(self.make_canvas, text="2", font=("Arial", 15, "bold"), bg="#8fce73",
fg="black")

        grn_2_label.place(x=340 + (40 * 3) + 40 + 40 + 60 + 30, y=15 + 40 + 5)

        grn_3_label = Label(self.make_canvas, text="3", font=("Arial", 15, "bold"), bg="#8fce73",
fg="black")
```

```
        grn_3_label.place(x=340 + (40 * 3) + 40 + 40 + 60 + 30, y=15 + 40 + 100 + 5)

        grn_4_label = Label(self.make_canvas, text="4", font=("Arial", 15, "bold"), bg="#8fce73",
fg="black")

        grn_4_label.place(x=340 + (40 * 3) + 40 + 10, y=15 + 40 + 100 + 5)

        self.grn_number_label.append(grn_1_label)

        self.grn_number_label.append(grn_2_label)

        self.grn_number_label.append(grn_3_label)

        self.grn_number_label.append(grn_4_label)


        # Make coin for sky_blue left down block

        blue_1_label_1_coin = self.make_canvas.create_oval(100 + 40, 340+80+15, 100 + 40 + 40,
340+80+40+15, width=3, fill="#073763", outline="black")

        blue_2_coin = self.make_canvas.create_oval(100 + 40 + 60 + 40+20, 340+80+15, 100 + 40 + 60 +
40 + 40+20, 340+80+40+15, width=3, fill="#073763", outline="black")

        blue_3_coin = self.make_canvas.create_oval(100 + 40 + 60 + 40 + 20, 340 + 80 + 60 + 40 + 15,
100 + 40 + 60 + 40 + 40 + 20, 340 + 80 + 60 + 40 + 40 + 15, width=3, fill="#073763", outline="black")

        blue_4_coin = self.make_canvas.create_oval( 100 + 40, 340+80+60+40+15, 100 + 40 + 40,
340+80+60+40+40+15, width=3, fill="#073763", outline="black")

        self.made_blue_coin.append(blue_2_coin)

        self.made_blue_coin.append(blue_2_coin)

        self.made_blue_coin.append(blue_3_coin)

        self.made_blue_coin.append(blue_4_coin)


        # Make coin under number label for blue left down block

        blue_1_label = Label(self.make_canvas, text="1", font=("Arial", 15, "bold"), bg="#073763",
fg="black")

        blue_1_label.place(x=100 + 40 + 10, y=30 + (40 * 6) + (40 * 3) + 40 + 10)

        blue_2_label = Label(self.make_canvas, text="2", font=("Arial", 15, "bold"), bg="#073763",
fg="black")

        blue_2_label.place(x=100 + 40 + 60 + 60 + 10, y=30 + (40 * 6) + (40 * 3) + 40 + 10)

        blue_3_label = Label(self.make_canvas, text="3", font=("Arial", 15, "bold"), bg="#073763",
fg="black")

        blue_3_label.place(x=100 + 40 + 60 + 60 + 10, y=30 + (40 * 6) + (40 * 3) + 40 + 60 + 40 + 10)
```

```
        blue_4_label = Label(self.make_canvas, text="4", font=("Arial", 15, "bold"), bg="#073763",
fg="black")

        blue_4_label.place(x=100 + 40 + 10, y=30 + (40 * 6) + (40 * 3) + 40 + 60 + 40 + 10)

        self.blue_number_label.append(blue_1_label)

        self.blue_number_label.append(blue_2_label)

        self.blue_number_label.append(blue_3_label)

        self.blue_number_label.append(blue_4_label)


        # Make coin for pink right down block

        pink_1_coin = self.make_canvas.create_oval(340 + (40 * 3) + 40, 340+80+15, 340 + (40 * 3) + 40
+ 40, 340+80+40+15, width=3, fill="pink", outline="black")

        pink_2_coin = self.make_canvas.create_oval(340 + (40 * 3) + 40 + 60 + 40 + 20, 340+80+15, 340
+ (40 * 3) + 40 + 60 + 40 + 40+20, 340+80+40+15, width=3, fill="pink", outline="black")

        pink_3_coin = self.make_canvas.create_oval(340 + (40 * 3) + 40 + 60 + 40 + 20, 340 + 80 + 60 +
40 + 15, 340 + (40 * 3) + 40 + 60 + 40 + 40 + 20, 340 + 80 + 60 + 40 + 40 + 15, width=3, fill="pink",
outline="black")

        pink_4_coin = self.make_canvas.create_oval(340 + (40 * 3) + 40, 340+80+60+40+15, 340 + (40 *
3) + 40 + 40,340+80+60+40+40+15, width=3, fill="pink", outline="black")

        self.made_pink_coin.append(pink_1_coin)

        self.made_pink_coin.append(pink_2_coin)

        self.made_pink_coin.append(pink_3_coin)

        self.made_pink_coin.append(pink_4_coin)


        # Make coin under number label for pink right down block

        pink_1_label = Label(self.make_canvas, text="1", font=("Arial", 15, "bold"), bg="pink",
fg="black")

        pink_1_label.place(x=340 + (40 * 3) + 40 + 10, y=30 + (40 * 6) + (40 * 3) + 40 + 10)

        pink_2_label = Label(self.make_canvas, text="2", font=("Arial", 15, "bold"), bg="pink",
fg="black")

        pink_2_label.place(x=340 + (40 * 3) + 40 + 40 + 60 + 30, y=30 + (40 * 6) + (40 * 3) + 40 + 10)

        pink_3_label = Label(self.make_canvas, text="3", font=("Arial", 15, "bold"), bg="pink",
fg="black")

        pink_3_label.place(x=340 + (40 * 3) + 40 + 40 + 60 + 30, y=30 + (40 * 6) + (40 * 3) + 40 + 100 +
10)
```

```
        pink_4_label = Label(self.make_canvas, text="4", font=("Arial", 15, "bold"), bg="pink",
fg="black")

        pink_4_label.place(x=340 + (40 * 3) + 40 + 10, y=30 + (40 * 6) + (40 * 3) + 40 + 100 + 10)

        self.pink_number_label.append(pink_1_label)

        self.pink_number_label.append(pink_2_label)

        self.pink_number_label.append(pink_3_label)

        self.pink_number_label.append(pink_4_label)




        #Making a STAR for the SAFEZONE(Whole Star = Right, Up ,Left, Down)


        # Right star

        common_x = 340+(40*6)+20

        common_y = 15+240+2

        #          A              B              C              D              E                   F
G                   H              I              J              K              L

        coord = [common_x,common_y,  common_x+5,common_y+15,  common_x+15,common_y+15,
common_x+8,common_y+20,   common_x+15,common_y+25,   common_x+5,common_y+25,
common_x,common_y+25+10,   common_x-5,common_y+25,   common_x-16,common_y+25,
common_x-8,common_y+15+5,   common_x-15,common_y+15,   common_x-5,common_y+15]

        self.make_canvas.create_polygon(coord,width=3,fill="blue")


        # Up star

        common_x = 100+240+2+18

        common_y = 15 + (40*2) + 2

        #          A                   B                   C                   D                   E
F              G                   H                   I              J                   K
L

        coord = [common_x, common_y,   common_x + 5,   common_y + 15,     common_x + 15,
common_y + 15,     common_x + 8, common_y + 20,     common_x + 15, common_y + 25,
common_x + 5, common_y + 25,     common_x, common_y + 25 + 10,    common_x - 5, common_y +
25,    common_x - 16, common_y + 25,    common_x - 8,common_y + 15 + 5,    common_x -
15,common_y + 15,    common_x - 5,common_y + 15]

        self.make_canvas.create_polygon(coord, width=3, fill="blue")
```

```
        # Left star
        common_x = 100 + (40*2) + 2 +18
        common_y = 15 + 240+(40*2) + 2
        #               A               B                       C                   D                       E
F                   G                       H                       I                   J                       K
L
        coord = [common_x, common_y,   common_x + 5, common_y + 15,    common_x + 15,
common_y + 15,    common_x + 8,common_y + 20,    common_x + 15, common_y + 25,    common_x
+ 5, common_y + 25,    common_x, common_y + 25 + 10,    common_x - 5, common_y + 25,
common_x - 16, common_y + 25,     common_x - 8, common_y + 15 + 5,     common_x - 15,
common_y + 15,     common_x - 5, common_y + 15]
        self.make_canvas.create_polygon(coord, width=3, fill="blue")


        # Down star
        common_x = 100 + 240 + (40*2) + 2 + 18
        common_y = 15 + (40 * 6) + (40*3)+(40*3)+2
        #           A                   B                       C                           D                   E
F                       G                       H                       I                       J
K                       L
        coord = [common_x, common_y,   common_x + 5, common_y + 15,    common_x + 15,
common_y + 15,    common_x + 8, common_y + 20,    common_x + 15, common_y + 25,
common_x + 5, common_y + 25,       common_x, common_y + 25 + 10,       common_x - 5,
common_y + 25,       common_x - 16, common_y + 25,       common_x - 8, common_y + 15 + 5,
common_x - 15, common_y + 15,      common_x - 5, common_y + 15]
        self.make_canvas.create_polygon(coord, width=3, fill="blue")


    # Total number of players: Control take at first
    def take_initial_control(self):
        for i in range(4):
            self.block_value_predict[i][1]['state'] = DISABLED


        # Make other window to control take
        top = Toplevel()
        top.geometry("530x300")
```

```
        top.maxsize(530,300)
        top.minsize(530,300)
        top.config(bg="#92c1ec")
        top.iconbitmap("Images/ludo_icon.ico")

        head = Label(top,text="Enter the number of
players",font=("Arial",25,"bold"),bg="#92c1ec",fg="Black")
        head.place(x=50,y=30)
        take_entry = Entry(top,font=("Arial",18,"bold","italic"),relief=SUNKEN,bd=5,width=12,
state=DISABLED)
        take_entry.place(x=130,y=85)
        take_entry.focus()

        def filtering():# Total player input value filtering
            def input_filtering(coin_number):# Input value Filtering
                try:
                    return True if (4>=int(coin_number)>=2) or type(coin_number) == int else False
                except:
                    return False

            response_take = input_filtering(take_entry.get())
            if response_take:
                for player_index in range(int(take_entry.get())):
                    self.total_people_play.append(player_index)
                print(self.total_people_play)
                self.make_command()
                top.destroy()
            else:
                messagebox.showerror("Input Error", "input number between 2 and 4")
                top.destroy()
                self.take_initial_control()
```

```
    submit_btn =
Button(top,text="Submit",bg="#262626",fg="#FFFFFF",font=("Arial",13,"bold"),relief=RAISED,bd=3,c
ommand=filtering,state=DISABLED)
    submit_btn.place(x=330,y=87)


    def operate(ind):
      if ind:
        self.robo_prem = 1
        for player_index in range(2):
          self.total_people_play.append(player_index)
        print(self.total_people_play)
        def delay_with_instrctions(time_is):
          if place_ins['text'] != "":
            place_ins.place_forget()
          if command_play['text'] != "":
            command_play.place_forget()

          place_ins['text'] = f"The game will start in {time_is} sec"
          place_ins.place(x=20, y=220)

          if time_is > 5:
            command_play['text'] = f"          The machine uses red, and you use blue."
          elif time_is>= 2 and time_is<5:
            command_play['text'] = f"                Ready.....Steady....."
          else:
            command_play['text'] = f"                                GO"
          command_play.place(x=10, y=260)

        time_is = 10
        place_ins = Label(top, text="", font=("Arial", 20, "bold"), fg="#000000", bg="#92c1ec")
        command_play = Label(top, text="", font=("Arial", 12, "bold"), fg="red", bg="#92c1ec")
```

```
            try:
                while time_is:
                    delay_with_instrctions(time_is)
                    time_is-=1
                    self.window.update()
                    time.sleep(1)
                top.destroy()
            except:
                print("Force Stop Error in Operate")
            self.block_value_predict[1][1]['state'] = NORMAL
        else:
            submit_btn['state'] = NORMAL
            take_entry['state'] = NORMAL


    mvc_btn = Button(top,text="Play With Computer",bg="#FFFFFF",fg="#c90076",font=("Arial",15,"bold"),relief=RAISED,bd=3,command=lambda: operate(1), activebackground="#262626")
    mvc_btn.place(x=30,y=160)


    mvh_btn = Button(top,text="Play With Friends",bg="#FFFFFF",fg="#c90076",font=("Arial",15,"bold"),relief=RAISED,bd=3,command=lambda: operate(0), activebackground="#262626")
    mvh_btn.place(x=260,y=160)


    top.mainloop()


# Get block value after prediction based on probability
def make_prediction(self,color_indicator):
    try:
        if color_indicator == "red":
            block_value_predict = self.block_value_predict[0]
            if self.robo_prem and self.count_robo_stage_from_start < 3:
```

```
            self.count_robo_stage_from_start += 1
        if self.robo_prem and self.count_robo_stage_from_start == 3 and self.six_counter < 2:
            permanent_block_number = self.move_red_counter = 6
            self.count_robo_stage_from_start += 1
        else:
            permanent_block_number = self.move_red_counter = randint(1, 6)

    elif color_indicator == "blue":
        block_value_predict = self.block_value_predict[1]
        permanent_block_number = self.move_blue_counter = randint(1, 6)
        if self.robo_prem and permanent_block_number == 6:
            for coin_loc in self.red_coin_position:
                if coin_loc>=40 and coin_loc<=46:
                    permanent_block_number = self.move_blue_counter = randint(1, 5)
                    break

    elif color_indicator == "pink":
        block_value_predict = self.block_value_predict[2]
        permanent_block_number = self.move_pink_counter = randint(1, 6)

    else:
        block_value_predict = self.block_value_predict[3]
        permanent_block_number = self.move_grn_counter = randint(1, 6)

    block_value_predict[1]['state'] = DISABLED

    # Illusion of coin floating
    temp_counter = 12
    while temp_counter>0:
        move_temp_counter = randint(1, 6)
        block_value_predict[0]['image'] = self.block_number_side[move_temp_counter - 1]
```

```
            self.window.update()
            time.sleep(0.1)
            temp_counter-=1

        print("Prediction result: ", permanent_block_number)

        # Permanent predicted value containing image set
        block_value_predict[0]['image'] = self.block_number_side[permanent_block_number-1]
        if self.robo_prem == 1 and color_indicator == "red":
            self.window.update()
            time.sleep(0.4)

self.instructional_btn_customization_based_on_current_situation(color_indicator,permanent_block
_number,block_value_predict)
    except:
        print("Force Stop Error in Prediction")


  def
instructional_btn_customization_based_on_current_situation(self,color_indicator,permanent_block
_number,block_value_predict):
    robo_operator = None
    if color_indicator == "red":
        temp_coin_position = self.red_coin_position
    elif color_indicator == "green":
        temp_coin_position = self.grn_coin_position
    elif color_indicator == "pink":
        temp_coin_position = self.pink_coin_position
    else:
        temp_coin_position = self.blue_coin_position

    all_in = 1
    for i in range(4):
```

```
    if temp_coin_position[i] == -1:
        all_in = 1
    else:
        all_in = 0
        break

if  permanent_block_number == 6:
    self.six_counter += 1
else:
    self.six_counter = 0

if ((all_in == 1 and permanent_block_number == 6) or (all_in==0)) and self.six_counter<3:
    permission = 1
    if color_indicator == "red":
        temp = self.red_coord_store
    elif color_indicator == "green":
        temp = self.grn_coord_store
    elif color_indicator == "pink":
        temp = self.pink_coord_store
    else:
        temp = self.blue_coord_store

    if  permanent_block_number<6:
        if self.six_with_overlap == 1:
            self.time_for-=1
            self.six_with_overlap=0
        for i in range(4):
            if  temp[i] == -1:
                permission=0
            elif temp[i]>100:
                if  temp[i]+permanent_block_number<=106:
```

```
                permission=1
                break
            else:
                permission=0
        else:
            permission=1
            break
    else:
        for i in range(4):
            if  temp[i]>100:
                if  temp[i] + permanent_block_number <= 106:
                    permission = 1
                    break
                else:
                    permission = 0
            else:
                permission = 1
                break
    if permission == 0:
        self.make_command(None)
    else:
        self.num_btns_state_controller(block_value_predict[2])

        if self.robo_prem == 1 and block_value_predict == self.block_value_predict[0]:
            robo_operator = "give"
        block_value_predict[1]['state'] = DISABLED# Predict btn deactivation

else:
    block_value_predict[1]['state'] = NORMAL# Predict btn activation
    if self.six_with_overlap == 1:
        self.time_for -= 1
```

```
            self.six_with_overlap = 0
        self.make_command()

    if  permanent_block_number == 6 and self.six_counter<3 and block_value_predict[2][0]['state']
== NORMAL:
        self.time_for-=1
    else:
        self.six_counter=0

    if self.robo_prem == 1 and robo_operator:
        self.robo_judge(robo_operator)

  # Player Scope controller
  def make_command(self, robo_operator=None):
    if  self.time_for == -1:
        pass
    else:
        self.block_value_predict[self.total_people_play[self.time_for]][1]['state'] = DISABLED
    if  self.time_for == len(self.total_people_play)-1:
        self.time_for = -1

    self.time_for+=1
    self.block_value_predict[self.total_people_play[self.time_for]][1]['state'] = NORMAL

    if self.robo_prem==1 and self.time_for == 0:
        robo_operator = "predict"
    if robo_operator:
        self.robo_judge(robo_operator)



  def instruction_btn_red(self):
```

```
        block_predict_red = Label(self.make_canvas,image=self.block_number_side[0])

        block_predict_red.place(x=34,y=15)

        predict_red = Button(self.make_canvas, bg="black", fg="#8fce73", relief=RAISED, bd=5,
text="Predict", font=("Arial", 8, "bold"), command=lambda: self.make_prediction("red"))

        predict_red.place(x=25, y=15 + 50)



        btn_1 =
Button(self.make_canvas,bg="#262626",fg="#00eb00",text="1",font=("Arial",13,"bold","italic"),relief
=RAISED,bd=3,command=lambda: self.main_controller("red",'1'), state=DISABLED,
disabledforeground="red")

        btn_1.place(x=20,y=15+100)

        btn_2 =
Button(self.make_canvas,bg="#262626",fg="#00eb00",text="2",font=("Arial",13,"bold","italic"),relief
=RAISED,bd=3,command=lambda: self.main_controller("red",'2'), state=DISABLED,
disabledforeground="red")

        btn_2.place(x=60,y=15+100)

        btn_3 =
Button(self.make_canvas,bg="#262626",fg="#00eb00",text="3",font=("Arial",13,"bold","italic"),relief
=RAISED,bd=3,command=lambda: self.main_controller("red",'3'), state=DISABLED,
disabledforeground="red")

        btn_3.place(x=20,y=15+100+40)

        btn_4 =
Button(self.make_canvas,bg="#262626",fg="#00eb00",text="4",font=("Arial",13,"bold","italic"),relief
=RAISED,bd=3,command=lambda: self.main_controller("red",'4'), state=DISABLED,
disabledforeground="red")

        btn_4.place(x=60,y=15+100+40)



        Label(self.make_canvas,text="Player
1",bg="#141414",fg="red",font=("Arial",15,"bold")).place(x=15,y=15+140+50)

        self.store_instructional_btn(block_predict_red,predict_red,[btn_1,btn_2,btn_3,btn_4])


    def instruction_btn_blue(self):

        block_predict_blue = Label(self.make_canvas, image=self.block_number_side[0])

        block_predict_blue.place(x=34, y=15+(40*6+40*3)+10)

        predict_blue = Button(self.make_canvas, bg="black", fg="#8fce73", relief=RAISED, bd=5,
text="Predict",font=("Arial", 8, "bold"), command=lambda: self.make_prediction("blue"))
```

```
        predict_blue.place(x=25, y=15+(40*6+40*3)+40 + 20)


        btn_1 =
Button(self.make_canvas,bg="#262626",fg="#00eb00",text="1",font=("Arial",13,"bold","italic"),relief
=RAISED,bd=3,command=lambda: self.main_controller("blue",'1'), state=DISABLED,
disabledforeground="red")

        btn_1.place(x=20,y=15+(40*6+40*3)+40 + 70)

        btn_2 =
Button(self.make_canvas,bg="#262626",fg="#00eb00",text="2",font=("Arial",13,"bold","italic"),relief
=RAISED,bd=3,command=lambda: self.main_controller("blue",'2'), state=DISABLED,
disabledforeground="red")

        btn_2.place(x=60,y=15+(40*6+40*3)+40 + 70)

        btn_3 =
Button(self.make_canvas,bg="#262626",fg="#00eb00",text="3",font=("Arial",13,"bold","italic"),relief
=RAISED,bd=3,command=lambda: self.main_controller("blue",'3'), state=DISABLED,
disabledforeground="red")

        btn_3.place(x=20,y=15+(40*6+40*3)+40 + 70+ 40)

        btn_4 =
Button(self.make_canvas,bg="#262626",fg="#00eb00",text="4",font=("Arial",13,"bold","italic"),relief
=RAISED,bd=3,command=lambda: self.main_controller("blue",'4'), state=DISABLED,
disabledforeground="red")

        btn_4.place(x=60,y=15+(40*6+40*3)+40 + 70+ 40)


        Label(self.make_canvas, text="Player 2", bg="#141414", fg="red", font=("Arial", 15,
"bold")).place(x=12,y=15+(40*6+40*3)+40 + 110+50)

        self.store_instructional_btn(block_predict_blue, predict_blue, [btn_1,btn_2,btn_3,btn_4])


    def instruction_btn_pink(self):

        block_predict_pink = Label(self.make_canvas, image=self.block_number_side[0])

        block_predict_pink.place(x=100 + (40 * 6 + 40 * 3 + 40 * 6 + 10)+20, y=15 + (40 * 6 + 40 * 3) +
10)

        predict_pink = Button(self.make_canvas, bg="black", fg="#8fce73", relief=RAISED, bd=5,
text="Predict",font=("Arial", 8, "bold"), command=lambda: self.make_prediction("pink"))

        predict_pink.place(x=100 + (40 * 6 + 40 * 3 + 40 * 6 + 2)+20, y=15 + (40 * 6 + 40 * 3) + 40 + 20)


        btn_1 =
Button(self.make_canvas,bg="#262626",fg="#00eb00",text="1",font=("Arial",13,"bold","italic"),relief
```

```
=RAISED,bd=3,command=lambda: self.main_controller("pink",'1'), state=DISABLED,
disabledforeground="red")

    btn_1.place(x=100 + (40 * 6 + 40 * 3 + 40 * 6 + 2)+15, y=15 + (40 * 6 + 40 * 3) + 40 + 70)

    btn_2 =
Button(self.make_canvas,bg="#262626",fg="#00eb00",text="2",font=("Arial",13,"bold","italic"),relief
=RAISED,bd=3,command=lambda: self.main_controller("pink",'2'), state=DISABLED,
disabledforeground="red")

    btn_2.place(x=100 + (40 * 6 + 40 * 3 + 40 * 6 + 2)+15 + 40, y=15 + (40 * 6 + 40 * 3) + 40 + 70)

    btn_3 =
Button(self.make_canvas,bg="#262626",fg="#00eb00",text="3",font=("Arial",13,"bold","italic"),relief
=RAISED,bd=3,command=lambda: self.main_controller("pink",'3'), state=DISABLED,
disabledforeground="red")

    btn_3.place(x=100 + (40 * 6 + 40 * 3 + 40 * 6 + 2)+15, y=15 + (40 * 6 + 40 * 3) + 40 + 70+ 40)

    btn_4 =
Button(self.make_canvas,bg="#262626",fg="#00eb00",text="4",font=("Arial",13,"bold","italic"),relief
=RAISED,bd=3,command=lambda: self.main_controller("pink",'4'), state=DISABLED,
disabledforeground="red")

    btn_4.place(x=100 + (40 * 6 + 40 * 3 + 40 * 6 + 2)+15 + 40, y=15 + (40 * 6 + 40 * 3) + 40 + 70+ 40)


    Label(self.make_canvas, text="Player 3", bg="#141414", fg="red", font=("Arial", 15,
"bold")).place(x=100 + (40 * 6 + 40 * 3 + 40 * 6 +7),y=15+(40*6+40*3)+40 + 110+50)

    self.store_instructional_btn(block_predict_pink, predict_pink, [btn_1,btn_2,btn_3,btn_4])


  def instruction_btn_grn(self):

    block_predict_grn = Label(self.make_canvas, image=self.block_number_side[0])

    block_predict_grn.place(x=100+(40*6+40*3+40*6+10)+20, y=15)

    predict_grn = Button(self.make_canvas, bg="black", fg="#8fce73", relief=RAISED, bd=5,
text="Predict", font=("Arial", 8, "bold"), command=lambda: self.make_prediction("green"))

    predict_grn.place(x=100+(40*6+40*3+40*6+2)+20, y=15 + 50)


    btn_1 =
Button(self.make_canvas,bg="#262626",fg="#00eb00",text="1",font=("Arial",13,"bold","italic"),relief
=RAISED,bd=3,command=lambda: self.main_controller("green",'1'), state=DISABLED,
disabledforeground="red")

    btn_1.place(x=100 + (40 * 6 + 40 * 3 + 40 * 6 + 2)+15,y=15+100)

    btn_2 =
Button(self.make_canvas,bg="#262626",fg="#00eb00",text="2",font=("Arial",13,"bold","italic"),relief
```

```
=RAISED,bd=3,command=lambda: self.main_controller("green",'2'), state=DISABLED,
disabledforeground="red")

    btn_2.place(x=100 + (40 * 6 + 40 * 3 + 40 * 6 + 2)+15 + 40,y=15+100)

    btn_3 =
Button(self.make_canvas,bg="#262626",fg="#00eb00",text="3",font=("Arial",13,"bold","italic"),relief
=RAISED,bd=3,command=lambda: self.main_controller("green",'3'), state=DISABLED,
disabledforeground="red")

    btn_3.place(x=100 + (40 * 6 + 40 * 3 + 40 * 6 + 2)+15,y=15+100+40)

    btn_4 =
Button(self.make_canvas,bg="#262626",fg="#00eb00",text="4",font=("Arial",13,"bold","italic"),relief
=RAISED,bd=3,command=lambda: self.main_controller("green",'4'), state=DISABLED,
disabledforeground="red")

    btn_4.place(x=100 + (40 * 6 + 40 * 3 + 40 * 6 + 2)+15 + 40,y=15+100+40)


    Label(self.make_canvas, text="Player 4", bg="#141414", fg="red", font=("Arial", 15,
"bold")).place(x=100+(40*6+40*3+40*6+7), y=15+140+50)

    self.store_instructional_btn(block_predict_grn, predict_grn, [btn_1,btn_2,btn_3,btn_4])


    hint_button = Button(self.make_canvas, text="Hint", bg="red", fg="white", font=("Arial", 13,
"bold"),

                relief=RAISED, bd=3, command=self.show_hint)

    hint_button.place(x=100 + (40 * 6 + 40 * 3 + 40 * 6 + 7), y=15 + 140 + 150)


    self.store_instructional_btn(block_predict_grn, predict_grn, [btn_1, btn_2, btn_3, btn_4])




  def show_hint(self):


    current_player = "red"  # Assume it's red's turn for this example


    # Get the AI-predicted best move for the player

    best_move = self.predict_best_move(current_player)
```

```
        # Display the AI-driven hint
        messagebox.showinfo("AI Hint", best_move)


    # Predict the best move based on simple AI decision-making
    def predict_best_move(self, color):
        # Get the positions of the player's coins
        if color == "red":
            coin_positions = self.red_coin_position
        elif color == "green":
            coin_positions = self.grn_coin_position
        elif color == "pink":
            coin_positions = self.pink_coin_position
        else:  color== "blue"
        coin_positions = self.blue_coin_position

        # Evaluate the risk of each coin
        risks = self.evaluate_risks(color)

        # Store possible moves with associated risk and reward factors
        move_evaluations = []

        # Loop through each coin and evaluate the possible moves
        for i, position in enumerate(coin_positions):
            if position == -1:
                # Coin is in the base, needs a 6 to move
                move_evaluations.append((f"Coin {i+1} can start moving with a 6", 0.166))  # Probability of
rolling 6
            else:
                # Simulate two future dice rolls
                roll_1 = random.randint(1, 6)
```

```
            roll_2 = random.randint(1, 6)

            # Evaluate the move for each roll
            move_1, reward_1 = self.evaluate_move(position, roll_1, risks[i])
            move_2, reward_2 = self.evaluate_move(position, roll_2, risks[i])

            move_evaluations.append((f"Coin {i+1} can move to position {position + roll_1} with roll {roll_1}", reward_1))
            move_evaluations.append((f"Coin {i+1} can move to position {position + roll_2} with roll {roll_2}", reward_2))

        # Select the best move based on the highest reward score
        best_move = max(move_evaluations, key=lambda x: x[1])
        return f"AI suggests: {best_move[0]}"



#     def get_game_state(self):
#     # Example: list of coin positions, current player turn, etc.
#         return [self.red_coin_position, self.green_coin_position, self.pink_coin_position, self.sky_blue_coin_position, self.current_player]


   #  class MCTSNode:
#      def __init__(self, state, parent=None):
#         self.state = state
#         self.parent = parent
#         self.children = []
#         self.visits = 0
#         self.value = 0.0

#     def mcts(root_node, num_simulations):
#      for _ in range(num_simulations):
#         leaf = select_leaf_node(root_node)
```

```
#         simulation_result = simulate_game_from_node(leaf)
#         backpropagate(leaf, simulation_result)

#     def select_leaf_node(node):
#     # Selection strategy: Upper Confidence Bound (UCB)
#         pass

#     def simulate_game_from_node(node):
#     # Play random moves until the game ends, return the result
#         pass

#     def backpropagate(node, result):
#     # Update values and visits based on the result
#         pass




# def make_prediction(self, color_indicator):
#     # MCTS logic
#     root_node = MCTSNode(get_game_state())
#     best_move = mcts(root_node, num_simulations=1000)
#     # Update the game with the best move




# def build_q_network(input_shape, output_shape):
#     model = tf.keras.Sequential()
#     model.add(layers.Input(shape=input_shape))
#     model.add(layers.Dense(128, activation='relu'))
#     model.add(layers.Dense(128, activation='relu'))
#     model.add(layers.Dense(output_shape))
```

```
#     return model


# def train_q_network():
#     # Training loop to update the Q-Network using experiences (states, actions, rewards)
#     pass


# def make_prediction(self, color_indicator):
#     state = get_game_state()
#     q_values = q_network.predict(state)
#     best_move = np.argmax(q_values)  # Select the action with the highest Q-value
#     # Perform the move in the game









    # Evaluate the risk for each coin (e.g., if it's at risk of being captured)
    def evaluate_risks(self, color):
        # Calculate risks based on opponent positions
        if color == "red":
            opponent_positions = self.grn_coin_position + self.pink_coin_position +
self.blue_coin_position
        elif color == "green":
            opponent_positions = self.red_coin_position + self.pink_coin_position +
self.blue_coin_position
        elif color == "pink":
```

```
        opponent_positions = self.red_coin_position + self.grn_coin_position + self.blue_coin_position
    else:
        opponent_positions = self.red_coin_position + self.grn_coin_position + self.pink_coin_position

    risks = []
    for i, position in enumerate(opponent_positions):
        if position == -1:
            risks.append(0)  # No risk if the opponent's coin is at start
        else:
            risk_value = self.calculate_risk(position)
            risks.append(risk_value)

    return risks


# Calculate the risk based on proximity to opponent's coins
def calculate_risk(self, position):
    risk_factor = 0

    # Assume a risk zone within 6 steps of the player's coin (adjust for actual game rules)
    for opponent_position in range(position - 6, position + 6):
        if opponent_position == position:
            risk_factor += 1  # Higher risk if an opponent is nearby

    return risk_factor


# Evaluate each move based on reward (distance to home) and risk (opponent proximity)
def evaluate_move(self, position, roll, risk):
    new_position = position + roll
    reward = 0
```

```
    # Moves that advance the coin closer to the home zone should be given priority.
    if new_position >= 106:
        reward += 10
    elif new_position >= 50:
        reward += 5

    # Penalize moves that put the coin at risk of being captured
    if risk > 0:
        reward -= risk * 2  # Penalize heavily for high-risk positions

    return new_position, reward

  def store_instructional_btn(self, block_indicator, predictor, entry_controller):
    temp = []
    temp.append(block_indicator)
    temp.append(predictor)
    temp.append(entry_controller)
    self.block_value_predict.append(temp)

  def red_circle_start_position(self, coin_number):
    self.make_canvas.delete(self.made_red_coin[int(coin_number)-1])
    self.made_red_coin[int(coin_number)-1] = self.make_canvas.create_oval(100 + 40, 15+(40*6),
100 +40 + 40, 15+(40*6)+40, fill="red", width=3, outline="black")

    self.red_number_label[int(coin_number)-1].place_forget()
    red_start_label_x = 100 + 40 + 10
    red_start_label_y = 15 + (40 * 6) + 5
    self.red_number_label[int(coin_number)-1].place(x=red_start_label_x, y=red_start_label_y)

    self.red_coin_position[int(coin_number)-1] = 1
```

```
        self.window.update()
        time.sleep(0.2)

    def grn_circle_start_position(self,coin_number):
        self.make_canvas.delete(self.made_grn_coin[int(coin_number)-1])
        self.made_grn_coin[int(coin_number)-1] = self.make_canvas.create_oval(100 + (40*8), 15 + 40,
100 +(40*9), 15 + 40+ 40, fill="#8fce73", width=3)

        self.grn_number_label[int(coin_number)-1].place_forget()
        grn_start_label_x = 100 + (40*8) + 10
        grn_start_label_y = 15 + 40 + 5
        self.grn_number_label[int(coin_number)-1].place(x=grn_start_label_x, y=grn_start_label_y)

        self.grn_coin_position[int(coin_number)-1] = 14
        self.window.update()
        time.sleep(0.2)

    def pink_circle_start_position(self,coin_number):
        self.make_canvas.delete(self.made_pink_coin[int(coin_number)-1])
        self.made_pink_coin[int(coin_number)-1] = self.make_canvas.create_oval(100 + (40 *
6)+(40*3)+(40*4), 15 + (40*8), 100 + (40 * 6)+(40*3)+(40*5), 15 + (40*9), fill="pink", width=3)

        self.pink_number_label[int(coin_number)-1].place_forget()
        pink_start_label_x = 100 + (40 * 6)+(40*3)+(40*4) + 10
        pink_start_label_y = 15 + (40*8) + 5
        self.pink_number_label[int(coin_number) - 1].place(x=pink_start_label_x, y=pink_start_label_y)

        self.pink_coin_position[int(coin_number) - 1] = 27
        self.window.update()
        time.sleep(0.2)

    def blue_circle_start_position(self,coin_number):
```

```
        self.make_canvas.delete(self.made_blue_coin[int(coin_number)-1])
        self.made_blue_coin[int(coin_number)-1] = self.make_canvas.create_oval(100+240,340+(40*5)-5,100+240+40,340+(40*6)-5,fill="#073763",width=3)

        self.blue_number_label[int(coin_number)-1].place_forget()
        blue_start_label_x = 100+240 + 10
        blue_start_label_y = 340+(40*5)-5 + 5
        self.blue_number_label[int(coin_number) - 1].place(x=blue_start_label_x, y=blue_start_label_y)

        self.blue_coin_position[int(coin_number) - 1] = 40
        self.window.update()
        time.sleep(0.2)

    def num_btns_state_controller(self, take_nums_btns_list, state_control = 1):
        if state_control:
            for num_btn in take_nums_btns_list:
                num_btn['state'] = NORMAL
        else:
            for num_btn in take_nums_btns_list:
                num_btn['state'] = DISABLED

    def main_controller(self, color_coin, coin_number):
        robo_operator = None

        if  color_coin == "red":
            self.num_btns_state_controller(self.block_value_predict[0][2], 0)

            if self.move_red_counter == 106:
                messagebox.showwarning("Destination reached","Reached at the destination")

            elif self.red_coin_position[int(coin_number)-1] == -1 and self.move_red_counter == 6:
```

```
            self.red_circle_start_position(coin_number)
            self.red_coord_store[int(coin_number) - 1] = 1

        elif self.red_coin_position[int(coin_number)-1] > -1:
            take_coord = self.make_canvas.coords(self.made_red_coin[int(coin_number)-1])
            red_start_label_x = take_coord[0] + 10
            red_start_label_y = take_coord[1] + 5
            self.red_number_label[int(coin_number) - 1].place(x=red_start_label_x,
y=red_start_label_y)

            if self.red_coin_position[int(coin_number)-1]+self.move_red_counter<=106:
                self.red_coin_position[int(coin_number)-1] =
self.motion_of_coin(self.red_coin_position[int(coin_number) -
1],self.made_red_coin[int(coin_number)-1],self.red_number_label[int(coin_number)-
1],red_start_label_x,red_start_label_y,"red",self.move_red_counter)
                if self.robo_prem and self.red_coin_position[int(coin_number)-1] == 106 and color_coin
== "red":
                    self.robo_store.remove(int(coin_number))
                    print("After removing: ", self.robo_store)

            else:
                if not self.robo_prem:
                    messagebox.showerror("Not possible","Sorry, not permitted")
                self.num_btns_state_controller(self.block_value_predict[0][2])

                if self.robo_prem:
                    robo_operator = "give"
                    self.robo_judge(robo_operator)
                return

            if  self.red_coin_position[int(coin_number)-1]==22 or
self.red_coin_position[int(coin_number)-1]==9 or self.red_coin_position[int(coin_number)-1]==48 or
self.red_coin_position[int(coin_number)-1]==35 or self.red_coin_position[int(coin_number)-1]==14
```

```
or self.red_coin_position[int(coin_number)-1]==27 or self.red_coin_position[int(coin_number)-1]==40 or self.red_coin_position[int(coin_number)-1]==1:
                pass
            else:
                if self.red_coin_position[int(coin_number) - 1] < 100:
                    self.coord_overlap(self.red_coin_position[int(coin_number)-1],color_coin, self.move_red_counter)

            self.red_coord_store[int(coin_number)-1] = self.red_coin_position[int(coin_number)-1]

        else:
            messagebox.showerror("Ooops","your coin cannot travel.")
            self.num_btns_state_controller(self.block_value_predict[0][2])

            if self.robo_prem == 1:
                robo_operator = "give"
                self.robo_judge(robo_operator)
            return

        self.block_value_predict[0][1]['state'] = NORMAL


    elif color_coin == "green":
        self.num_btns_state_controller(self.block_value_predict[3][2], 0)

        if self.move_grn_counter == 106:
            messagebox.showwarning("Destination reached","Reached at the destination")

        elif self.grn_coin_position[int(coin_number) - 1] == -1 and self.move_grn_counter == 6:
            self.grn_circle_start_position(coin_number)
            self.grn_coord_store[int(coin_number) - 1] = 14
```

```
        elif self.grn_coin_position[int(coin_number) - 1] > -1:
            take_coord = self.make_canvas.coords(self.made_grn_coin[int(coin_number) - 1])
            grn_start_label_x = take_coord[0] + 10
            grn_start_label_y = take_coord[1] + 5
            self.grn_number_label[int(coin_number) - 1].place(x=grn_start_label_x,
y=grn_start_label_y)



            if  self.grn_coin_position[int(coin_number) - 1] + self.move_grn_counter <= 106:
                self.grn_coin_position[int(coin_number) - 1] =
self.motion_of_coin(self.grn_coin_position[int(coin_number) - 1],
self.made_grn_coin[int(coin_number) - 1], self.grn_number_label[int(coin_number) - 1],
grn_start_label_x, grn_start_label_y, "green", self.move_grn_counter)
            else:
              messagebox.showerror("Not possible","No path available")
              self.num_btns_state_controller(self.block_value_predict[3][2])
              return



            if  self.grn_coin_position[int(coin_number)-1]==22 or
self.grn_coin_position[int(coin_number)-1]==9 or self.grn_coin_position[int(coin_number)-1]==48 or
self.grn_coin_position[int(coin_number)-1]==35 or self.grn_coin_position[int(coin_number)-1]==1 or
self.grn_coin_position[int(coin_number)-1]==27 or self.grn_coin_position[int(coin_number)-1]==40
or self.grn_coin_position[int(coin_number)-1]==14:
                pass
            else:
              if self.grn_coin_position[int(coin_number) - 1] < 100:
                  self.coord_overlap(self.grn_coin_position[int(coin_number) - 1],color_coin,
self.move_grn_counter)


            self.grn_coord_store[int(coin_number) - 1] = self.grn_coin_position[int(coin_number) - 1]


        else:
            messagebox.showerror("Ooops", "your coin cannot travel.")
```

```
            self.num_btns_state_controller(self.block_value_predict[3][2])
            return

        self.block_value_predict[3][1]['state'] = NORMAL

    elif color_coin == "pink":

        self.num_btns_state_controller(self.block_value_predict[2][2], 0)

        if self.move_pink_counter == 106:
            messagebox.showwarning("Destination reached","Reached at the destination")

        elif self.pink_coin_position[int(coin_number) - 1] == -1 and self.move_pink_counter == 6:
            self.pink_circle_start_position(coin_number)
            self.pink_coord_store[int(coin_number) - 1] = 27

        elif self.pink_coin_position[int(coin_number) - 1] > -1:
            take_coord = self.make_canvas.coords(self.made_pink_coin[int(coin_number) - 1])
            pink_start_label_x = take_coord[0] + 10
            pink_start_label_y = take_coord[1] + 5
            self.pink_number_label[int(coin_number) - 1].place(x=pink_start_label_x,
y=pink_start_label_y)

            if  self.pink_coin_position[int(coin_number) - 1] + self.move_pink_counter <= 106:
                self.pink_coin_position[int(coin_number) - 1] =
self.motion_of_coin(self.pink_coin_position[int(coin_number) - 1],
self.made_pink_coin[int(coin_number) - 1], self.pink_number_label[int(coin_number) - 1],
pink_start_label_x, pink_start_label_y, "pink", self.move_pink_counter)
            else:
                messagebox.showerror("Not possible","No path available")

                self.num_btns_state_controller(self.block_value_predict[2][2])
```

```
                return


            if  self.pink_coin_position[int(coin_number)-1]==22 or
self.pink_coin_position[int(coin_number)-1]==9 or self.pink_coin_position[int(coin_number)-1]==48
or self.pink_coin_position[int(coin_number)-1]==35 or self.pink_coin_position[int(coin_number)-
1]==1 or self.pink_coin_position[int(coin_number)-1]==14 or
self.pink_coin_position[int(coin_number)-1]==40 or self.pink_coin_position[int(coin_number)-
1]==27:

                pass

            else:

                if self.pink_coin_position[int(coin_number) - 1] < 100:

                    self.coord_overlap(self.pink_coin_position[int(coin_number) - 1],color_coin,
self.move_pink_counter)


                self.pink_coord_store[int(coin_number) - 1] = self.pink_coin_position[int(coin_number) - 1]


        else:

            messagebox.showerror("Ooops", "your coin cannot travel")

            self.num_btns_state_controller(self.block_value_predict[2][2])

            return


        self.block_value_predict[2][1]['state'] = NORMAL



    elif color_coin == "blue":

        self.num_btns_state_controller(self.block_value_predict[1][2], 0)


        if self.move_red_counter == 106:

            messagebox.showwarning("Destination reached","Reached at the destination")


        elif self.blue_coin_position[int(coin_number) - 1] == -1 and self.move_blue_counter == 6:

            self.blue_circle_start_position(coin_number)

            self.blue_coord_store[int(coin_number) - 1] = 40
```

```
        elif self.blue_coin_position[int(coin_number) - 1] > -1:
            take_coord = self.make_canvas.coords(self.made_blue_coin[int(coin_number) - 1])
            blue_start_label_x = take_coord[0] + 10
            blue_start_label_y = take_coord[1] + 5
            self.blue_number_label[int(coin_number) - 1].place(x=blue_start_label_x,
y=blue_start_label_y)


            if  self.blue_coin_position[int(coin_number) - 1] + self.move_blue_counter <= 106:
                self.blue_coin_position[int(coin_number) - 1] =
self.motion_of_coin(self.blue_coin_position[int(coin_number) - 1],
self.made_blue_coin[int(coin_number) - 1], self.blue_number_label[int(coin_number) - 1],
blue_start_label_x, blue_start_label_y, "blue", self.move_blue_counter)
            else:
              messagebox.showerror("Not possible","No path available")


              self.num_btns_state_controller(self.block_value_predict[1][2])
              return


            if  self.blue_coin_position[int(coin_number)-1]==22 or
self.blue_coin_position[int(coin_number)-1]==9 or self.blue_coin_position[int(coin_number)-1]==48
or self.blue_coin_position[int(coin_number)-1]==35 or self.blue_coin_position[int(coin_number)-
1]==1 or self.blue_coin_position[int(coin_number)-1]==14 or
self.blue_coin_position[int(coin_number)-1]==27 or self.blue_coin_position[int(coin_number)-
1]==40:
                pass
            else:
                if self.blue_coin_position[int(coin_number) - 1] < 100:
                    self.coord_overlap(self.blue_coin_position[int(coin_number) - 1],color_coin,
self.move_blue_counter)


            self.blue_coord_store[int(coin_number) - 1] = self.blue_coin_position[int(coin_number) - 1]


        else:
```

```
            messagebox.showerror("Ooops", "your coin cannot travel")
            self.num_btns_state_controller(self.block_value_predict[1][2])
            return

        self.block_value_predict[1][1]['state'] = NORMAL

    print(self.red_coord_store)
    print(self.grn_coord_store)
    print(self.pink_coord_store)
    print(self.blue_coord_store)
    if self.robo_prem == 1:
        print("Robo Store is: ", self.robo_store)

    permission_granted_to_proceed = True

    if  color_coin == "red" and self.red_coin_position[int(coin_number)-1] == 106:
        permission_granted_to_proceed = self.check_winner_and_runner(color_coin)
    elif  color_coin == "green" and self.grn_coin_position[int(coin_number)-1] == 106:
        permission_granted_to_proceed = self.check_winner_and_runner(color_coin)
    elif  color_coin == "pink" and self.pink_coin_position[int(coin_number)-1] == 106:
        permission_granted_to_proceed = self.check_winner_and_runner(color_coin)
    elif  color_coin == "blue" and self.blue_coin_position[int(coin_number)-1] == 106:
        permission_granted_to_proceed = self.check_winner_and_runner(color_coin)

    if permission_granted_to_proceed:# if that is False, Game is over and not proceed more
        self.make_command(robo_operator)

  def motion_of_coin(self,counter_coin,specific_coin,number_label,number_label_x
,number_label_y,color_coin,path_counter):
    try:
        number_label.place(x=number_label_x,y=number_label_y)
```

```
        while True:
            if path_counter == 0:
                break
            elif (counter_coin == 51 and color_coin == "red") or (counter_coin==12 and color_coin == "green") or (counter_coin == 25 and color_coin == "pink") or (counter_coin == 38 and color_coin == "blue") or counter_coin>=100:
                if counter_coin<100:
                    counter_coin=100

                counter_coin = self.under_room_traversal_control(specific_coin, number_label, number_label_x, number_label_y, path_counter, counter_coin, color_coin)

                if  counter_coin == 106:

                    if self.robo_prem == 1 and color_coin == "red":
                        messagebox.showinfo("Destination reached","Hey! I am at the destination")
                    else:
                        messagebox.showinfo("Destination reached","Congrats! You now at the destination")
                    if path_counter == 6:
                        self.six_with_overlap = 1
                    else:
                        self.time_for -= 1
                break

            counter_coin += 1
            path_counter -=1
            number_label.place_forget()

            print(counter_coin)

            if counter_coin<=5:
                self.make_canvas.move(specific_coin, 40, 0)
```

```
    number_label_x+=40
elif counter_coin == 6:
    self.make_canvas.move(specific_coin, 40, -40)
    number_label_x += 40
    number_label_y-=40
elif 6< counter_coin <=11:
    self.make_canvas.move(specific_coin, 0, -40)
    number_label_y -= 40
elif counter_coin <=13:
    self.make_canvas.move(specific_coin, 40, 0)
    number_label_x += 40
elif counter_coin <=18:
    self.make_canvas.move(specific_coin, 0, 40)
    number_label_y += 40
elif counter_coin == 19:
    self.make_canvas.move(specific_coin, 40, 40)
    number_label_x += 40
    number_label_y += 40
elif counter_coin <=24:
    self.make_canvas.move(specific_coin, 40, 0)
    number_label_x += 40
elif counter_coin <=26:
    self.make_canvas.move(specific_coin, 0, 40)
    number_label_y += 40
elif counter_coin <=31:
    self.make_canvas.move(specific_coin, -40, 0)
    number_label_x -= 40
elif counter_coin == 32:
    self.make_canvas.move(specific_coin, -40, 40)
    number_label_x -= 40
    number_label_y += 40
```

```
        elif counter_coin <= 37:
            self.make_canvas.move(specific_coin, 0, 40)
            number_label_y += 40
        elif counter_coin <= 39:
            self.make_canvas.move(specific_coin, -40, 0)
            number_label_x -= 40
        elif counter_coin <= 44:
            self.make_canvas.move(specific_coin, 0, -40)
            number_label_y -= 40
        elif counter_coin == 45:
            self.make_canvas.move(specific_coin, -40, -40)
            number_label_x -= 40
            number_label_y -= 40
        elif counter_coin <= 50:
            self.make_canvas.move(specific_coin, -40, 0)
            number_label_x -= 40
        elif 50< counter_coin <=52:
            self.make_canvas.move(specific_coin, 0, -40)
            number_label_y -= 40
        elif counter_coin == 53:
            self.make_canvas.move(specific_coin, 40, 0)
            number_label_x += 40
            counter_coin = 1

        number_label.place_forget()
        number_label.place(x=number_label_x, y=number_label_y)

        self.window.update()
        time.sleep(0.2)

    return counter_coin
```

```
        except:
            print("Force Stop Error Came in motion of coin")


    # For same position, previous coin deleted and set to the room
    def coord_overlap(self, counter_coin, color_coin, path_to_traverse_before_overlap):
        if  color_coin!="red":
            for take_coin_number in range(len(self.red_coord_store)):
                if  self.red_coord_store[take_coin_number] == counter_coin:
                    if path_to_traverse_before_overlap == 6:
                        self.six_with_overlap=1
                    else:
                        self.time_for-=1


                    self.make_canvas.delete(self.made_red_coin[take_coin_number])
                    self.red_number_label[take_coin_number].place_forget()
                    self.red_coin_position[take_coin_number] = -1
                    self.red_coord_store[take_coin_number] = -1
                    if self.robo_prem == 1:
                        self.robo_store.remove(take_coin_number+1)
                        if self.red_coin_position.count(-1)>=1:
                            self.count_robo_stage_from_start = 2


                    if take_coin_number == 0:
                       remade_coin = self.make_canvas.create_oval(100+40, 15+40, 100+40+40, 15+40+40,
width=3, fill="red", outline="black")
                       self.red_number_label[take_coin_number].place(x=100 + 40 + 10, y=15 + 40 + 5)
                    elif take_coin_number == 1:
                        remade_coin = self.make_canvas.create_oval(100+40+60+60, 15 + 40,
100+40+60+60+40, 15 + 40 + 40, width=3, fill="red", outline="black")
                        self.red_number_label[take_coin_number].place(x=100 + 40 + 60 +60 + 10, y=15 + 40 +
5)
                    elif take_coin_number == 2:
```

```
                remade_coin = self.make_canvas.create_oval(100 + 40 + 60 + 60, 15 + 40 + 100, 100 +
40 + 60 + 60 + 40, 15 + 40 + 40 + 100, width=3, fill="red", outline="black")

                self.red_number_label[take_coin_number].place(x=100 + 40 + 60 + 60 + 10, y=15 + 40 +
100 + 5)

            else:

                remade_coin = self.make_canvas.create_oval(100 + 40, 15 + 40+100, 100 + 40 + 40, 15
+ 40 + 40+100, width=3,fill="red", outline="black")

                self.red_number_label[take_coin_number].place(x=100 + 40 + 10, y=15 + 40 + 100 + 5)


            self.made_red_coin[take_coin_number]=remade_coin


    if  color_coin != "green":

        for take_coin_number in range(len(self.grn_coord_store)):

            if  self.grn_coord_store[take_coin_number] == counter_coin:

                if path_to_traverse_before_overlap == 6:

                    self.six_with_overlap = 1

                else:

                    self.time_for-=1


                self.make_canvas.delete(self.made_grn_coin[take_coin_number])

                self.grn_number_label[take_coin_number].place_forget()

                self.grn_coin_position[take_coin_number] = -1

                self.grn_coord_store[take_coin_number] = -1


                if take_coin_number == 0:

                    remade_coin = self.make_canvas.create_oval(340+(40*3)+40, 15 + 40, 340+(40*3)+40 +
40, 15 + 40 + 40, width=3, fill="#8fce73", outline="black")

                    self.grn_number_label[take_coin_number].place(x=340 + (40 * 3) + 40 + 10, y=15 + 40
+ 5)

                elif take_coin_number == 1:

                    remade_coin = self.make_canvas.create_oval(340+(40*3)+40+ 60 + 40+20, 15 + 40,
340+(40*3)+40 + 60 + 40 + 40+20, 15 + 40 + 40, width=3, fill="#8fce73", outline="black")
```

```
                self.grn_number_label[take_coin_number].place(x=340 + (40 * 3) + 40 + 40 + 60 + 30,
y=15 + 40 + 5)

            elif take_coin_number == 2:

                remade_coin = self.make_canvas.create_oval(340 + (40 * 3) + 40 + 60 + 40 + 20, 15 + 40
+ 100, 340 + (40 * 3) + 40 + 60 + 40 + 40 + 20, 15 + 40 + 40 + 100, width=3, fill="#8fce73",
outline="black")

                self.grn_number_label[take_coin_number].place(x=340 + (40 * 3) + 40 + 40 + 60 + 30,
y=15 + 40 + 100 + 5)

            else:

                remade_coin = self.make_canvas.create_oval(340+(40*3)+40, 15 + 40 + 100,
340+(40*3)+40 + 40, 15 + 40 + 40 + 100, width=3, fill="#8fce73", outline="black")

                self.grn_number_label[take_coin_number].place(x=340+(40*3) + 40 + 10, y=15 + 40 +
100 + 5)


            self.made_grn_coin[take_coin_number] = remade_coin



    if  color_coin != "pink":

        for take_coin_number in range(len(self.pink_coord_store)):

            if  self.pink_coord_store[take_coin_number] == counter_coin:

                if path_to_traverse_before_overlap == 6:

                    self.six_with_overlap = 1

                else:

                    self.time_for -= 1


                self.make_canvas.delete(self.made_pink_coin[take_coin_number])

                self.pink_number_label[take_coin_number].place_forget()

                self.pink_coin_position[take_coin_number] = -1

                self.pink_coord_store[take_coin_number] = -1


                if take_coin_number == 0:

                    remade_coin = self.make_canvas.create_oval(340 + (40 * 3) + 40, 340+80+15, 340 + (40
* 3) + 40 + 40, 340+80+40+15, width=3, fill="pink", outline="black")
```

```
                self.pink_number_label[take_coin_number].place(x=340+(40*3) + 40 + 10, y=30 +
(40*6)+(40*3)+40+10)

            elif take_coin_number == 1:

                remade_coin = self.make_canvas.create_oval(340 + (40 * 3) + 40 + 60 + 40 + 20,
340+80+15, 340 + (40 * 3) + 40 + 60 + 40 + 40+20, 340+80+40+15, width=3, fill="pink",
outline="black")

                self.pink_number_label[take_coin_number].place(x=340+(40*3)+ 40 + 40+ 60 + 30,
y=30 + (40*6)+(40*3)+40+10)

            elif take_coin_number == 2:

                remade_coin = self.make_canvas.create_oval(340 + (40 * 3) + 40 + 60 + 40 + 20, 340 +
80 + 60 + 40 + 15, 340 + (40 * 3) + 40 + 60 + 40 + 40 + 20, 340 + 80 + 60 + 40 + 40 + 15, width=3,
fill="pink", outline="black")

                self.pink_number_label[take_coin_number].place(x=340+(40*3)+ 40 + 40+ 60 + 30,
y=30 + (40*6)+(40*3)+40+100+10)

            else:

                remade_coin = self.make_canvas.create_oval(340 + (40 * 3) + 40, 340+80+60+40+15,
340 + (40 * 3) + 40 + 40,340+80+60+40+40+15, width=3, fill="pink", outline="black")

                self.pink_number_label[take_coin_number].place(x=340 + (40 * 3) + 40 + 10, y=30 + (40
* 6) + (40 * 3) + 40 + 100 + 10)


            self.made_pink_coin[take_coin_number] = remade_coin


    if  color_coin != "blue":

        for take_coin_number in range(len(self.blue_coord_store)):

            if  self.blue_coord_store[take_coin_number] == counter_coin:

                if path_to_traverse_before_overlap == 6:

                    self.six_with_overlap = 1

                else:

                    self.time_for -= 1


                self.make_canvas.delete(self.made_blue_coin[take_coin_number])

                self.blue_number_label[take_coin_number].place_forget()

                self.blue_coin_position[take_coin_number] = -1

                self.blue_coord_store[take_coin_number]=-1
```

```
            if take_coin_number == 0:
                remade_coin = self.make_canvas.create_oval(100 + 40, 340+80+15, 100 + 40 + 40, 340+80+40+15, width=3, fill="#073763", outline="black")
                self.blue_number_label[take_coin_number].place(x=100+40+10, y=30 + (40*6)+(40*3)+40+10)
            elif take_coin_number == 1:
                remade_coin = self.make_canvas.create_oval(100 + 40 + 60 + 40+20, 340+80+15, 100 + 40 + 60 + 40 + 40+20, 340+80+40+15, width=3, fill="#073763", outline="black")
                self.blue_number_label[take_coin_number].place(x=100 + 40 + 60 +60 + 10, y=30 + (40*6)+(40*3)+40+10)
            elif take_coin_number == 2:
                remade_coin = self.make_canvas.create_oval(100 + 40 + 60 + 40 + 20, 340 + 80 + 60 + 40 + 15, 100 + 40 + 60 + 40 + 40 + 20, 340 + 80 + 60 + 40 + 40 + 15, width=3, fill="#073763", outline="black")
                self.blue_number_label[take_coin_number].place(x=100 + 40 + 60 + 60 + 10, y=30 + (40 * 6) + (40 * 3) + 40 + 60 + 40 + 10)
            else:
                remade_coin = self.make_canvas.create_oval( 100 + 40, 340+80+60+40+15, 100 + 40 + 40, 340+80+60+40+40+15, width=3, fill="#073763", outline="black")
                self.blue_number_label[take_coin_number].place(x=100+40+10, y=30 + (40*6)+(40*3)+40+60+40+10)

            self.made_blue_coin[take_coin_number] = remade_coin


    def under_room_traversal_control(self,specific_coin,number_label,number_label_x,number_label_y,path_counter,counter_coin,color_coin):
        if color_coin == "red" and counter_coin >= 100:
            if int(counter_coin)+int(path_counter)<=106:
                counter_coin = self.room_red_traversal(specific_coin, number_label, number_label_x, number_label_y, path_counter, counter_coin)

        elif color_coin == "green" and counter_coin >= 100:
```

```
            if  int(counter_coin) + int(path_counter) <= 106:
                counter_coin = self.room_grn_traversal(specific_coin, number_label, number_label_x,
number_label_y,path_counter,counter_coin)

        elif color_coin == "pink" and counter_coin >= 100:
            if  int(counter_coin) + int(path_counter) <= 106:
                counter_coin = self.room_pink_traversal(specific_coin, number_label, number_label_x,
number_label_y,path_counter,counter_coin)

        elif color_coin == "blue" and counter_coin >= 100:
            if  int(counter_coin) + int(path_counter) <= 106:
                counter_coin = self.room_blue_traversal(specific_coin, number_label, number_label_x,
number_label_y,path_counter,counter_coin)

        return counter_coin


    def room_red_traversal(self, specific_coin, number_label, number_label_x, number_label_y,
path_counter, counter_coin):
        while path_counter>0:
            counter_coin += 1
            path_counter -= 1
            self.make_canvas.move(specific_coin, 40, 0)
            number_label_x+=40
            number_label.place(x=number_label_x,y=number_label_y)
            self.window.update()
            time.sleep(0.2)
        return counter_coin

    def room_grn_traversal(self, specific_coin, number_label, number_label_x, number_label_y,
path_counter, counter_coin):
        while path_counter > 0:
            counter_coin += 1
```

```
            path_counter -= 1
            self.make_canvas.move(specific_coin, 0, 40)
            number_label_y += 40
            number_label.place(x=number_label_x, y=number_label_y)
            self.window.update()
            time.sleep(0.2)
        return counter_coin


    def room_pink_traversal(self, specific_coin, number_label, number_label_x,
number_label_y,path_counter,counter_coin):
        while path_counter > 0:
            counter_coin += 1
            path_counter -= 1
            self.make_canvas.move(specific_coin, -40, 0)
            number_label_x -= 40
            number_label.place(x=number_label_x, y=number_label_y)
            self.window.update()
            time.sleep(0.2)
        return counter_coin


    def room_blue_traversal(self, specific_coin, number_label, number_label_x,
number_label_y,path_counter,counter_coin):
        while path_counter > 0:
            counter_coin += 1
            path_counter -= 1
            self.make_canvas.move(specific_coin, 0, -40)
            number_label_y -= 40
            number_label.place(x=number_label_x, y=number_label_y)
            self.window.update()
            time.sleep(0.2)
        return counter_coin
```

```
def check_winner_and_runner(self,color_coin):
    destination_reached = 0 # Check for all specific color coins
    if color_coin == "red":
        temp_store = self.red_coord_store
        temp_delete = 0# Player index
    elif color_coin == "green":
        temp_store = self.grn_coord_store
        temp_delete = 3# Player index
    elif color_coin == "pink":
        temp_store = self.pink_coord_store
        temp_delete = 2# Player index
    else:
        temp_store = self.blue_coord_store
        temp_delete = 1# Player index


    for take in temp_store:
        if take == 106:
            destination_reached = 1
        else:
            destination_reached = 0
            break
            # winner and runner check
    if  destination_reached == 1:
        self.take_permission += 1
        if self.take_permission == 1:
            if self.robo_prem == 1 and color_coin == "red":
                messagebox.showinfo("Wohoo", "I am the winner")
            else:
                messagebox.showinfo("Wohoo","You are the winner")
        elif self.take_permission == 2:# 1st runner check
            if self.robo_prem == 1 and color_coin == "red":
```

```
                messagebox.showinfo("Wohoo","I am 1st runner")

            else:
                messagebox.showinfo("Wohoo", "Wow! You are 1st runner")
        elif self.take_permission == 3:# 2nd runner check
            if self.robo_prem == 1 and color_coin == "red":
                messagebox.showinfo("Result", "I am 2nd runner....Not bad at all")
            else:
                messagebox.showinfo("Result", "You are 2nd runner....Better Luck next time")

        self.block_value_predict[temp_delete][1]['state'] = DISABLED
        self.total_people_play.remove(temp_delete)

        if len(self.total_people_play) == 1:
            messagebox.showinfo("Game Over","Good bye!!!!")
            self.block_value_predict[0][1]['state'] = DISABLED
            return False
        else:
            self.time_for-=1
    else:
        print("Winner not decided")

    return True

def robo_judge(self, ind="give"):
    if ind == "give":# For give the value
        all_in = 1# Denoting all the coins are present in the room
        for i in range(4):
            if self.red_coin_position[i] == -1:
                all_in = 1
            else:
```

```
                all_in = 0# Denoting all the coins not present in the room
                break


            if all_in == 1:# All coins are present in room
                if self.move_red_counter == 6:
                    predicted_coin = choice([1,2,3,4])
                    self.robo_store.append(predicted_coin)
                    self.main_controller("red", predicted_coin)
                else:
                    pass
            else:# All coins not present in room
                temp = self.red_coin_position# Take red coin position reference
                take_ref = self.blue_coin_position# Take sky_blue coin position reference


                if len(self.robo_store) == 1:# When only one coin is outside of the room
                    if self.move_red_counter<6:# When prediction less than 6
                        if (self.count_robo_stage_from_start>3) and (temp[self.robo_store[0]-1] >=33 and
temp[self.robo_store[0]-1]<=38):
                            self.count_robo_stage_from_start = 2
                        self.main_controller("red", self.robo_store[0])
                    else:# When prediction is 6
                        forward_perm = 0# Controlling process to be forward or not
                        for coin in take_ref:# coin is sky_blue individual coin distance
                            if coin>-1 and coin<101:
                                if (coin != 40 or coin != 35 or coin != 27 or coin != 22 or coin != 14 or coin != 9 or
coin !=1 or coin !=48) and coin-temp[self.robo_store[0]-1] >= 6 and coin-temp[self.robo_store[0]-1]
<= 12:
                                    forward_perm = 1
                                    break
                                else:
                                    forward_perm = 0
                            else:
```

```
                    forward_perm = 0


                if forward_perm == 0:# Not forward the process
                    store = [1,2,3,4]
                    store.remove(self.robo_store[0])
                    predicted_coin = choice(store)
                    self.robo_store.append(predicted_coin)
                    self.main_controller("red", predicted_coin)
                else:# Forward the entire process
                    self.main_controller("red", self.robo_store[0])
            else:
                def normal_movement_according_condition():
                    # This portion is for checking if current location + predicted value <= 106 or not.....Coin
Filtering
                    normal_movement = 1# Normal Movement of the entite coin


                    for coin in self.robo_store:# coin is coin number
                        if temp[coin-1]+self.move_red_counter <= 106:# For all coins having predicted
location <=106
                            pass
                        else:
                            normal_movement = 0
                            break


                    if normal_movement:
                        temp_robo_store = [coin for coin in self.robo_store]
                    else:
                        temp_robo_store = [coin for coin in self.robo_store if temp[coin-
1]+self.move_red_counter <= 106]


                    # This portion is for coin filtering under some constrains
                    for coin in temp_robo_store:# coin is coin number
```

```
                if len(temp_robo_store)>1 and temp[coin-1]<101: # See Diagram under help to
unserstand to understand the location

                    if (temp[coin-1] in take_ref) and (temp[coin-1] != 1 or temp[coin-1] != 9 or
temp[coin-1] != 14 or temp[coin-1] != 22 or temp[coin-1] != 27 or temp[coin-1] != 35 or temp[coin-1]
!= 40 or temp[coin-1] != 48):

                        temp_robo_store.remove(coin)

                    elif temp[coin-1]<=39 and temp[coin-1]+self.move_red_counter>39:

                        for loc_coin_other in take_ref:

                            if (loc_coin_other>=40 and loc_coin_other<=46) and (temp[coin-
1]+self.move_red_counter>loc_coin_other):

                                temp_robo_store.remove(coin)

                                break


            # Overlapp checking with predicted value to eliminate other coin

            process_forward = 1

            for coin in temp_robo_store:

                if temp[coin-1]+self.move_red_counter in take_ref:

                    process_forward = 0

                    self.main_controller("red", coin)

                    break


            # Not a single overlapp found so now self rescue or safe forward

            if process_forward:

                take_len = len(temp_robo_store)

                store = {}

                if take_ref:

                    for robo in temp_robo_store:#  robo is coin number

                        for coin_other in take_ref:# coin_other is blue coin location

                            if coin_other>-1 and coin_other<100:

                                if take_len>1 and (temp[robo-1]>38 and coin_other<=38) or ((temp[robo-1]
== 9 or temp[robo-1] == 14 or temp[robo-1] == 27 or temp[robo-1] == 35 or temp[robo-1] == 40 or
temp[robo-1] == 48 or temp[robo-1] == 22) and (coin_other<=temp[robo-1] or
(coin_other>temp[robo-1] and coin_other<=temp[robo-1]+3))):  # avoid case to store
```

```
                    take_len-=1
                else:
                    store[temp[robo-1]-coin_other] = (robo, take_ref.index(coin_other)+1)# Store coin number

            # positive_distance = robo front        negative_distance = robo_behind
            if store:
                store_positive_dis = {}
                store_negative_dis = {}
                take_max = 0
                take_min = 0

                try:
                    store_positive_dis = dict((k,v) for k,v in store.items() if k>0)
                    take_min = min(store_positive_dis.items())
                except:
                    pass
                try:
                    store_negative_dis = dict((k,v) for k,v in store.items() if k<0)
                    take_max = max(store_negative_dis.items())
                except:
                    pass

                # Positive forward checking
                work_comp_in_pos = 0
                take_len = len(store_positive_dis)
                index_from_last = -1

                while take_len:
                    if take_min and take_min[0] <= 6:
```

```
                        work_comp_in_pos = 1
                        self.main_controller("red", take_min[1][0])
                        break
                    else:
                        index_from_last -= 1
                        try:
                            take_min = min(sorted(store_positive_dis.items()))[index_from_last])
                        except:
                            break
                    take_len -= 1



                # Negative forward checking
                work_comp_in_neg = 0
                if not work_comp_in_pos:
                    take_len = len(store_negative_dis)
                    index_from_last = len(store_negative_dis)-1
                    while take_len:
                        if take_max and temp[take_max[1][0]-1] + self.move_red_counter <=
take_ref[take_max[1][1]-1]:
                            work_comp_in_neg = 1
                            self.main_controller("red", take_max[1][0])
                            break
                        else:
                            index_from_last -= 1
                            try:
                                take_max = max(sorted(store_negative_dis.items()))[index_from_last])
                            except:
                                break
                        take_len -= 1
```

```
                    # Not operate in positive and negative distance method...So now cover it by closest distance to the destination
                    if not work_comp_in_neg and not work_comp_in_pos:
                        close_to_dest = temp_robo_store[0]
                        for coin_index in range(1,len(temp_robo_store)):
                            if temp[temp_robo_store[coin_index]-1] > temp[close_to_dest-1]:
                                close_to_dest = temp_robo_store[coin_index]

                        self.main_controller("red", close_to_dest)
                    else:# If store(Not find the location difference) is empty
                        close_to_dest = temp_robo_store[0]
                        for coin_index in range(1,len(temp_robo_store)):
                            if temp[temp_robo_store[coin_index]-1] > temp[close_to_dest-1]:
                                close_to_dest = temp_robo_store[coin_index]
                        self.main_controller("red", close_to_dest)
                else:
                    pass

            # For multiple Coin control Giving
            if self.move_red_counter<6:
                normal_movement_according_condition()
            else:
                coin_proceed = 0

                for coin in self.robo_store:
                    if temp[coin-1] + self.move_red_counter in self.blue_coin_position:
                        coin_proceed = coin
                        break

                if not coin_proceed:
                    if -1 in self.red_coin_position:
```

```
                    # Coin out
                    temp_store = [1,2,3,4]
                    for coin in self.robo_store:
                        temp_store.remove(coin)
                    take_pred = choice(temp_store)
                    self.robo_store.append(take_pred)
                    self.main_controller("red", take_pred)
                else:
                    # coin proceed
                    normal_movement_according_condition()
            else:
                self.main_controller("red", coin_proceed)
        else:
            self.make_prediction("red")# Prediction Function Call


if __name__ == '__main__':
    window = Tk()
    window.geometry("800x630")
    window.maxsize(800,630)
    window.minsize(800,630)
    window.title("LUDOMANIA BY Dhruvi Patel")
    window.iconbitmap("Images/ludo_icon.ico")
    # block_six_side = ImageTk.PhotoImage(Image.open("Images/6_block.png").resize((33, 33),
Image.ANTIALIAS))
    block_six_side = ImageTk.PhotoImage(Image.open("Images/6_block.png").resize((33, 33),
Image.LANCZOS))

    block_five_side = ImageTk.PhotoImage(Image.open("Images/5_block.png").resize((33, 33),
Image.LANCZOS))
    block_four_side = ImageTk.PhotoImage(Image.open("Images/4_block.png").resize((33, 33),
Image.LANCZOS))
```

```
    block_three_side = ImageTk.PhotoImage(Image.open("Images/3_block.png").resize((33, 33),
Image.LANCZOS))

    block_two_side = ImageTk.PhotoImage(Image.open("Images/2_block.png").resize((33, 33),
Image.LANCZOS))

    block_one_side = ImageTk.PhotoImage(Image.open("Images/1_block.png").resize((33, 33),
Image.LANCZOS))


Ludomania(window,block_six_side,block_five_side,block_four_side,block_three_side,block_two_sid
e,block_one_side)

    window.mainloop()
```

Hint.py

```
import tkinter as tk


class Ludomania:
    def __init__(self):
        self.window =("link unavailable")
        self.window.title("Ludo Game")
        self.hint_button = tk.Button(self.window, text="Hint", command=self.show_hint)
        self.hint_button.pack()

    def show_hint(self):
        # Generate possible moves
        possible_moves = self.generate_moves()

        # Evaluate and rank the moves
        ranked_moves = self.evaluate_moves(possible_moves)

        # Display the top two moves as suggestions
        hint_window = tk.Toplevel(self.window)
        hint_window.title("Hint")
        hint_label = tk.Label(hint_window, text="Possible moves:")
        hint_label.pack()
        for move in ranked_moves[:2]:
            move_label = tk.Label(hint_window, text=str(move))
            move_label.pack()

    def generate_moves(self):
        # Implement move generation logic here
        pass
```

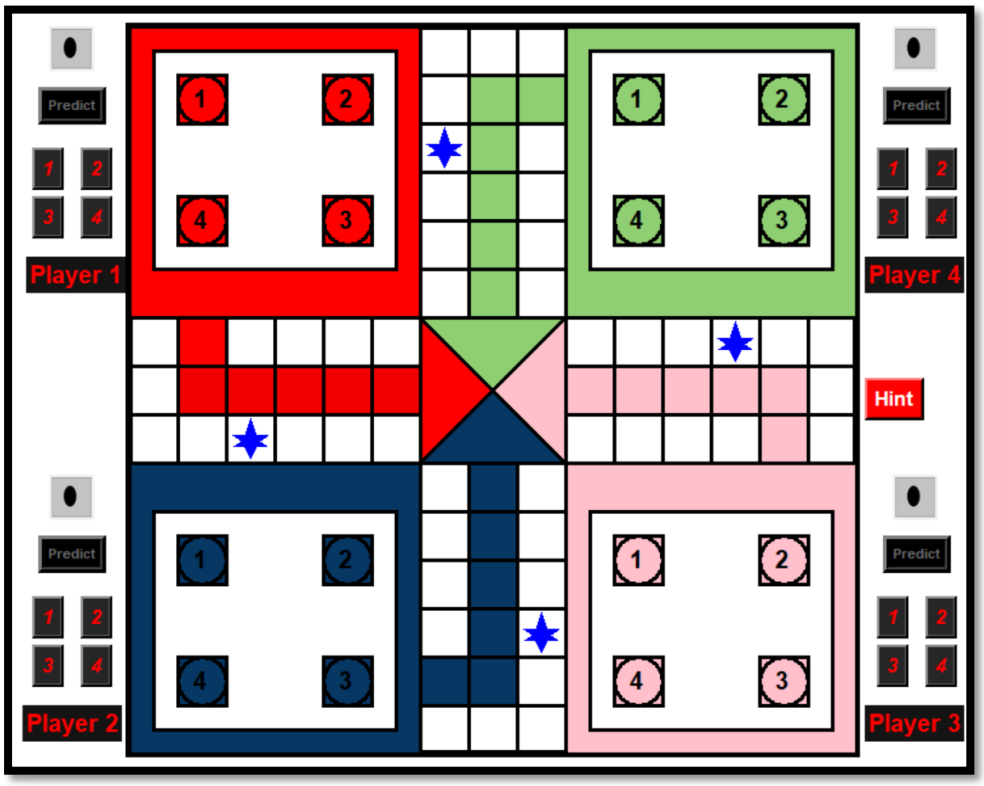
```
    def evaluate_moves(self, moves):
        # Implement move evaluation logic here
        pass

    def run(self):
        self.window.mainloop()

if __name__ == "__main__":
    game = Ludomania()
    game.run()
```

main page of the ludomania :

Game Board



Board Positions: