main_tests

September 3, 2024

[ ]:
```
import pygame
import sys
import os
import numpy as np
from time import strftime

# Define colors here
BLACK = (0, 0, 0)
WHITE = (255, 255, 255)
RED = (255, 0, 0)
LIGHT_RED = (255, 100, 100)

# Initialize Pygame
pygame.init()

# Set up the window
os.environ["SDL_VIDEO_CENTERED"] = "1"
clock = pygame.time.Clock()
padding = 0
surface = pygame.display.set_mode(display=1)
displayX, displayY = surface.get_size()
windowX, windowY = displayX - padding, displayY - padding
screen = pygame.display.set_mode((windowX, windowY), pygame.RESIZABLE,␣
 ↪display=1)
pygame.display.set_caption("Resizable Window")

# Set up fonts
font = pygame.font.Font(None, 36)

# Variables
total_score = 0

# Define the Ball class
class Ball:
    def __init__(self, x, y, dx, dy, radius, color, clicked_color):
        self.x = x
        self.y = y
```

```
        self.dx = dx
        self.dy = dy
        self.radius = radius
        self.color = color
        self.clicked_color = clicked_color
        self.default_color = color
        self.clicked = False
        self.reinforced = False

    def draw(self, screen):
        pygame.draw.circle(screen, self.color, (int(self.x), int(self.y)), self.
 ↪radius)

    def move(self):
        self.x += self.dx
        self.y += self.dy

        if self.x - self.radius < 0 or self.x + self.radius > windowX:
            self.dx = -self.dx
        if self.y - self.radius < 0 or self.y + self.radius > windowY:
            self.dy = -self.dy

    def check_click(self, pos):
        distance = np.sqrt((self.x - pos[0])**2 + (self.y - pos[1])**2)
        if distance < self.radius:
            self.clicked = True
            return True
        return False

    def darken_color(self):
        self.color = self.clicked_color

    def reset_color(self):
        self.color = self.default_color

# Define the Simulation class
class Simulation:
    def __init__(self):
        self.balls = []
        self.reinforcement_texts = []
        self.create_balls()

    def create_balls(self):
        # Example ball creation, customize as needed
        self.balls.append(Ball(100, 100, 2, 2, 30, RED, LIGHT_RED))
        self.balls.append(Ball(200, 200, -3, -3, 40, RED, LIGHT_RED))
```

```
    def handle_click(self, pos):
        for ball in self.balls:
            if ball.check_click(pos):
                ball.darken_color()
                ball.reinforced = True
                self.add_reinforcement_text(ball.x, ball.y)

    def add_reinforcement_text(self, x, y):
        self.reinforcement_texts.append({
            'text': '+1',
            'x': x,
            'y': y,
            'font_size': 36,
            'alpha': 255
        })

    def update_reinforcement_texts(self):
        for text in self.reinforcement_texts[:]:
            text['y'] -= 1
            text['font_size'] += 1
            text['alpha'] -= 5
            if text['alpha'] <= 0:
                self.reinforcement_texts.remove(text)

    def draw_reinforcement_texts(self, screen):
        for text in self.reinforcement_texts:
            font = pygame.font.Font(None, text['font_size'])
            rendered_text = font.render(text['text'], True, WHITE)
            rendered_text.set_alpha(text['alpha'])
            screen.blit(rendered_text, (text['x'], text['y']))

    def move_balls(self):
        for ball in self.balls:
            ball.move()

    def draw_balls(self, screen):
        for ball in self.balls:
            ball.draw(screen)

    def reset_ball_colors(self):
        for ball in self.balls:
            ball.reset_color()

# Main game loop
def main():
    sim = Simulation()
    running = True
```

```
    while running:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False
            elif event.type == pygame.MOUSEBUTTONDOWN:
                sim.handle_click(event.pos)

        sim.move_balls()
        sim.update_reinforcement_texts()

        screen.fill(BLACK)
        sim.draw_balls(screen)
        sim.draw_reinforcement_texts(screen)
        pygame.display.flip()

        clock.tick(60)

    pygame.quit()
    sys.exit()

if __name__ == "__main__":
    main()
```

```
pygame 2.6.0 (SDL 2.28.4, Python 3.12.5)
Hello from the pygame community. https://www.pygame.org/contribute.html
```

```
An exception has occurred, use %tb to see the full traceback.

SystemExit
```

```
C:\Users\Admin\AppData\Roaming\Python\Python312\site-
packages\IPython\core\interactiveshell.py:3585: UserWarning: To exit: use
'exit', 'quit', or Ctrl-D.
  warn("To exit: use 'exit', 'quit', or Ctrl-D.", stacklevel=1)
```

[ ]:
```
# %%
import logtocsv
# logtocsv.write_data(string)
# NOTE: Change over delay can be to or from given ball
import pygame
import sys
import os
import numpy as np
from time import strftime  # see format codes: https://docs.python.org/3/
 ↪library/datetime.html#format-codes
```

```
## Define colors here
BLACK = (0, 0, 0)
# LIGHT_BLACK = tuple(min(x + y, 255) for x, y in zip(BLACK, (50, 50, 50)))
# print(LIGHT_BLACK)
RED = (255, 0, 0)
# LIGHT_RED = tuple(min(x + y, 255) for x, y in zip(RED, (50, 50, 50)))
# print(LIGHT_RED)
DARK_RED = (139, 0, 0)
ORANGE = (255, 165, 0)
DARK_ORANGE = (255, 140, 0)
YELLOW = (255, 255, 0)
DARK_YELLOW = (185, 185, 0)
GREEN = (0, 128, 0)
DARK_GREEN = (0, 100, 0)
BLUE = (0, 0, 255)
DARK_BLUE = (0, 0, 139)
INDIGO = (75, 0, 130)
DARK_INDIGO = (54, 0, 94)
VIOLET = (128, 0, 128)
DARK_VIOLET = (80, 0, 80)
SQUARE_COLOR = (255, 255, 255)
SQUARE_THICKNESS = 4

## Define phases here
## Add global blockers based on switching the clicked stimuli
score_clicks_required = 0
last_reinforced_ball = None
last_reinforced_time = None
reinforcement_blocked_until_time = None

phase_options = {
    "phase_1": {
        "duration" : 300,
        "number_balls": 3,
        "initial_speed": [1,1,1,1,1,1,1],
        "radii": [60,60,60,60,60,60,60],
        "base_colors" : [RED, ORANGE, YELLOW, GREEN, BLUE, INDIGO, VIOLET],
        "clicked_colors" : [DARK_RED, DARK_ORANGE, DARK_YELLOW, DARK_GREEN,␣
 ↪DARK_BLUE, DARK_INDIGO, DARK_VIOLET],
        #"reinforced_colors":,
        #"discriminative_stimulus_colors":,
        "time_required" :[0.1,0.1,0.1,0.1,0.1,0.1,0.1],
        "clicks_required" :[1,1,1,1,1,1,1],
        "change_to_clicks" : [1,1,1,1,1,1,1],
        "change_to_delay" : [5,2.5,1,1,1,1,1],
        "change_from_clicks" : [1,1,1,1,1,1,1],
```

```
        "change_from_delay": [1,1,1,1,1,1,1],
        "block_score_until_time":[0,0,0,0,0,0,0],
        "block_score_until_clicks" : [0,0,0,0,0,0,0],
        "yoked" : False,
        "debug" : True
    },
    # "phase_2": {
    #     "duration" : 4,
    #     "number_balls": 3,
    #     "initial_speed": [1,1,1,1,1,1,1],
    #     "radii": [60,60,60,60,60,60,60],
    #     "base_colors" : [RED, ORANGE, YELLOW, GREEN, BLUE, INDIGO, VIOLET],
    #     "clicked_colors" : [DARK_RED, DARK_ORANGE, DARK_YELLOW, DARK_GREEN,␣
 ↪DARK_BLUE, DARK_INDIGO, DARK_VIOLET],
    #     "time_required" :[0.1,0.1,0.1,0.1,0.1,0.1,0.1],
    #     "clicks_required" :[1,1,1,1,1,1,1],
    #     "change_to_clicks" : [1,1,1,1,1,1,1],
    #     "change_to_delay" : [1,1,1,1,1,1,1],
    #     "change_from_clicks" : [1,1,1,1,1,1,1],
    #     "change_from_delay": [1,1,1,1,1,1,1],
    #     "block_score_until_time":[0,0,0,0,0,0,0],
    #     "block_score_until_clicks" : [0,0,0,0,0,0,0],
    #     "yoked" : False,
    #     "debug" : True
    # },
    # "phase_3":  {
    #     "duration" : 5,
    #     "number_balls": 3,
    #     "initial_speed": [1,1,1,1,1,1,1],
    #     "radii": [60,60,60,60,60,60,60],
    #     "base_colors" : [RED, ORANGE, YELLOW, GREEN, BLUE, INDIGO, VIOLET],
    #     "clicked_colors" : [DARK_RED, DARK_ORANGE, DARK_YELLOW, DARK_GREEN,␣
 ↪DARK_BLUE, DARK_INDIGO, DARK_VIOLET],
    #     "time_required" :[0.1,0.1,0.1,0.1,0.1,0.1,0.1],
    #     "clicks_required" :[1,1,1,1,1,1,1],
    #     "change_to_clicks" : [1,1,1,1,1,1,1],
    #     "change_to_delay" : [1,1,1,1,1,1,1],
    #     "change_from_clicks" : [1,1,1,1,1,1,1],
    #     "change_from_delay": [1,1,1,1,1,1,1],
    #     "block_score_until_time":[0,0,0,0,0,0,0],
    #     "block_score_until_clicks" : [0,0,0,0,0,0,0],
    #     "yoked" : False,
    #     "debug" : True
    # },
}

# Initialize Pygame
```

```
pygame.init()

SCHEDULED_EVENT = pygame.USEREVENT + 1


# pygame.font.init()
font = pygame.font.Font(None, 36)  # Choose a font and size

experimentdate = strftime('%a %d %b %Y, %I:%M%p')
logtocsv.write_data(experimentdate)

# Set up the window
os.environ["SDL_VIDEO_CENTERED"] = "1"
clock = pygame.time.Clock()
padding = 0
surface = pygame.display.set_mode(display=1)
displayX, displayY = surface.get_size()
windowX, windowY = displayX - padding, displayY - padding # Here I was␣
 ↪subtracging padding
screen = pygame.display.set_mode((windowX, windowY), pygame.
 ↪RESIZABLE,display=1)  #screen = pygame.display.set_mode((windowX, windowY),␣
 ↪pygame.RESIZABLE,display=1)
pygame.display.set_caption("Resizable Window")

# Set up the square
square_color = (255, 0, 0)
min_margin = 20
square_size = min(windowX, windowY) - 2 * min_margin
square_rect = pygame.Rect((windowX - square_size) // 2, (windowY - square_size)␣
 ↪// 2, square_size, square_size)

margin = 100
margin_left = margin
margin_right = margin
margin_top = margin
margin_bottom = margin
values = None

bounce_box_left = margin_left
bounce_box_right = windowX - margin_right
bounce_box_top = windowY - margin_top
square_rect = pygame.Rect((windowX - square_size) // 2, (windowY - square_size)␣
 ↪// 2, square_size, square_size)
bounce_box_bottom = margin_bottom

#Random variables for right here:
total_score = 0
```

```
current_phase = 1
event = None
current_seconds = 0
#counters
clicked_on_ball = False

## This portion is key for our "Reverse lookup" dictionary
color_names = {
    "BLACK": (0, 0, 0),
    "RED": (255, 0, 0),
    "DARK_RED": (139, 0, 0),
    "ORANGE": (255, 165, 0),
    "DARK_ORANGE": (255, 140, 0),
    "YELLOW": (255, 255, 0),
    "DARK_YELLOW": (185, 185, 0),
    "GREEN": (0, 128, 0),
    "DARK_GREEN": (0, 100, 0),
    "BLUE": (0, 0, 255),
    "DARK_BLUE": (0, 0, 139),
    "INDIGO": (75, 0, 130),
    "DARK_INDIGO": (54, 0, 94),
    "VIOLET": (128, 0, 128),
    "DARK_VIOLET": (80, 0, 80),
    "SQUARE_COLOR": (255, 255, 255),
    "SQUARE_THICKNESS": 4,
}

reverse_lookup = {v: k for k, v in color_names.items()}

scheduled_events = {
    "Event Time": None,
    "Event Type": None,
    "Event Object": None,
    "Status": None,
}

text_rect = None
# %%

# Function to post a custom event with a timestamp and mouse position
def post_scheduled_event(delay, position):
    event_time = pygame.time.get_ticks() + delay
    event = pygame.event.Event(SCHEDULED_EVENT, {
        'timestamp': event_time,
        'position': position
    })
    pygame.event.post(event)
```

```
class Balls:
    # ball = ball(x, y, dx, dy, radius, color,␣
 ↪ball_color,clicked_colors[i],reinforcement_interval,change_over_delay)
    def __init__(self, x, y, dx, dy,
                 radius, ball_color, clicked_color,speed,
                 change_to_clicks,change_to_delay,
                 change_from_clicks,change_from_delay,
                 block_score_until_clicks,block_score_until_time,
                 time_required, clicks_required):#fixed_ratio

        self.x = x
        self.y = y
        self.dx = dx
        self.dy = dy
        self.min_speed = speed
        self.max_speed = None
        self.radius = radius
        self.clicked_color = clicked_color
        self.default_color = ball_color
        self.color = ball_color
        self.colorname = reverse_lookup.get(self.color, "Unknown Color")
        self.clicked = False
        self.clicks = 0
        self.valid_clicks = 0 # set the amount of clicks to zero, so we can use␣
 ↪the fixed ratio & interval
        self.score = 0
        self.block_score_until_time = block_score_until_time
        self.block_score_until_clicks = block_score_until_clicks # self.
 ↪valid_clicks
        self.change_to_clicks = change_to_clicks_current_ball
        self.change_to_delay = change_to_delay
        self.change_from_clicks = change_from_clicks
        self.change_from_delay = change_from_delay
        self.time_required = time_required
        self.clicks_required = clicks_required


    def draw(self, screen):
        pygame.draw.circle(screen, self.color, (int(self.x), int(self.y)), self.
 ↪radius)

    def advance(self, dt):
        self.x += self.dx * dt
        self.y += self.dy * dt

        if self.x - self.radius < bounce_box_left:
```

```
            self.x = bounce_box_left + self.radius
            self.dx = abs(self.dx)
        elif self.x + self.radius > bounce_box_right:
            self.x = bounce_box_right - self.radius
            self.dx = -abs(self.dx)

        if self.y - self.radius < bounce_box_bottom:
            self.y = bounce_box_bottom + self.radius
            self.dy = abs(self.dy)
        elif self.y + self.radius > bounce_box_top:
            self.y = bounce_box_top - self.radius
            self.dy = -abs(self.dy)

    def darken_color(self):
        self.color = self.clicked_color# tuple(int(c * 0.8) for c in self.
 ↪color) #self.color = tuple(int(c * 0.8) for c in self.base_color)

    def reset_color(self):
        self.color = self.default_color

# %%
class Simulation:
    global screen
    def __init__(self, phase_options):
        global base_colors, clicked_colors
        base_colors = phase_options['base_colors']
        self.last_reinforced = None
        self.block_score_until_time = 0
        self.last_clicked = None
        self.last_clicked = None
        self.last_clicked_time = None
        self.last_reinforcement_time = None ##TODO: Must integrate this with␣
 ↪the other logic, alter this even if we click the same ball but don't score
        self.last_reinforced_ball_click_time = None         ## Added 8/20
        self.reinforcement_texts = []

        self.balls = self.init_balls(
            phase_options['number_balls'],
            phase_options['initial_speed'],
            phase_options['radii'],
            phase_options['base_colors'],
            phase_options['clicked_colors'],
            phase_options['change_to_clicks'],
            phase_options['change_to_delay'],
            phase_options['change_from_clicks'],
            phase_options['change_from_delay'],
            phase_options['block_score_until_time'],
```

```
            phase_options['block_score_until_clicks'],
            phase_options['time_required'],
            phase_options['clicks_required']
        )

    def init_balls(self, number_balls, initial_speed, radii, base_colors,␣
↪clicked_colors, change_to_clicks, change_to_delay, change_from_clicks,␣
↪change_from_delay, block_score_until_time, block_score_until_clicks,␣
↪time_required, clicks_required):
        global change_to_clicks_current_ball
        balls = []
        logtocsv.write_data(('################## INIT balls␣
↪######################'))
        event_string = str(current_seconds) + ', Init stimuli, ' +␣
↪str(total_score) + ', '

        for i in range(int(number_balls)):
            change_to_clicks_current_ball = change_to_clicks[i]
            radius = radii[i]
            speed = initial_speed[i] / 10
            while True:
                x = np.random.uniform(radius, windowX - radius)
                y = np.random.uniform(radius, windowY - radius)
                angle = np.random.uniform(0, 2 * np.pi)
                dx = np.random.choice([-1, 1]) * speed * np.cos(angle)
                dy = np.random.choice([-1, 1]) * speed * np.sin(angle)
                color = base_colors[i]


                new_ball = Balls(x, y, dx, dy, radius, base_colors[i],␣
↪clicked_colors[i],
                    initial_speed[i], change_to_clicks[i], change_to_delay[i],
                    change_from_clicks[i], change_from_delay[i],
                    block_score_until_clicks[i], block_score_until_time[i],
                    time_required[i], clicks_required[i])


                if not any(np.hypot(new_ball.x - existing_ball.x, new_ball.y -␣
↪existing_ball.y) < new_ball.radius + existing_ball.radius for existing_ball␣
↪in balls):
                    balls.append(new_ball)
                    break
                else:
                    print('Overlap Detected')

        logtocsv.write_data(event_string)
        return balls
```

```
    def draw_text_boxes(self, screen):
        # Calculate text box padding and spacing
        TEXT_BOX_PADDING = 10
        x_offset = windowX - margin_right  # Position of the rightmost edge of␣
↪the text boxes
        y_offset = margin_top

        # Font and color settings
        font = pygame.font.Font(None, 24)
        text_color = BLACK

        for i, ball in enumerate(self.balls):
            # Create text for each ball
            ball_info = (f"Ball {i + 1}: Color: {ball.colorname}\n"
                         f"Position: ({int(ball.x)}, {int(ball.y)})\n"
                         f"Speed: ({ball.dx:.2f}, {ball.dy:.2f})\n"
                         f"Clicks: {ball.clicks}\n"
                         f"Score: {ball.score}\n")

            text_surface = font.render(ball_info, True, text_color)
            screen.blit(text_surface, (x_offset + TEXT_BOX_PADDING, y_offset))

            # Update y_offset for the next ball
            y_offset += text_surface.get_height() + TEXT_BOX_PADDING

    def handle_collisions(self):
        for i in range(len(self.balls)):
            for j in range(i + 1, len(self.balls)):
                if np.hypot(self.balls[i].x - self.balls[j].x,
                            self.balls[i].y - self.balls[j].y) < self.balls[i].
↪radius + self.balls[
                        j].radius:
                    self.change_velocities(self.balls[i], self.balls[j])

    def change_velocities(self, p1, p2):
        m1, m2 = p1.radius ** 2, p2.radius ** 2
        M = m1 + m2
        r1, r2 = np.array([p1.x, p1.y]), np.array([p2.x, p2.y])
        d = np.linalg.norm(r1 - r2) ** 2
        v1, v2 = np.array([p1.dx, p1.dy]), np.array([p2.dx, p2.dy])
        u1 = v1 - 2 * m2 / M * np.dot(v1 - v2, r1 - r2) / d * (r1 - r2)
        u2 = v2 - 2 * m1 / M * np.dot(v2 - v1, r2 - r1) / d * (r2 - r1)
        p1.dx, p1.dy = u1
        p2.dx, p2.dy = u2

    def advance(self, dt):
```

```
        for ball in self.balls:
            ball.advance(dt)
        self.handle_collisions()


    def add_reinforcement_text(self, x, y):
        self.reinforcement_texts.append({
            'text': '+1',
            'x': x,
            'y': y,
            'font_size': 36,
            'alpha': 255
        })

    def update_reinforcement_texts(self):
        for text in self.reinforcement_texts[:]:
            text['y'] -= 1
            text['font_size'] += 1
            text['alpha'] -= 5
            if text['alpha'] <= 0:
                self.reinforcement_texts.remove(text)

    def draw_reinforcement_texts(self, screen):
        for text in self.reinforcement_texts:
            font = pygame.font.Font(None, text['font_size'])
            rendered_text = font.render(text['text'], True, WHITE)
            rendered_text.set_alpha(text['alpha'])
            screen.blit(rendered_text, (text['x'], text['y']))



# %%
def main():
    global screen, windowX, windowY, bounce_box_right, bounce_box_top,␣
 ↪square_rect, font, text_rect, current_seconds,clicked_on_ball, total_score,␣
 ↪phase_duration, current_phase
    # callback()
    logtocsv.write_data(('################## Phase '+str(current_phase)+'␣
 ↪######################'))
    clock = pygame.time.Clock()
    # sim = Simulation()
    shuffle_button_rect = pygame.Rect(windowX - 150, 20, 120, 30)
    shuffle_button_color = (255, 100, 100)
    total_score = 0

    # while True:
    print(current_seconds)
```

```
    for phase in phase_options:
        sim = Simulation(phase_options[phase])
        print(phase_options[phase]["duration"])
        phase_duration = phase_options[phase]["duration"]
        end_time = current_seconds + int(phase_options[phase]["duration"])
        print('End time:',end_time)
        start_time = current_seconds

        while current_seconds < end_time:
            # Handle events here
            current_seconds = pygame.time.get_ticks()/1000 #- start_time NOTE:␣
↪Removed this because the start time per phase was always changing
            # print(current_seconds)
            for event in pygame.event.get():
                event_string = str(current_seconds)+', ' + str(total_score) +␣
↪', '# start making my string
                if event.type == pygame.QUIT:
                    pygame.quit()
                    sys.exit()

                elif event.type == SCHEDULED_EVENT:
                    current_ticks = pygame.time.get_ticks()
                    if current_ticks >= event.timestamp:
                        # Process the event (in this case, we'll just print a␣
↪message)
                        print(f"Scheduled event triggered at position: {event.
↪position}")
                        for ball in sim.balls:
                            if ball.clicked:
                                ball.reset_color()
                                ball.clicked = False
                    else:
                        # Repost the event with the same original timestamp and␣
↪position
                        pygame.event.post(pygame.event.Event(SCHEDULED_EVENT, {
                            'timestamp': event.timestamp,
                            'position': event.position
                        }))

                elif event.type == pygame.MOUSEBUTTONDOWN:
                    if event.button == 1:
                        # logtocsv.write_data(str(current_seconds)+' Testing␣
↪doing a random string')

                        for ball in sim.balls:
                            # if current_seconds > ball.block_score_until:
```

```
                            #     break
                            if np.hypot(event.pos[0] - ball.x, event.pos[1] -␣
↪ball.y) < ball.radius: # Handle the clicked ball

                                # ball.block_score_until_time = current_seconds␣
↪+ ball.min_score_delay
                                clicked_on_ball = True
                                ball.darken_color()
                                ball.clicked = True
                                ball.clicks += 1
                                post_scheduled_event(100, 'TEST LINE 418')
                                # color = reverse_lookup.get(ball.color,␣
↪"Unknown Color")
                                #clicked_color = reverse_lookup.get(ball.color,␣
↪"Unknown Color") #ball.color
                                event_string += "Clicked: "+ball.colorname + ',␣
↪'
                                event_string += 'x='+ str(event.pos[0])+', ' +␣
↪' y=' + str(event.pos[1]) + ', '


                                # Determine if we need to use changeover logic
                                if sim.last_reinforced is not None and sim.
↪last_reinforced != ball:
                                    # ball.block_score_until_time = ball.
↪change_to_delay + current_seconds
                                    # if current_seconds <= ball.
↪block_score_until_time:  #TODO: Add blocker for click number
                                    #     print('Scoring blocked by change over␣
↪delay')
                                    #     break
                                    if sim.last_clicked == ball:
                                        ball.clicks_required -= 1
                                    else:
                                        ball.clicks_required = ball.
↪change_to_clicks - 1
                                        ball.block_score_until_time = ball.
↪change_to_delay + current_seconds
                                        # ball.block_score_until_time = ball.
↪change_to_delay + current_seconds
                                        print('Clicked',ball.colorname)
                                        # pass
                                    # Reset valid click count on any clicked␣
↪ball
                                    # ball.valid_clicks = 0
```

```
                                    print('Changed COlors, clicked:',ball.
↪colorname,'last color:',sim.last_reinforced)

                                    # Simulation.last_clicked = ball
                                    # ball.clicks_required -= 1
                                    sim.last_clicked = ball

                                    if ball.clicks_required <=0:
                                        if current_seconds < ball.
↪block_score_until_time:##TODO: This NEEDS to interact with sim.
↪block_score_until_time
                                            break
                                        sim.last_reinforced = ball
                                        ball.score += 1
                                        sim.last_reinforced_ball_click_time =␣
↪current_seconds # TODO: use this for change over delay
                                        total_score +=1
                                        ball.valid_clicks += 1
                                        # ball.clicks_required -= 1

                                    if current_seconds < ball.
↪block_score_until_time or current_seconds < sim.block_score_until_time:
                                        print('clicked:',ball.
↪colorname,current_seconds , "can't score now, score blocked by time", end='')
                                        for ball in sim.balls:
                                            print(ball.block_score_until_time,␣
↪end=' ,')
                                        print('')
                                        break

                                    elif sim.last_reinforced is not None and␣
↪sim.last_reinforced != ball:
                                        print('Changed Colors, clicked:',ball.
↪colorname,'last color:',sim.last_reinforced)
                                        Simulation.last_clicked = ball

                                    elif ball.valid_clicks < ball.
↪block_score_until_clicks: #        self.block_score_until_clicks =␣
↪block_score_until_clicks # self.valid_clicks
                                        print('clicked:',ball.
↪colorname,current_seconds , "can't score now, score blocked by score until␣
↪clicks", end='')
                                        # ball.valid_clicks +=1
                                        print()
                                        break
```

```
                                else:
                                    print('scored at',current_seconds,'Was␣
↪blocked until: ',ball.block_score_until_time)
                                    ball.score += 1
                                    sim.last_reinforced = ball
                                    sim.last_reinforced_ball_click_time =␣
↪current_seconds
                                    # ball.score += 1
                                    total_score += 1
                                    ball.block_score_until_time =␣
↪current_seconds + ball.time_required
                                    for ball in sim.balls:
                                        if np.hypot(event.pos[0] - ball.x,␣
↪event.pos[1] - ball.y) < ball.radius:
                                            pass
                                        else:
                                            ball.block_score_until_time =␣
↪current_seconds + ball.time_required

                    if not clicked_on_ball and not shuffle_button_rect.
↪collidepoint(event.pos):
                        event_string += 'Clicked: None, '
                        event_string += f'x={event.pos[0]}, y={event.pos[1]}, '

                    clicked_on_ball = False
                    if shuffle_button_rect.collidepoint(event.pos):
                        # Check if the shuffle button is clicked
                        sim = Simulation(phase_options[phase])  # Create a new␣
↪simulation to reorient all balls
                        event_string += 'Clicked: Shuffle, '
                        event_string += f'x={event.pos[0]}, y={event.pos[1]}, '
                        # print('Clicked: Shuffle')

                    for ball in sim.balls:
                        color = reverse_lookup.get(ball.color, "Unknown Color")
                        event_string += ' ' + str(color) +':'
                        event_string += ' x='+ str(int(ball.x)) +', '+ ' y='+␣
↪str(int(ball.y))+', ' + ' dx='+ str((ball.dx))+ ', '+' dy='+ str((ball.dy)) ␣
↪+', '+' clicks='+ str((ball.clicks))+', '+' score='+ str((ball.score))+','

                    logtocsv.write_data(event_string)

                # elif event.type == pygame.MOUSEBUTTONUP:
                #     if event.button == 1:
                #         for ball in sim.balls:
                #             if ball.clicked:
```

```
                #                 ball.reset_color()
                #                 ball.clicked = False
                elif event.type == pygame.VIDEORESIZE:
                    windowX, windowY = event.w, event.h
                    screen = pygame.display.set_mode((windowX, windowY), pygame.
↪RESIZABLE)
                    bounce_box_right = windowX - margin_right
                    bounce_box_top = windowY - margin_top
                    square_rect = pygame.Rect((windowX - square_size) // 2,␣
↪(windowY - square_size) // 2, square_size,
                                              square_size)

            screen.fill((0, 0, 0))
            sim.advance(20.0)

            for ball in sim.balls:
                ball.draw(screen)

            pygame.draw.rect(screen, SQUARE_COLOR, (margin, margin, windowX - 2␣
↪* (margin), windowY - 2 * (margin)),
                             SQUARE_THICKNESS)
            pygame.draw.rect(screen, shuffle_button_color, shuffle_button_rect)
            #TODO: Add drawing for reinforcement info here
            sim.update_reinforcement_texts()


            text_score = font.render(f'Score: {total_score}', True, YELLOW)
            text_rect_score = text_score.get_rect(center=(windowX // 2, windowY␣
↪- 60))
            screen.blit(text_score, text_rect_score)

            # text_ball_1 = font.render(f'Score: {total_score}', True, YELLOW)
            # text_rect_ball_1 = text_ball_1.get_rect(center=(windowX - 80, 60))
            # screen.blit(text_ball_1, text_rect_ball_1)

            # text_ball_2 = font.render(f'Score: {total_score}', True, YELLOW)
            # text_rect_ball_2 = text_ball_2.get_rect(center=(windowX - 80, 90))
            # screen.blit(text_ball_2, text_rect_ball_2)
            # Example attributes to display (replace these with actual␣
↪attributes you want to debug)
            debug_info = [
                sim.balls[1].clicks,
                "Phase:"+phase,
                "Phase Duration: "+str(phase_options[phase]["duration"]),
                'end time:'+str(end_time),
                "Time Remaining:"+str(round(end_time - current_seconds, 1)),
                "Current Time"+str(round(current_seconds, 1)),
```

```
                "Attribute 5: value5",
                "scoring blocked until:",ball.block_score_until_time
            ]

            # Starting y-position for the text
            start_y = 90
            line_height = 35  # Adjust this according to your font size and␣
 ↪spacing

            for i, attributes in enumerate(debug_info):
                text = font.render(f'{attributes}', True, YELLOW)
                text_rect = text.get_rect(center=(windowX - 250, start_y + i *␣
 ↪line_height))
                screen.blit(text, text_rect)


            font = pygame.font.Font(None, 36)
            text = font.render("Shuffle", True, (255, 255, 255))
            Simulation.draw_text_boxes(sim, screen)
            screen.blit(text, (windowX - 140, 25))
            pygame.display.flip()
            clock.tick(60)
        current_phase += 1
        print('Current time',current_seconds,'end tiime',end_time)
        Simulation.draw_text_boxes(sim, screen)

# %%
if __name__ == "__main__":

    main()
print(current_seconds)
```

```
0
300
End time: 300
scored at 15.746 Was blocked until:  0
Scheduled event triggered at position: TEST LINE 418
scored at 15.896 Was blocked until:  15.846
scored at 15.977 Was blocked until:  15.996
Scheduled event triggered at position: TEST LINE 418
Scheduled event triggered at position: TEST LINE 418
scored at 16.126 Was blocked until:  16.077
Scheduled event triggered at position: TEST LINE 418
Clicked YELLOW
Changed COlors, clicked: YELLOW last color: <__main__.Balls object at
0x00000203E3581D00>
Scheduled event triggered at position: TEST LINE 418
```

```
Changed COlors, clicked: YELLOW last color: <__main__.Balls object at
0x00000203E3581D00>
Scheduled event triggered at position: TEST LINE 418
Changed COlors, clicked: YELLOW last color: <__main__.Balls object at
0x00000203E3581D00>
Scheduled event triggered at position: TEST LINE 418
Changed COlors, clicked: YELLOW last color: <__main__.Balls object at
0x00000203E3581D00>
Scheduled event triggered at position: TEST LINE 418
```

```
An exception has occurred, use %tb to see the full traceback.

SystemExit
```

[ ]:
```python
def main():
    global screen, windowX, windowY, bounce_box_right, bounce_box_top,␣
 ↪square_rect, font, text_rect, current_seconds, clicked_on_ball, total_score,␣
 ↪phase_duration, current_phase
    # callback()
    logtocsv.write_data(('################## Phase ' + str(current_phase) + '␣
 ↪#######################'))
    clock = pygame.time.Clock()
    # sim = Simulation()
    shuffle_button_rect = pygame.Rect(windowX - 150, 20, 120, 30)
    shuffle_button_color = (255, 100, 100)
    total_score = 0

    # while True:
    print(current_seconds)
    for phase in phase_options:
        sim = Simulation(phase_options[phase])
        print(phase_options[phase]["duration"])
        phase_duration = phase_options[phase]["duration"]
        end_time = current_seconds + int(phase_options[phase]["duration"])
        print('End time:', end_time)
        start_time = current_seconds

        while current_seconds < end_time:
            # Handle events here
            current_seconds = pygame.time.get_ticks() / 1000  # Updated current␣
 ↪time
            # print(current_seconds)
            for event in pygame.event.get():
                event_string = str(current_seconds) + ', ' + str(total_score) +␣
 ↪', '  # Start making my string
```

```
                if event.type == pygame.QUIT:
                    pygame.quit()
                    sys.exit()

                elif event.type == SCHEDULED_EVENT:
                    current_ticks = pygame.time.get_ticks()
                    if current_ticks >= event.timestamp:
                        # Process the event (in this case, we'll just print a␣
↪message)
                        print(f"Scheduled event triggered at position: {event.
↪position}")
                        for ball in sim.balls:
                            if ball.clicked:
                                ball.reset_color()
                                ball.clicked = False
                    else:
                        # Repost the event with the same original timestamp and␣
↪position
                        pygame.event.post(pygame.event.Event(SCHEDULED_EVENT, {
                            'timestamp': event.timestamp,
                            'position': event.position
                        }))

                elif event.type == pygame.MOUSEBUTTONDOWN:
                    if event.button == 1:
                        # logtocsv.write_data(str(current_seconds)+' Testing␣
↪doing a random string')

                        for ball in sim.balls:
                            # if current_seconds > ball.block_score_until:
                            #     break
                            if np.hypot(event.pos[0] - ball.x, event.pos[1] -␣
↪ball.y) < ball.radius:  # Handle the clicked ball

                                # ball.block_score_until_time = current_seconds␣
↪+ ball.min_score_delay
                                clicked_on_ball = True
                                ball.darken_color()
                                ball.clicked = True
                                ball.clicks += 1
                                post_scheduled_event(100, 'TEST LINE 418')
                                # color = reverse_lookup.get(ball.color,␣
↪"Unknown Color")
                                #clicked_color = reverse_lookup.get(ball.color,␣
↪"Unknown Color") #ball.color
```

```
                                event_string += "Clicked: " + ball.colorname +␣
↪', '
                                event_string += 'x=' + str(event.pos[0]) + ', '␣
↪+ ' y=' + str(event.pos[1]) + ', '

                                # Determine if we need to use changeover logic
                                if sim.last_reinforced is not None and sim.
↪last_reinforced != ball:
                                    # ball.block_score_until_time = ball.
↪change_to_delay + current_seconds
                                    # if current_seconds <= ball.
↪block_score_until_time:  #TODO: Add blocker for click number
                                    #     print('Scoring blocked by change over␣
↪delay')
                                    #     break
                                    if sim.last_clicked == ball:
                                        ball.clicks_required -= 1
                                    else:
                                        ball.clicks_required = ball.
↪change_to_clicks - 1
                                        ball.block_score_until_time = ball.
↪change_to_delay + current_seconds
                                        # ball.block_score_until_time = ball.
↪change_to_delay + current_seconds
                                        print('Clicked', ball.colorname)
                                        # pass
                                    # Reset valid click count on any clicked␣
↪ball
                                    # ball.valid_clicks = 0
                                    print('Changed Colors, clicked:', ball.
↪colorname, 'last color:', sim.last_reinforced)

                                    # Simulation.last_clicked = ball
                                    # ball.clicks_required -= 1
                                    sim.last_clicked = ball

                                    if ball.clicks_required <= 0:
                                        if current_seconds < ball.
↪block_score_until_time:  # TODO: This NEEDS to interact with sim.
↪block_score_until_time
                                            break
                                        sim.last_reinforced = ball
                                        ball.score += 1
                                        sim.last_reinforced_ball_click_time =␣
↪current_seconds  # TODO: use this for change over delay
                                        total_score += 1
```

```
                                    ball.valid_clicks += 1
                                    # ball.clicks_required -= 1

                                    # Add reinforcement text
                                    sim.add_reinforcement_text(event.
↪pos[0], event.pos[1])

                                if current_seconds < ball.
↪block_score_until_time or current_seconds < sim.block_score_until_time:
                                    print('clicked:', ball.colorname,␣
↪current_seconds, "can't score now, score blocked by time", end='')
                                    for ball in sim.balls:
                                        print(ball.block_score_until_time,␣
↪end=' ,')
                                    print('')
                                    break

                                elif sim.last_reinforced is not None and␣
↪sim.last_reinforced != ball:
                                    print('Changed Colors, clicked:', ball.
↪colorname, 'last color:', sim.last_reinforced)
                                    Simulation.last_clicked = ball

                                elif ball.valid_clicks < ball.
↪block_score_until_clicks:  # self.block_score_until_clicks =␣
↪block_score_until_clicks # self.valid_clicks
                                    print('clicked:', ball.colorname,␣
↪current_seconds, "can't score now, score blocked by score until clicks",␣
↪end='')
                                    # ball.valid_clicks +=1
                                    print()
                                    break

                            else:
                                print('scored at', current_seconds, 'Was␣
↪blocked until: ', ball.block_score_until_time)
                                ball.score += 1
                                sim.last_reinforced = ball
                                sim.last_reinforced_ball_click_time =␣
↪current_seconds
                                # ball.score += 1
                                total_score += 1
                                ball.block_score_until_time =␣
↪current_seconds + ball.time_required
                                for ball in sim.balls:
```

```
                                    if np.hypot(event.pos[0] - ball.x,␣
↪event.pos[1] - ball.y) < ball.radius:
                                        pass
                                    else:
                                        ball.block_score_until_time =␣
↪current_seconds + ball.time_required

                    if not clicked_on_ball and not shuffle_button_rect.
↪collidepoint(event.pos):
                        event_string += 'Clicked: None, '
                        event_string += f'x={event.pos[0]}, y={event.pos[1]}, '

                    clicked_on_ball = False
                    if shuffle_button_rect.collidepoint(event.pos):
                        # Check if the shuffle button is clicked
                        sim = Simulation(phase_options[phase])  # Create a new␣
↪simulation to reorient all balls
                        event_string += 'Clicked: Shuffle, '
                        event_string += f'x={event.pos[0]}, y={event.pos[1]}, '
                        # print('Clicked: Shuffle')

                    for ball in sim.balls:
                        color = reverse_lookup.get(ball.color, "Unknown Color")
                        event_string += ' ' + str(color) + ':'
                        event_string += ' x=' + str(int(ball.x)) + ', ' + ' y='␣
↪+ str(int(ball.y)) + ', ' + ' dx=' + str((ball.dx)) + ', ' + ' dy=' +␣
↪str((ball.dy)) + ', ' + ' clicks...'

            # Update and draw reinforcement texts
            sim.update_reinforcement_texts()

            # Draw everything
            screen.fill(BLACK)  # Clear the screen
            for ball in sim.balls:
                ball.draw(screen)
            sim.draw_text_boxes(screen)
            sim.draw_reinforcement_texts(screen)
            pygame.draw.rect(screen, shuffle_button_color, shuffle_button_rect)␣
↪ # Draw shuffle button

            pygame.display.flip()  # Update the screen
            clock.tick(60)  # Maintain 60 FPS
```