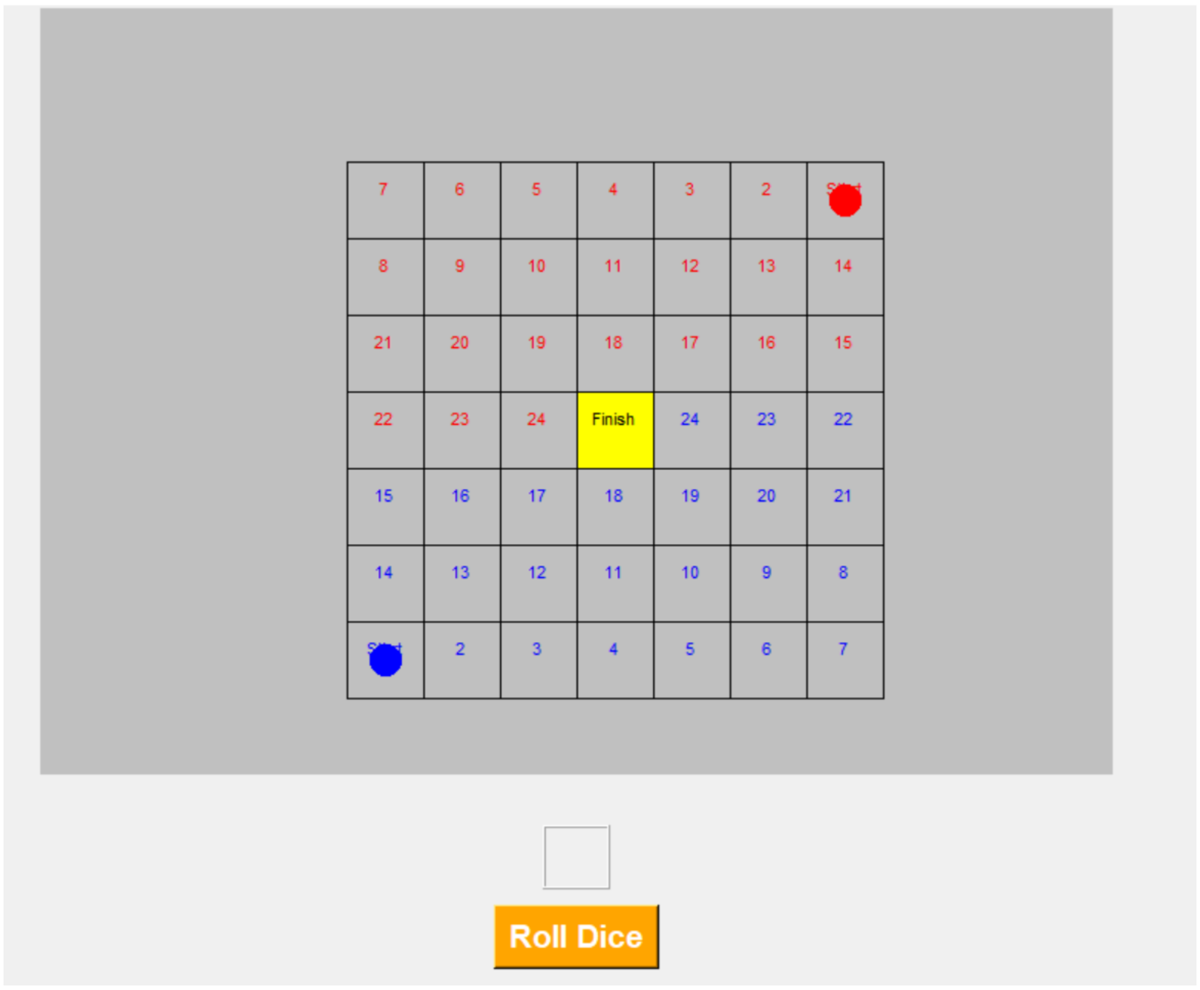
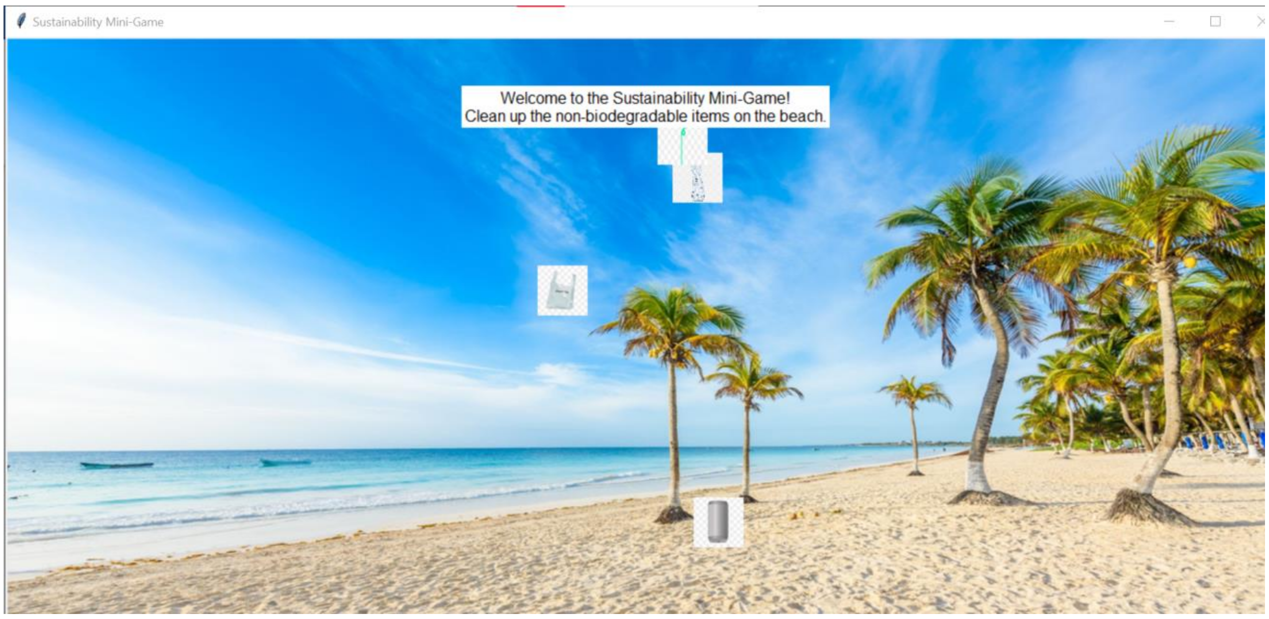
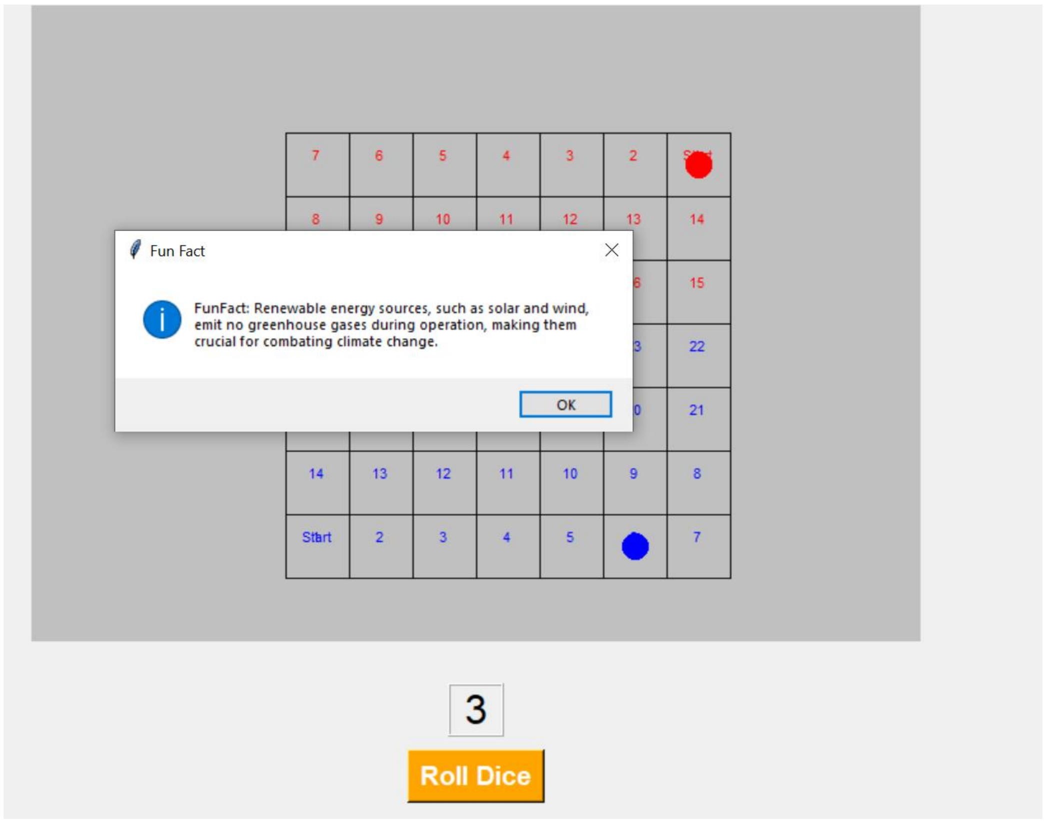


Figure 1 GUI design.



Welcome to the Sustainability Mini-Game!
Clean up the non-biodegradable items on the beach.

Figure 2 Sustainability mini game.



Fun Fact

FunFact: Renewable energy sources, such as solar and wind, emit no greenhouse gases during operation, making them crucial for combating climate change.

Figure 3 Fun fact about sustainability.

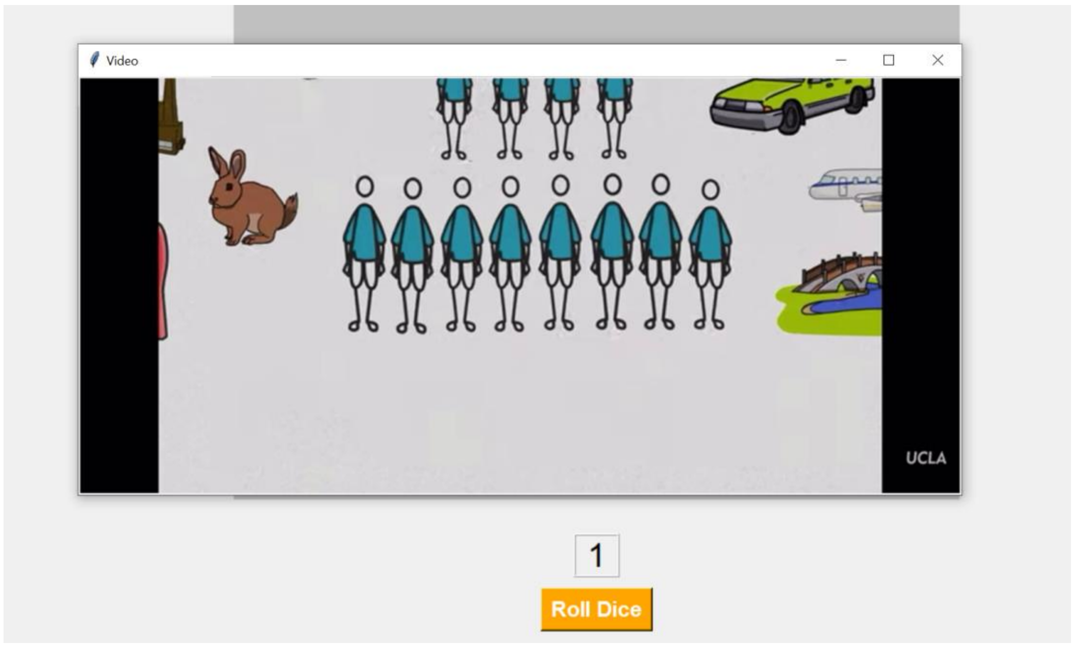
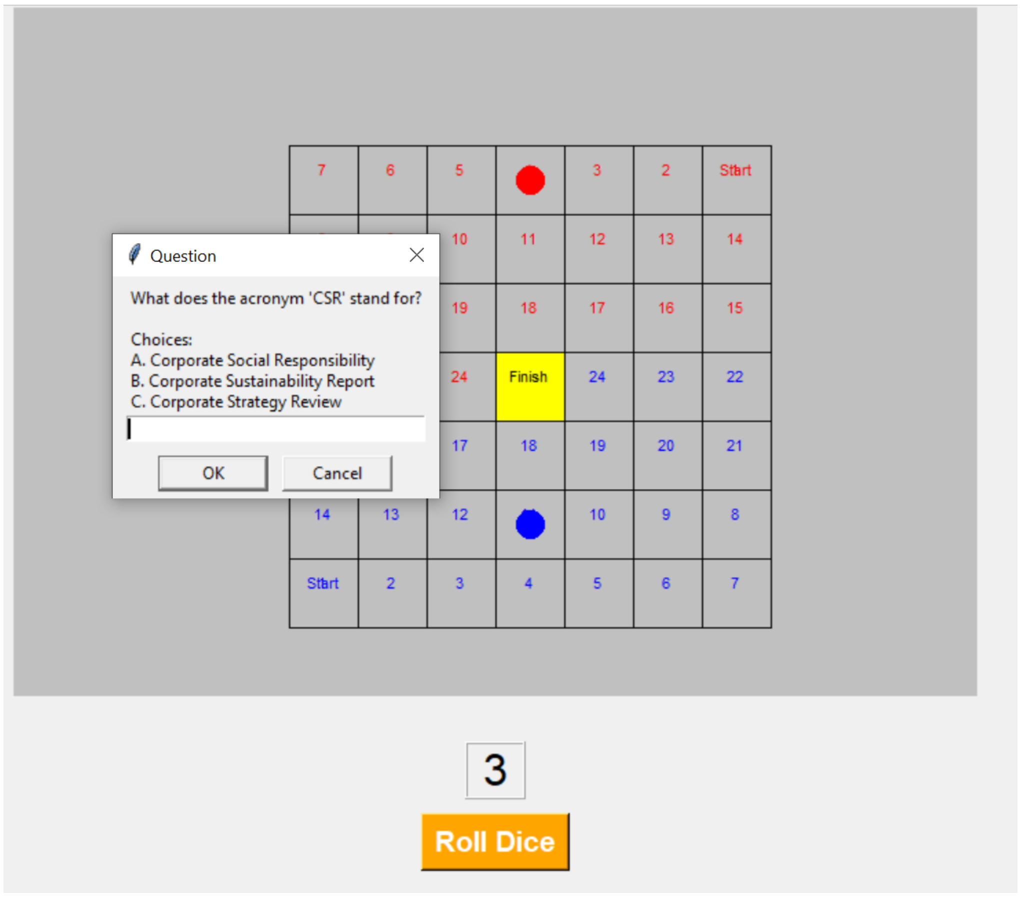


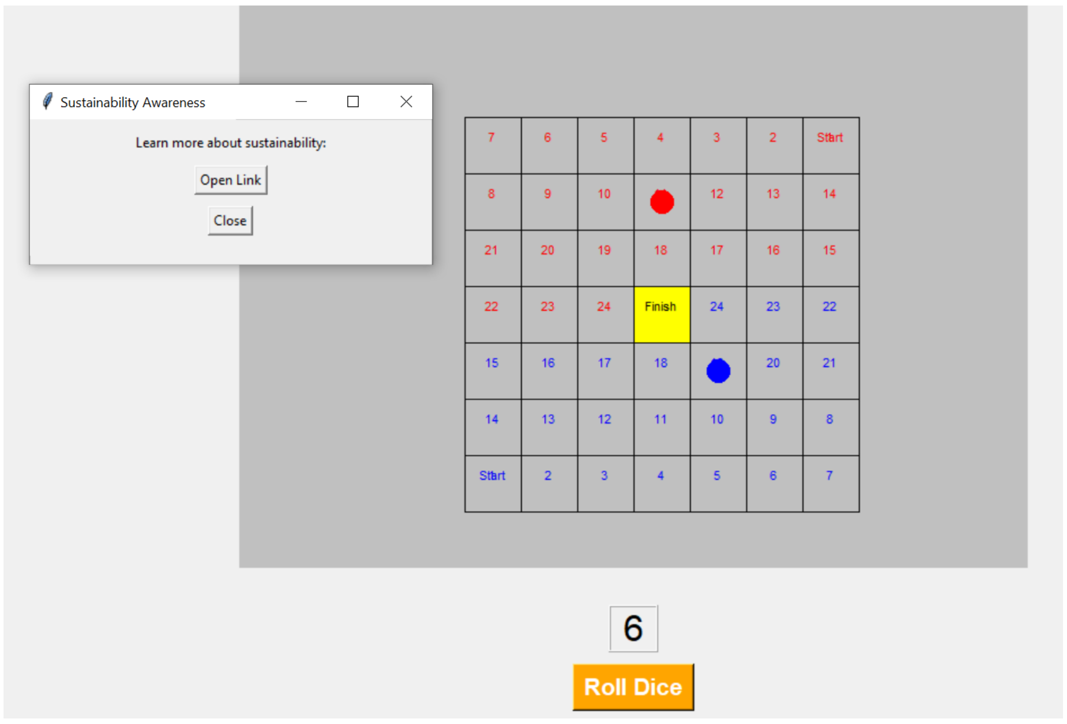
Figure 4 Video about sustainability.


Question

What does the acronym 'CSR' stand for?

Choices:
A. Corporate Social Responsibility
B. Corporate Sustainability Report
C. Corporate Strategy Review

Figure 5 Questions about sustainability.


Sustainability Awareness

Learn more about sustainability:

Figure 6 Links for sustainability awareness

```
import tkinter as tk
from tkinter import simpledialog, messagebox
import turtle
import random
import webbrowser
import time
import cv2
import threading
from tkinter import Toplevel, Label
from PIL import Image, ImageTk

# Initialize the main Tkinter window
root = tk.Tk()
  # Create a Turtle screen object
root.title("Board Game")
# Set window size to full screen
root.geometry("{0}x{1}+0+0".format(root.winfo_screenwidth(),
root.winfo_screenheight()))


# Setup Turtle Screen
canvas = tk.Canvas(master=root, width=700, height=500)
canvas.pack()
t_screen = turtle.TurtleScreen(canvas)
t_screen.bgcolor("silver")

# Define questions and answers for each coordinate
questions_answers = {
    (0, 0): ("What does the acronym 'CSR' stand for?", ["A. Corporate Social
Responsibility", "B. Corporate Sustainability Report", "C. Corporate Strategy
Review"], "A"),
    (0, 1): ("Which of the following is a renewable energy source?", ["A. Coal",
"B. Natural Gas", "C. Solar"], "C"),
    (0, 2): ("What is the primary greenhouse gas emitted through human
activities?", ["A. Carbon Dioxide", "B. Methane", "C. Nitrous Oxide"], "A"),
    (0, 3): ("What is the term for the sustainable use and management of Earth's
resources?", ["A. Conservation", "B. Exploitation", "C. Overconsumption"], "A"),
    (0, 4): ("Which of the following is a key component of a circular economy?",
["A. Waste Disposal", "B. Recycling", "C. Landfill"], "B"),
    (0, 5): ("What is the process of reducing waste by designing products that
can be reused or recycled?", ["A. Upcycling", "B. Downcycling", "C.
Incineration"], "A"),
    (0, 6): ("Which of the following is an example of sustainable
transportation?", ["A. Diesel Trucks", "B. Electric Cars", "C. Motorcycles"],
"B"),
    (1, 0): ("What is the main purpose of sustainable agriculture?", ["A.
Maximize Crop Yields", "B. Protect Biodiversity", "C. Increase Pesticide Use"],
"B"),
    (1, 1): ("What is the term for the practice of using resources in a way that
meets the needs of the present without compromising the ability of future
generations to meet their own needs?", ["A. Sustainability", "B. Conservation",
"C. Preservation"], "A"),
    (1, 2): ("Which of the following is a benefit of sustainable business
practices?", ["A. Increased Short-Term Profits", "B. Improved Brand Reputation",
"C. Environmental Degradation"], "B"),
}
# Define the boxes with video content and their corresponding URLs
video_coordinates = {
    # Red Player's Sustainability Videos
    (1, 3): r"C:\Users\Abc\OneDrive - Asia Pacific
University\Desktop\gdp\video1.mp4",
    (1, 4): r"C:\Users\Abc\OneDrive - Asia Pacific
University\Desktop\gdp\video1.mp4",
    (1, 5): r"C:\Users\Abc\OneDrive - Asia Pacific
University\Desktop\gdp\video1.mp4",
    (1, 6): r"C:\Users\Abc\OneDrive - Asia Pacific
University\Desktop\gdp\video1.mp4",
    (2, 0): r"C:\Users\Abc\OneDrive - Asia Pacific
University\Desktop\gdp\video1.mp4",
    (2, 1): r"C:\Users\Abc\OneDrive - Asia Pacific
University\Desktop\gdp\video1.mp4",
    (2, 2): r"C:\Users\Abc\OneDrive - Asia Pacific
University\Desktop\gdp\video1.mp4",
    (2, 3): r"C:\Users\Abc\OneDrive - Asia Pacific
University\Desktop\gdp\video1.mp4",
    (2, 4): r"C:\Users\Abc\OneDrive - Asia Pacific
University\Desktop\gdp\video1.mp4",
    (2, 5): r"C:\Users\Abc\OneDrive - Asia Pacific
University\Desktop\gdp\video1.mp4",
}
sustainability_facts = {
    (2, 6): "FunFact:Efficient water usage practices in agriculture can reduce
water waste and improve food security worldwide. ",
    (3, 0): "FunFact: Renewable energy sources, such as solar and wind, emit no
greenhouse gases during operation, making them crucial for combating climate
change.",
    (3, 1): "FunFact:Reducing, reusing, and recycling materials can
significantly cut down waste, lessen environmental impact, and conserve
resources.",
    (3, 2): "FunFact:Green building designs and materials reduce energy
consumption and increase the sustainability of construction projects.",
    (3, 3): "FunFact :Sustainable transport systems, including cycling, walking,
and electric vehicles, can greatly reduce carbon emissions and urban
pollution.",
    (3, 4): "FunFact:Conserving natural habitats and ecosystems is essential for
maintaining biodiversity and supporting life on Earth.",
    (3, 5): "FunFact: Sustainable agriculture practices, such as crop rotation
and organic farming, enhance soil health and reduce the need for chemical
fertilizers",
    (3, 6): "FunFact: Investing in energy efficiency not only reduces emissions
but also lowers energy costs for businesses and households.",
    (4, 0): "FunFact:Plastic pollution in oceans can be combated with improved
waste management systems and increased recycling efforts.",
    (4, 1): "FunFact:Public awareness and education on sustainability are key to
driving change and encouraging environmentally friendly behaviors.",
}

sustainability_links = {
    (4, 2): "https://www.irena.org/",
    (4, 3): "https://www.fao.org/sustainability/en/",
    (4, 4): "https://www.watercalculator.org/",
    (4, 5): "https://www.usgbc.org/leed",
    (4, 6): "https://www.itdp.org/",
    (5, 0): "https://www.epa.gov/recycle",
    (5, 1): "https://www.conservation.org/",
    (5, 2): "https://www.iea.org/topics/energy-efficiency",
    (5, 3): "https://www.plasticoceans.org/",
    (5, 4): "https://www.earthday.org/",
}

# Sustainability Links Function
def open_sustainability_link(url):
    link_window = tk.Toplevel(root)
    link_window.title("Sustainability Awareness")

    tk.Label(link_window, text="Learn more about
sustainability:").pack(pady=(10, 0))
```

```
    def callback(url=url):
        webbrowser.open_new_tab(url)

    link_button = tk.Button(link_window, text="Open Link", command=lambda:
callback(url))
    link_button.pack(pady=10)

    tk.Button(link_window, text="Close",
command=link_window.destroy).pack(pady=(0, 10))


def play_video(video_path, callback=None):
    def close_video_window():
        video_window.destroy()
        if callback:
            callback()

    def _play():
        cap = cv2.VideoCapture(video_path)
        if not cap.isOpened():
            print("Error: Could not open video.")
            return

        nonlocal video_window  # Use nonlocal keyword to access video_window
from the outer scope

        # Create a new Toplevel window for video playback
        video_window = tk.Toplevel(root)
        video_window.title("Video")

        # Position the video window within the boundaries of the main window
        root_x = root.winfo_x()
        root_y = root.winfo_y()
        root_width = root.winfo_width()
        root_height = root.winfo_height()
        video_window_x = root_x + (root_width - video_window.winfo_reqwidth()) /
2
        video_window_y = root_y + (root_height - video_window.winfo_reqheight())
/ 2
        video_window.geometry("+%d+%d" % (video_window_x, video_window_y))

        # Create a label widget to display video frames
        label = tk.Label(video_window)
        label.pack()

        while True:
            ret, frame = cap.read()
            if not ret:
                break

            frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            img = ImageTk.PhotoImage(image=Image.fromarray(frame))
            label.configure(image=img)
            label.image = img
            video_window.update_idletasks()
            video_window.update()

        cap.release()

        # Create a close button for the video window
        close_button = tk.Button(video_window, text="Close",
command=close_video_window)
        close_button.pack()

        # Destroy the video window after playback completes
        close_video_window()

    # Initialize video_window variable
    video_window = None

    # Start the video playback in a new thread to avoid freezing the GUI
    threading.Thread(target=_play).start()


# Define a label widget where you want to display the video frames
video_label = tk.Label(root)
video_label.pack()

# Define a label widget where you want to display the video frames


# Define the function for the sustainability mini-game
litter_images = []  # This list will hold the PhotoImage objects to prevent them
from being garbage collected


# At the top level of your script, initialize litter_images
def sustainability_mini_game():
    global litter_images

    mini_game_window = tk.Toplevel(root)
    mini_game_window.title("Sustainability Mini-Game")

mini_game_window.geometry("{0}x{1}+0+0".format(mini_game_window.winfo_screenwidt
h(), mini_game_window.winfo_screenheight()))

    beach_canvas = tk.Canvas(mini_game_window,
width=mini_game_window.winfo_screenwidth(),
height=mini_game_window.winfo_screenheight())
    beach_canvas.pack()

    # Load and display background image
    background_image_path = r"C:\Users\Abc\OneDrive - Asia Pacific
University\Desktop\gdp\beach.jpeg"  # Update this path
    background_image = Image.open(background_image_path)
    resized_background_image =
background_image.resize((mini_game_window.winfo_screenwidth(),
mini_game_window.winfo_screenheight()), Image.LANCZOS)
    beach_image = ImageTk.PhotoImage(resized_background_image)
    beach_canvas.create_image(0, 0, anchor=tk.NW, image=beach_image)
    beach_canvas.image = beach_image  # Keep the reference

    # Display instructions or objective
    instructions_text = "Welcome to the Sustainability Mini-Game!\nClean up the
non-biodegradable items on the beach."
    instructions_label = tk.Label(mini_game_window, text=instructions_text,
font=("Arial", 12), bg="white")
    instructions_label.place(relx=0.5, rely=0.1, anchor=tk.CENTER)

    # Track active litter items
    active_litter_items = []

    # Load litter items
    litter_paths = [
        r"C:\Users\Abc\OneDrive - Asia Pacific
University\Desktop\gdp\plastic.png",  # Update paths
        r"C:\Users\Abc\OneDrive - Asia Pacific University\Desktop\gdp\can.png",
        r"C:\Users\Abc\OneDrive - Asia Pacific
University\Desktop\gdp\straw.png",
        r"C:\Users\Abc\OneDrive - Asia Pacific
University\Desktop\gdp\plasticbag.png"
```

```
    ]

    for item_path in litter_paths:
        icon = Image.open(item_path)
        icon = icon.resize((50, 50), Image.LANCZOS)
        photo_image = ImageTk.PhotoImage(icon)
        litter_images.append(photo_image)  # Keep references to prevent garbage
collection
        x = random.randint(50, mini_game_window.winfo_screenwidth() - 50)
        y = random.randint(50, mini_game_window.winfo_screenheight() - 50)
        item_id = beach_canvas.create_image(x, y, image=photo_image)
        active_litter_items.append(item_id)  # Track item IDs

        # Function to handle litter pickup
        def remove_litter(event, item_id=item_id):
            beach_canvas.delete(item_id)
            active_litter_items.remove(item_id)
            if not active_litter_items:
                messagebox.showinfo("Congratulations!", "You've cleaned up all
the litter!")
                mini_game_window.destroy()
                # Move the player after the mini-game is completed
                roll_dice()  # This will simulate rolling the dice and moving
the player

        beach_canvas.tag_bind(item_id, "<ButtonPress-1>", remove_litter)  # Bind
the remove_litter function to the litter item


# Define coordinates for sustainability mini-games
sustainability_mini_game_coordinates = {
    (5, 0): sustainability_mini_game,
    (5, 1): sustainability_mini_game,
    (5, 2): sustainability_mini_game,
    (5, 3): sustainability_mini_game,
    (5, 4): sustainability_mini_game,
    (5, 5): sustainability_mini_game,
    (5, 6): sustainability_mini_game,
    (6, 0): sustainability_mini_game,
    (6, 1): sustainability_mini_game,
    (6, 2): sustainability_mini_game,
    (6, 3): sustainability_mini_game,
    (6, 4): sustainability_mini_game,
    (6, 5): sustainability_mini_game,
    (4, 2): sustainability_mini_game,
    (4, 5): sustainability_mini_game,
}


move_sequence_blue = [
    (0, 6), (1, 6), (2, 6), (3, 6), (4, 6), (5, 6), (6, 6), (6, 5), (5, 5), (4,
5), (3, 5), (2, 5), (1, 5), (0, 5),
    (0, 4), (1, 4), (2, 4), (3, 4), (4, 4), (5, 4), (6, 4), (6, 3), (5, 3), (4,
3), (3, 3)
]

move_sequence_red = [
    (6, 0), (5, 0), (4, 0), (3, 0), (2, 0), (1, 0), (0, 0), (0, 1), (1, 1), (2,
1), (3, 1), (4, 1), (5, 1),
    (6, 1), (6, 2), (5, 2), (4, 2), (3, 2), (2, 2), (1, 2), (0, 2), (0, 3), (1,
3), (2, 3), (3, 3)
]

# Modify the Player class to include sustainability mini-games
class Player(turtle.RawTurtle):
    def __init__(self, shape, color, start_row_col, move_sequence,
video_coordinates, canvas):
        super().__init__(canvas=canvas, shape=shape)
        self.screen.bgcolor("silver")
        self.color(color)
        self.penup()
        self.goto(self.calculate_position(start_row_col))
        self.start_row_col = start_row_col
        self.current_row_col = start_row_col
        self.move_sequence = move_sequence
        self.video_coordinates = video_coordinates

    def move(self, steps):
        # Get the index of the current position in the player's move sequence
        current_index = self.move_sequence.index(self.current_row_col)

        # Calculate the index of the next position based on the steps to move
        next_index = current_index + steps

        # Calculate the maximum index allowed in the move sequence
        max_index = len(self.move_sequence) - 1

        # Check if the next index exceeds the maximum index, indicating reaching
the end
        if next_index > max_index:
            messagebox.showinfo("Cannot Move", "Roll exceeds steps needed to
reach the end. Wait for next turn.")
            return  # Early return to prevent moving beyond the end

        # Retrieve the coordinates of the next position in the move sequence
        next_position = self.move_sequence[next_index]

        # Check different types of positions the player may land on and take
appropriate actions
        if next_position in questions_answers:
            # If the next position corresponds to a question, display the
question and await the user's answer
            question, choices, correct_answer = questions_answers[next_position]
            user_answer = simpledialog.askstring(
                "Question",
                question + "\n\nChoices:\n" + "\n".join(choices),
                parent=root
            )

            if user_answer and user_answer.upper() in ['A', 'B', 'C']:
                if user_answer.upper() == correct_answer:
                    messagebox.showinfo("Correct", "You answered correctly!
Moving forward.")
                    self.goto(self.calculate_position(next_position))
                    self.current_row_col = next_position

                    # Check if the player has reached the finish line
                    if next_position == (3, 3):
                        messagebox.showinfo("Game Over", f"Congratulations!
{self.color()} Player has reached the end.")
                        root.quit()  # Terminate the program when a player
reaches the finish line
                else:
                    correct_choice = choices[ord(correct_answer) - ord('A')]
                    messagebox.showinfo(
                        "Incorrect",
                        f"Sorry, '{user_answer}' is incorrect. The correct
answer is '{correct_choice}'. "
                        "You stay at the same position."
                    )
            else:
                messagebox.showwarning("Invalid Answer", "Please enter a valid
answer (A, B, or C).")
                return

        elif next_position in sustainability_mini_game_coordinates:
            # If the next position corresponds to a sustainability mini-game,
run the mini-game function
            mini_game_function =
sustainability_mini_game_coordinates[next_position]
            mini_game_function()  # Call the sustainability mini-game function
            # Move the player automatically to the next position after
completing the mini-game
            self.goto(self.calculate_position(next_position))
            self.current_row_col = next_position

        elif next_position in sustainability_facts:
            # If the next position corresponds to a fun fact box, display the
fun fact
            messagebox.showinfo("Fun Fact", sustainability_facts[next_position])
            # Move the player automatically to the next position
            self.goto(self.calculate_position(next_position))
            self.current_row_col = next_position

        elif next_position in sustainability_links:
            # If the next position corresponds to a sustainability link, open
the link in a new window
            open_sustainability_link(sustainability_links[next_position])
            # Move the player automatically to the next position
            self.goto(self.calculate_position(next_position))
            self.current_row_col = next_position

            # Check if the player has reached the finish line
            if next_position == (3, 3):
                messagebox.showinfo("Game Over", f"Congratulations!
{self.color()} Player has reached the end.")
                root.quit()  # Terminate the program when a player reaches the
finish line

        elif next_position in self.video_coordinates:
            # If the next position corresponds to a video coordinate, play the
associated video
            # Define what should happen after the video ends
            def after_video():
                # Move the player automatically to the next position after
completing the video
                self.goto(self.calculate_position(next_position))
                self.current_row_col = next_position
                # Add any additional logic here to continue the game after video
playback

            # Play the video and pass the `after_video` function as a callback
            play_video(video_coordinates[next_position], callback=after_video)

        else:
            # If the next position is a regular position, move the player to
that position
            self.goto(self.calculate_position(next_position))
            self.current_row_col = next_position

            # Check if the player has reached the finish line
        if next_position == (3, 3):
                messagebox.showinfo("Game Over", f"Congratulations!
{self.color()} Player has reached the end.")
                root.quit()  # Terminate the program when a player reaches the
finish line

    def calculate_position(self, row_col):
        """Calculate the pixel position centering in the box based on row and
column."""
        x = -150 + row_col[0] * 50 + 25  # Center in the box horizontally
        y = 150 - row_col[1] * 50 - 25  # Center in the box vertically
        return (x, y)


def draw_box(turtle_obj, x, y, special=False, text=""):
    turtle_obj.penup()
    start_x = -150 + x * 50
    start_y = 150 - y * 50
    turtle_obj.goto(start_x, start_y)
    turtle_obj.pendown()
    if special:
        turtle_obj.fillcolor("yellow")
        turtle_obj.begin_fill()
    for _ in range(4):
        turtle_obj.forward(50)
        turtle_obj.right(90)
    if special:
        turtle_obj.end_fill()
        turtle_obj.penup()
        turtle_obj.goto(start_x + 25, start_y - 25)
        turtle_obj.write("Finish", align="center", font=("Arial", 8, "normal"))
    elif text:
        turtle_obj.penup()
        turtle_obj.goto(start_x + 25, start_y - 25)
        turtle_obj.write(text, align="center", font=("Arial", 8, "normal"))


def draw_number(turtle_obj, x, y, number, color):
    turtle_obj.penup()
    start_x = -150 + x * 50 + 25
    start_y = 150 - y * 50 - 25
    turtle_obj.goto(start_x, start_y)
    turtle_obj.pendown()
    turtle_obj.color(color)
    turtle_obj.write(str(number), align="center", font=("Arial", 8, "normal"))


def draw_start_text(turtle_obj, x, y, color):
    turtle_obj.penup()
    start_x = -150 + x * 50 + 25
    start_y = 150 - y * 50 - 25
    turtle_obj.goto(start_x, start_y)
    turtle_obj.pendown()
    turtle_obj.color(color)
    turtle_obj.write("Start", align="center", font=("Arial", 8, "normal"))


# Function to draw the game board with numbers for player paths
def draw_board():
    board_turtle = turtle.RawTurtle(t_screen)
    board_turtle.speed(0)
    board_turtle.hideturtle()
    t_screen.tracer(0)  # Disable screen updates while drawing

    for y in range(7):
        for x in range(7):
            if (x, y) == (3, 3):  # Check if the box is at position (3, 3)
                draw_box(board_turtle, x, y, special=True)  # Pass special=True
for the special box
            else:
                draw_box(board_turtle, x, y)

    # Display numbers for blue player path
```

```python
    for i, position in enumerate(move_sequence_blue, start=1):
        if i != 25:
            x, y = position
            draw_number(board_turtle, x, y, i, "blue")

    # Display numbers for red player path
    for i, position in enumerate(move_sequence_red, start=1):
        if i != 25:
            x, y = position
            draw_number(board_turtle, x, y, i, "red")

    # Display "start" at position (0, 6)
    draw_start_text(board_turtle, 0, 6, "blue")

    # Display "start" at position (6, 0)
    draw_start_text(board_turtle, 6, 0, "red")

    t_screen.update()  # Update the screen after drawing
    t_screen.tracer(1)  # Enable screen updates


# Create blue player
# Assuming you have defined the necessary variables and the game board
# Initialize players
blue_player = Player("circle", "blue", (0, 6), move_sequence_blue,
video_coordinates, t_screen.getcanvas())
red_player = Player("circle", "red", (6, 0), move_sequence_red,
video_coordinates, t_screen.getcanvas())


# Set the current player
current_player = blue_player


# Function to roll the dice and move the player
def roll_dice(roll=None):
    global current_player

    if roll is None:
        # If roll is None, it means we need to prompt the user to roll the dice
        dice_roll_button.config(state="normal")  # Enable the dice roll button
        return

    messagebox.showinfo("Dice Roll", f"You rolled: {roll}")
    current_player.move(roll)

    # Switch to the next player's turn
    if current_player == blue_player:
        current_player = red_player
    else:
        current_player = blue_player


# Function to animate the rolling of the die and alternate player turns
def animate_die():
    # Roll the die
    roll = random.randint(1, 6)
    die_label.config(text=str(roll))
    root.update()
    time.sleep(0.2)  # Add a short delay for animation effect
    roll_dice(roll)  # After animation, call the function to move the player


# Create and position the animated die widget
die_label = tk.Label(root, text="", font=("Helvetica", 24), width=2,
relief="ridge")
```

```python
die_label.pack(pady=10)

# Create and position the dice roll button
dice_roll_button = tk.Button(
    root,
    text="Roll Dice",
    command=animate_die,
    bg="orange",
    fg="white",
    font=("Helvetica", 16, "bold"),
    relief="raised"
)
dice_roll_button.pack()

# Draw the game board
draw_board()

# Start the Tkinter event loop
tk.mainloop()
```