Data Analyst Project

Data analyst project.py > InputBox > handle_event

```python
import pygame
import pandas as pd
import os

# Pygame setup
pygame.init()
screen = pygame.display.set_mode((600, 500))
pygame.display.set_caption("5K Split Time Entry")
font = pygame.font.Font(None, 32)
clock = pygame.time.Clock()

# Colors
COLOR_INACTIVE = pygame.Color('lightskyblue3')
COLOR_ACTIVE = pygame.Color('dodgerblue2')
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)

```

This initialises the screen so it will have the correct size when it is run.

```python
# Input box class
class InputBox:
    def __init__(self, x, y, w, h, text=''):
        self.rect = pygame.Rect(x, y, w, h)
        self.color = COLOR_INACTIVE
        self.text = text
        self.txt_surface = font.render(text, True, self.color)
        self.active = False

    def handle_event(self, event):
        if event.type == pygame.MOUSEBUTTONDOWN:
            # Toggle the active state
            self.active = self.rect.collidepoint(event.pos)
        if event.type == pygame.KEYDOWN and self.active:
            if event.key == pygame.K_RETURN:
                self.active = False
            elif event.key == pygame.K_BACKSPACE:
                self.text = self.text[:-1]
            else:
                allowed_chars = "abcdefghijklmnopqrstuvwxyz0123456789./"
                if event.unicode.lower() in allowed_chars:
                    self.text += event.unicode
            self.txt_surface = font.render(self.text, True, BLACK)

```

This is the class for the input boxes anywhere that needs an input on the screen. This is the allowed characters that can be entered into the boxes and backspace will get rid of them.

```
    def draw(self, screen):
        screen.blit(self.txt_surface, (self.rect.x+5, self.rect.y+5))
        pygame.draw.rect(screen, self.color, self.rect, 2)

    def update(self):
        self.color = COLOR_ACTIVE if self.active else COLOR_INACTIVE

    def get_value(self):
        text = self.text.strip().lower()
        if text == "n/a" or text == "na":
            return "N/A"
        try:
            return float(self.text)
        except:
            return None
```

This is still part of the input class, this is how it will appear on screen when it is called.

```
# Back button for pages
back_button = pygame.Rect(20, 20, 80, 40)

def draw_back_button():
    pygame.draw.rect(screen, COLOR_ACTIVE, back_button)
    label = font.render("Back", True, WHITE)
    screen.blit(label, (back_button.x + 15, back_button.y + 7))

# Menu buttons
menu_buttons = {
    "Top Scores": pygame.Rect(200, 150, 200, 50),
    "Recent Scores": pygame.Rect(200, 220, 200, 50),
    "Average Scores": pygame.Rect(200, 290, 200, 50),
    "Predicted Scores": pygame.Rect(200, 360, 200, 50),
    "Enter Times": pygame.Rect(200, 430, 200, 50),
}

def draw_menu():
    screen.fill(WHITE)
    title = font.render("5K Running Stats Menu", True, BLACK)
    screen.blit(title, (180, 80))
    for text, rect in menu_buttons.items():
        pygame.draw.rect(screen, COLOR_ACTIVE, rect)
        label = font.render(text, True, WHITE)
        screen.blit(label, (rect.x + 30, rect.y + 12))
```

This is the button to go back to the menu which will appear on any sub screen.

These are the buttons that will appear on the main menu to take you to any sub menu.

This will draw the title on the main screen and how it will look.

```
# Input screen setup
labels = ["5K Total", "Split 1K", "Split 2K", "Split 3K", "Split 4K", "Split 5K"]
input_boxes = [InputBox(250, 150 + i*50, 140, 32) for i in range(len(labels))]

submit_button = pygame.Rect(230, 500, 140, 40)

def draw_input_screen():
    screen.fill(WHITE)
    title = font.render("Enter your times (type N/A if not available)", True, BLACK)
    screen.blit(title, (70, 70))

    for i, box in enumerate(input_boxes):
        label = font.render(labels[i], True, BLACK)
        screen.blit(label, (120, 150 + i*50))
        box.draw(screen)

    pygame.draw.rect(screen, COLOR_ACTIVE, submit_button)
    submit_label = font.render("Submit", True, WHITE)
    screen.blit(submit_label, (submit_button.x + 40, submit_button.y + 8))

    draw_back_button()
```

These are the labels above the boxes on the screen which you enter the time.

This is the screen to enter the times that you have got, the boxes are separated evenly going down the screen.

The submit button is written at the bottom so it can be clicked when the user is ready.

```
def save_to_csv(values):
    df_new = pd.DataFrame([{
        "Total_5k": values[0],
        "Split_1k": values[1],
        "Split_2k": values[2],
        "Split_3k": values[3],
        "Split_4k": values[4],
        "Split_5k": values[5],
    }])
    if os.path.exists(FILE_PATH):
        df_new.to_csv(FILE_PATH, mode='a', header=False, index=False)
    else:
        df_new.to_csv(FILE_PATH, index=False)

def draw_coming_soon(title_text):
    screen.fill(WHITE)
    draw_back_button()
    title = font.render(title_text, True, BLACK)
    screen.blit(title, (180, 250))
    coming = font.render("Coming Soon!", True, BLACK)
    screen.blit(coming, (230, 300))
```

This is creating the data frame and saving it to the csv file attached to keep the data entered.

This is the coming soon screen which will be displayed on all other screens that are not completed yet.

```
def reset_input_boxes():
    for box in input_boxes:
        box.text = ""
        box.txt_surface = font.render("", True, BLACK)

running = True
while running:
    screen.fill(WHITE)
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

        if current_screen == "menu":
            if event.type == pygame.MOUSEBUTTONDOWN:
                pos = event.pos
                for name, rect in menu_buttons.items():
                    if rect.collidepoint(pos):
                        if name == "Top Scores":
                            current_screen = "top_scores"
                        elif name == "Recent Scores":
                            current_screen = "recent_scores"
                        elif name == "Average Scores":
                            current_screen = "average_scores"
                        elif name == "Predicted Scores":
                            current_screen = "predicted_scores"
                        elif name == "Enter Times":
                            current_screen = "input"
                            reset_input_boxes()
```

This will reset the input boxes when submit is clicked so that they will be empty.

This allows you to exit if the exit in the top right x is clicked.

This shows the main menu screen and detects if the button for the sunscreen is clicked.

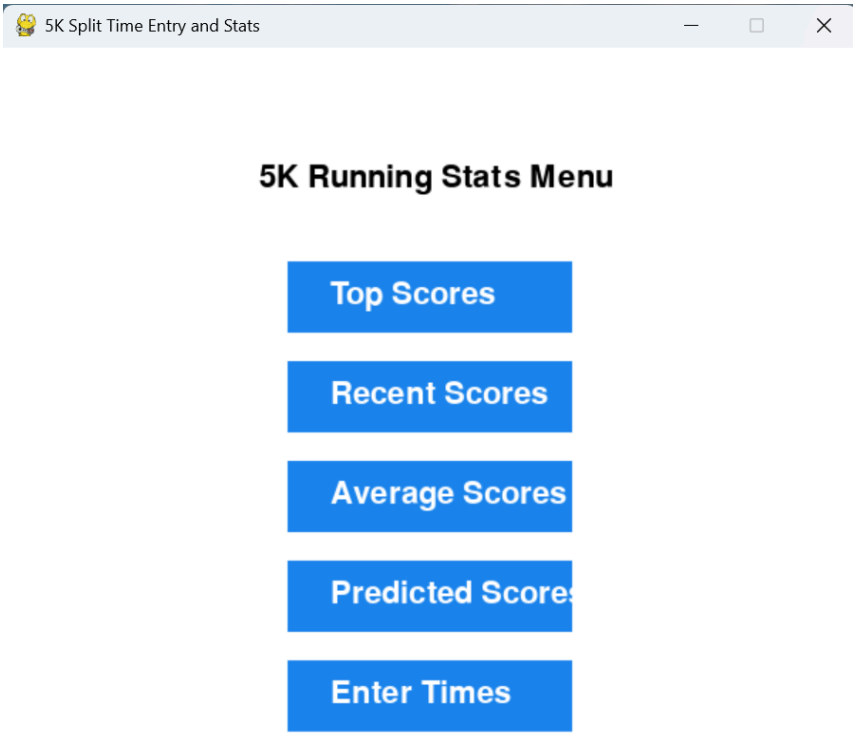
```
        elif current_screen == "input":
            for box in input_boxes:
                box.handle_event(event)
            if event.type == pygame.MOUSEBUTTONDOWN:
                if back_button.collidepoint(event.pos):
                    current_screen = "menu"
                if submit_button.collidepoint(event.pos):
                    values = [box.get_value() for box in input_boxes]
                    # Allow N/A or float, reject None
                    if all(v is not None for v in values):
                        save_to_csv(values)
                        reset_input_boxes()
                    else:
                        print("Please fill all fields with numbers or 'N/A'")

        else:  # On any data screen
            if event.type == pygame.MOUSEBUTTONDOWN:
                if back_button.collidepoint(event.pos):
                    current_screen = "menu"
```

This is the code for the input screen, knowing the values of the input boxes and saving them to the file.

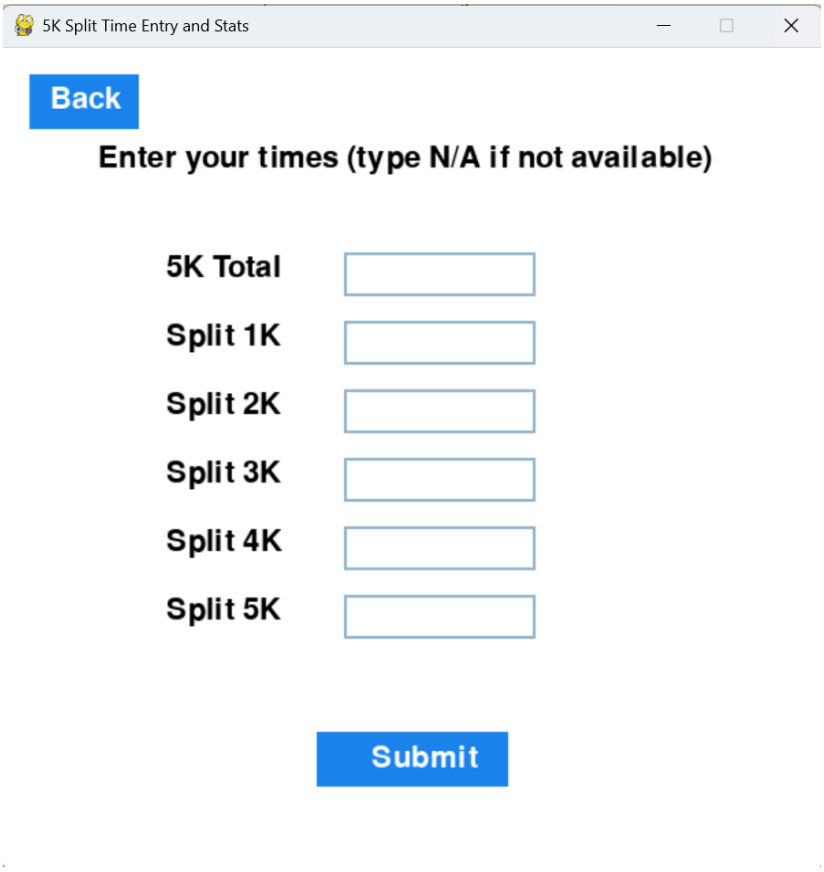
This is the button to get back onto the menu screen.

```
    # Draw current screen
    if current_screen == "menu":
        draw_menu()
    elif current_screen == "input":
        for box in input_boxes:
            box.update()
        draw_input_screen()
    elif current_screen == "top_scores":
        draw_top_scores()
    elif current_screen == "recent_scores":
        draw_recent_scores()
    elif current_screen == "average_scores":
        draw_average_scores()
    elif current_screen == "predicted_scores":
        draw_predicted_scores()

    pygame.display.flip()
    clock.tick(30)

pygame.quit()
```

This called the functions to draw the screen for each of the submenus.



This is what the menu screen looks like when the game is run.



This is what the enter times screen looks like when the button is clicked.

```python
def draw_top_scores():

    screen.fill(WHITE)
    title = font.render("Top 10 Times!", True, BLACK)
    screen.blit(title, (240, 70))

    # Check if data exists
    if not os.path.exists(FILE_PATH) or os.stat(FILE_PATH).st_size == 0:
        no_data = font.render("No data available yet!", True, BLACK)
        screen.blit(no_data, (200, 300))
        return
```

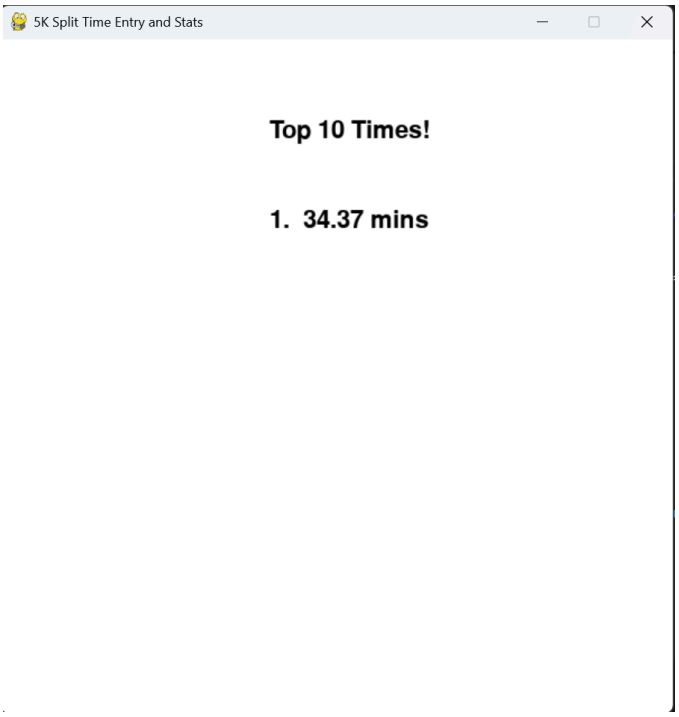
This is the start of the top 10 times screen and writes the title and then checks if the file that wants to be accessed exists.

```python
    try:
        # Read CSV
        df = pd.read_csv(FILE_PATH)

        # Drop rows without valid 5k total time
        df = df[pd.to_numeric(df["Total_5k"], errors='coerce').notna()]

        # Sort ascending (fastest times first)
        df = df.sort_values(by="Total_5k").head(10).reset_index(drop=True)

        # Display top 10 on screen
        y_offset = 150
        for i, row in df.iterrows():
            time_text = f"{i+1}.  {row['Total_5k']} mins"
            entry = font.render(time_text, True, BLACK)
            screen.blit(entry, (240, y_offset))
            y_offset += 40

    except Exception as e:
        error_msg = font.render(f"Error reading scores: {str(e)}", True, (200, 0, 0))
        screen.blit(error_msg, (100, 300))
```

This is now reading the file and sorting it by the top 10 times.

Then it writes it on the screen jumping by 40 pixels each time for spacing.

Then it is error handling just in case anything goes wrong in between so it does not crash.



This is what the screen looks like when the top scores button is clicked.

```python
def draw_recent_scores():

    screen.fill(WHITE)
    title = font.render("10 Recent Times!", True, BLACK)
    screen.blit(title, (240, 70))

    # Check if data exists
    if not os.path.exists(FILE_PATH) or os.stat(FILE_PATH).st_size == 0:
        no_data = font.render("No data available yet!", True, BLACK)
        screen.blit(no_data, (200, 300))
        return
```

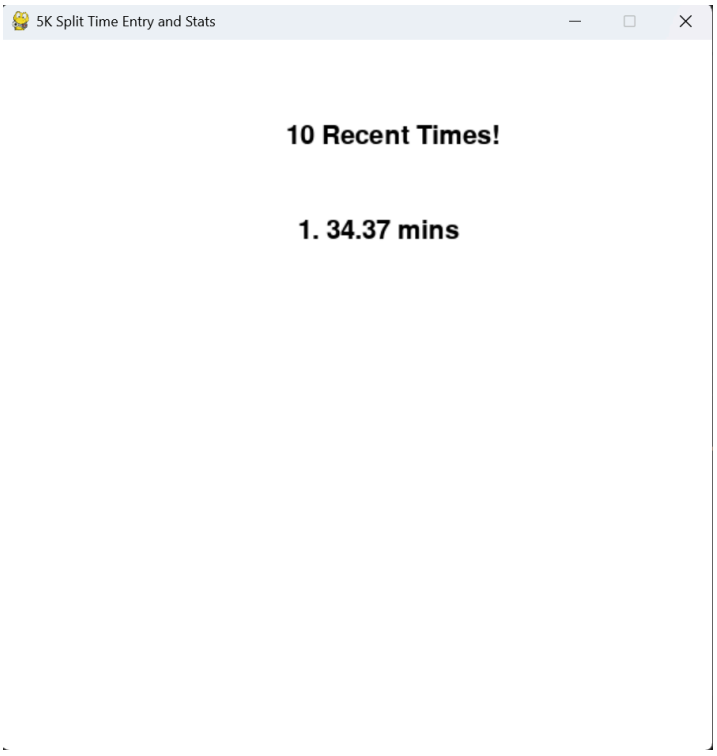
This is the recent scores screen and will draw the title and check if the file exists.

```python
try:
    # Read CSV
    df = pd.read_csv(FILE_PATH)

    # Get the last 10 rows (most recent entries)
    recent_df = df.tail(10).iloc[::-1].reset_index(drop=True)  # reverse order for newest first

    # Display them on screen
    y_offset = 150
    for i, row in recent_df.iterrows():
        line = f"{i+1}. {row['Total_5k']} mins"
        entry = font.render(line, True, BLACK)
        screen.blit(entry, (250, y_offset))
        y_offset += 40

except Exception as e:
        error_msg = font.render(f"Error reading scores: {str(e)}", True, (200, 0, 0))
        screen.blit(error_msg, (100, 300))
```

This reads the file and then sorts it by reverse order.

Then it will show it on screen with a y offset each time for spacing.

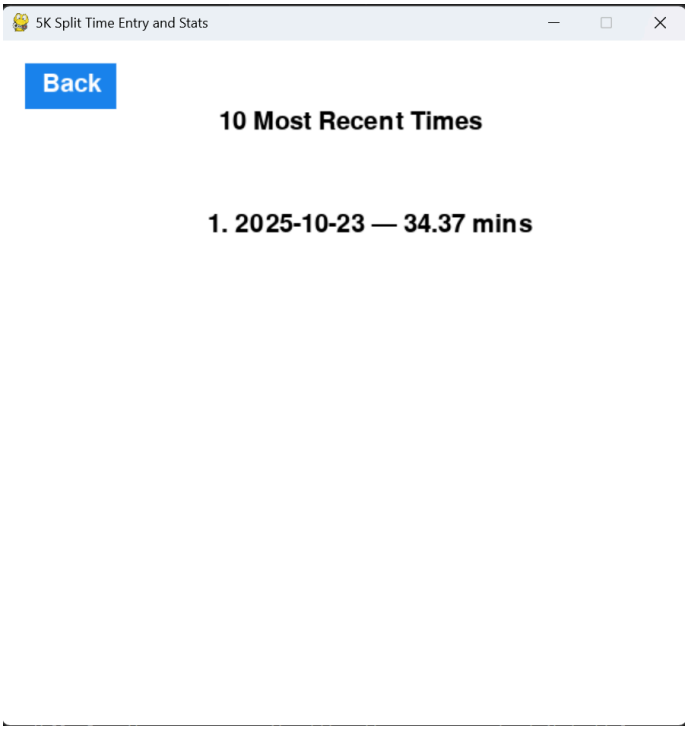
There is also error handling in case anything goes wrong to prevent a crash.



This is what the recent scores screen looks like when the recent scores button is clicked.

```
y_offset = 150
for i, row in recent_df.iterrows():
    date = row['Date'] if 'Date' in df.columns else 'Unknown Date'
    line = f"{i+1}. {date} — {row['Total_5k']} mins"
    entry = font.render(line, True, BLACK)
    screen.blit(entry, (180, y_offset))
    y_offset += 40
```

There is now a date which will be displayed on recent scores.



This is what the updated screen looks like on the recent scores menu.

```python
def draw_recent_graph(df):
    if df.empty:
        return

    # Graph area
    graph_rect = pygame.Rect(350, 150, 220, 300)
    pygame.draw.rect(screen, (230, 230, 230), graph_rect)  # light gray background
    pygame.draw.rect(screen, BLACK, graph_rect, 2)  # border

    # Numeric Total_5k values
    times = pd.to_numeric(df['Total_5k'], errors='coerce').dropna()
    if times.empty:
        return

    min_time = times.min()
    max_time = times.max()
    range_time = max_time - min_time
    if range_time == 0:
        range_time = 1  # prevent division by zero
```

This is the start of the graph that is going to be made alongside the recent scores.

It plots where the graph is going to be on the screen and generates the most recent time.

It sets a maximum and minimum range on the y coordinate so the graph looks sensible.

```python
points = []
n_points = len(times)
for i, t in enumerate(times):
    # x-coordinate: newest on the right
    if n_points == 1:
        x = graph_rect.x + graph_rect.width / 2
    else:
        # Reverse order: i=  (variable) graph_rect: Rect
        x = graph_rect.x + graph_rect.width - 10 - i * (graph_rect.width - 20) / (n_points - 1)
    y = graph_rect.y + graph_rect.height - 10 - ((t - min_time) / range_time) * (graph_rect.heig
    points.append((x, y))

# Draw connecting lines
if len(points) > 1:
    pygame.draw.lines(screen, (0, 100, 200), False, points, 3)

# Draw points as red circles
for x, y in points:
    pygame.draw.circle(screen, (200, 0, 0), (int(x), int(y)), 5)

# Draw x-axis labels (entry numbers)
for i, (x, _) in enumerate(points):
    label = font.render(str(i+1), True, BLACK)
    screen.blit(label, (x - 5, graph_rect.y + graph_rect.height + 5))
```

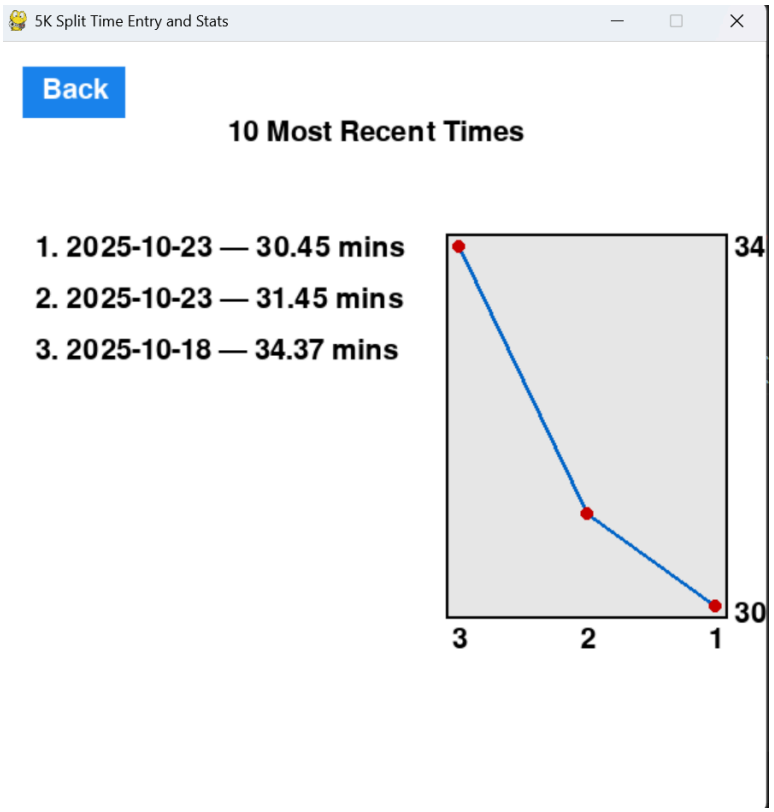
This plots the points on the graph with the correct distancing.

It makes the most recent data on the right.

It draws the lines between the plotted points and draws the points as circles.

```python
# Draw min/max labels on y-axis
min_label = font.render(f"{min_time:.2f}", True, BLACK)
max_label = font.render(f"{max_time:.2f}", True, BLACK)
screen.blit(min_label, (graph_rect.x + graph_rect.width + 5, graph_rect.y + graph_rect.height - 15))
screen.blit(max_label, (graph_rect.x + graph_rect.width + 5, graph_rect.y))
```

This is the labels for both of the axes and draws them onto the screen.



This is what the update's recent time screen looks like with the graph.