MAPÚA UNIVERSITY

SCHOOL OF ELECTRICAL, ELECTRONICS, AND COMPUTER ENGINEERING

<Sports Play: Football Betting>

CPE106L (Software Design Laboratory)
<B31>

Member 1 (Nicole Allyson B. Torres)
Member 2 (FirstName M.I. LastName)
Member 3 (FirstName M.I. LastName)

Group No.: **9**

Documents Sections:

Sport Play: Football Betting

Project Specifications: Sport Play: Football Betting

Game Overview:

Welcome to our Football Betting Game! Players can bet on simulated football matches between various teams in this game. The game uses a graphical user interface (GUI) implemented using the Tkinter library in Python. Match schedules are predefined, and new matches are available for betting after each match concludes. The game is purely virtual and does not involve real money gambling. The project uses Python and the Tkinter GUI library to implement a simple football betting simulation game. It simulates football matches, allows users to place bets on teams, and calculates outcomes based on simulated match results.

Game Rules:

Age Restriction: Players must be 21 years old or older to participate.

Bankroll: Players start with a bankroll of 1000 units.

Match Betting: Players choose a team and enter a bet amount before each match.

Match Outcome: Matches are simulated with random scores. If the selected team wins, the player wins double the bet amount. If the match ends in a draw or the chosen team loses, the player loses the bet amount.

Game Flow:

Players can continue playing by betting on the next scheduled match after each match outcome. Players can check their current bankroll at any time during the game. Players can quit the game at any time.

Analysis

Our project aims to predict winning teams and earn virtual points based on bets. It targets individuals who want to learn betting strategies, calculate odds, and manage risk. Our simulation teaches concepts and demonstrates the impact of different betting strategies to help users make informed decisions. Our interactive platform simulates betting scenarios responsibly and is designed to be engaging and fun.

Upon review, we identified several important aspects that contribute to the functionality and user experience of the football betting simulation code.

1. FootballMatch Class:

- Represents the concept of a football match in the simulation by generating random scores and determining the winner.

2. BetWindow Class:

- Manages the interface for placing bets on selected teams, reflecting user interactions. It also utilizes variables like "team_choice" and "amount_entry" to capture user inputs. It displays options to choose a team and enter a bet amount, simulates the game, and shows the outcome with corresponding winnings or losses.

3. MenuWindow Class:

- Controls the main menu display and user navigation within the game and shows the current match of the day and options to place bets, play again, or quit the game and tracks and updates the player's bankroll ("bankroll" attribute) based on betting outcomes.

4. PlayerInfoWindow Class:

- Collects player information such as name and age, ensuring game eligibility and validating player age for game rules compliance.

5. GameWindow Class:

- Orchestrates the overall game flow, including introduction, player info collection, and menu display, and manages the initialization of game components, flow, and interactions between different game elements.

The code for the football betting simulation contains several key nouns, including "match," "bet," "player," "bankroll," and "team." These align with verbs such as "simulate," "place," "update," and "start," reflecting dynamic actions like simulating match outcomes and managing bets. Together, these components form a structured and interactive environment for users to learn various betting strategies and responsible gambling

practices. The code's design emphasizes modularity, encapsulation, and user interaction. It is an engaging and educational tool for learning betting strategies, odds calculation, and responsible gambling practices. The GUI elements enhance the user experience and understanding.

Concepts

GUI Development involves utilizing the tkinter library to create Graphical User Interfaces (GUI) that include widgets such as labels, buttons, radio buttons, entries, and windows (Tk, Label, Button, Radiobutton, Entry). Object-Oriented Programming (OOP) organizes code into modular components with distinct functionalities by implementing classes such as FootballMatch, BetWindow, MenuWindow, PlayerInfoWindow, and GameWindow. Randomization is accomplished using the random module to generate random scores for simulating football matches via the simulate_match method in the FootballMatch class. Event Handling manages user interactions and events such as button clicks (place_bet, start_game, play_again, quit_game) through callback functions. Data Management includes storing and updating data such as player information, match details, and bankroll (bankroll attribute in MenuWindow, match_names list in MenuWindow). Message Display involves showing informational and error messages to users using the Tkinter message box. Flow Control controls the game's flow, navigation between screens (intro_window, player_info_window, menu_window), and managing game states (current_match_index). User Input Processing retrieves and processes user input such as player name, age, bet amount, and team choice. These concepts work together to facilitate the development of an interactive football betting simulation with a graphical interface, user input handling, game logic implementation, and feedback mechanisms for user actions and game outcomes.

Relationships between Concepts

The project has several components that interact with each other. These include:

- FootballMatch: This simulates a football match by generating scores for two teams and determining the winner.

- BetWindow: This allows users to place bets by selecting teams, entering bet amounts, and seeing payout information.

- MenuWindow: This is the main menu where players can see their information, available matches, and balance. It also manages game flow by creating and interacting with FootballMatch instances.

-PlayerInfoWindow: This is where players enter their information to ensure they meet age requirements before playing.

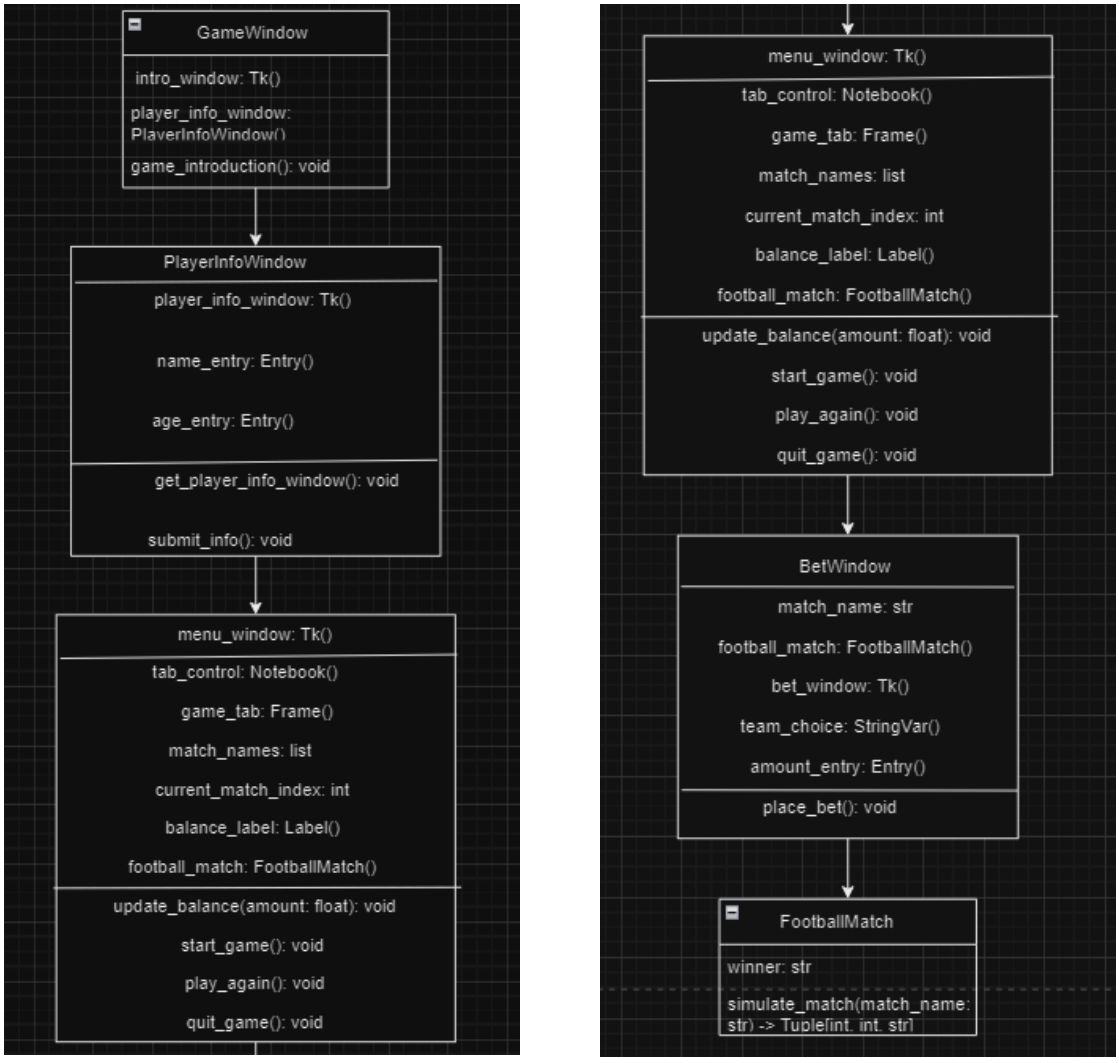
- GameWindow: This guides players through the game's introduction process, including inputting player information before proceeding to the main menu.

- GUI Components: These include various elements such as labels, buttons, radio buttons, and entries used in different windows to create interactive interfaces.

- Randomization: This is used to generate random scores for football matches.

- Messagebox: This shows informative messages to the user, such as match outcomes, bet results, and error notifications.

The diagram would illustrate these connections, emphasizing composition over inheritance for assembling the solution's objects and functionalities.



Nouns and Verbs

These nouns and verbs represent the functionalities and interactions of the football betting simulation program.

Nouns:

1. `FootballMatch`
2. `winner`
3. `BetWindow`
4. `match_name`
5. `menu_window`
6. `football_match`
7. `bet_window`
8. `team_choice`
9. `amount_entry`
10. `MenuWindow`
11. `player_name`
12. `match_names`
13. `current_match_index`
14. `tab_control`
15. `game_tab`
16. `balance_label`
17. `bankroll`
18. `FootballMatch`
19. `PlayerInfoWindow`
20. `intro_window`
21. `player_info_window`
22. `name_entry`
23. `age_entry`
24. `submit_button`
25. `GameWindow`
26. `intro_label`
27. `welcome_label`
28. `reminder_label`
29. `next_button`

Verbs:

1. `__init__` (for initializing objects)
2. `simulate_match`
3. `place_bet`
4. `update_balance`
5. `start_game`
6. `play_again`
7. `quit_game`

8. `get_player_info_window`
9. `submit_info`
10. `game_introduction`

Design and Implementation of Classes

class FootballMatch:

```
    def __init__(self):
        self.winner = None

    def simulate_match(self, match_name):
        # Adjust the weights to increase winning chances for higher scores
        score1 = random.choices(range(6), weights=[15, 20, 25, 20, 15, 5])[0]
        score2 = random.choices(range(6), weights=[15, 20, 25, 20, 15, 5])[0]

        if score1 > score2:
            self.winner = match_name.split(" vs. ")[0]
        elif score2 > score1:
            self.winner = match_name.split(" vs. ")[1]
        else:
            self.winner = "Draw"

        return score1, score2, self.winner  # Also return the winner
```

class BetWindow:

```
    def __init__(self, match_name, menu_window, football_match):
        self.match_name = match_name
        self.menu_window = menu_window  # Store the MenuWindow instance
        self.football_match = football_match  # Store the FootballMatch instance
        self.bet_window = tk.Tk()
        self.bet_window.title("Place Bet")

        team_label = tk.Label(self.bet_window, text=f"Choose a team to bet on
({match_name}):")
        team_label.pack()

        self.team_choice = tk.StringVar()
        team_choice_radio = tk.Radiobutton(self.bet_window, text=match_name.split(" vs.
")[0], variable=self.team_choice, value=match_name.split(" vs. ")[0])
        team_choice_radio.pack()
        team_choice_radio = tk.Radiobutton(self.bet_window, text=match_name.split(" vs.
")[1], variable=self.team_choice, value=match_name.split(" vs. ")[1])
        team_choice_radio.pack()
```

```
        amount_label = tk.Label(self.bet_window, text="Enter bet amount:")
        amount_label.pack()
        self.amount_entry = tk.Entry(self.bet_window)
        self.amount_entry.pack()

        bet_button = tk.Button(self.bet_window, text="Place Bet", command=lambda:
self.place_bet())
        bet_button.pack()

    def place_bet(self):
        team_name = self.team_choice.get()
        amount = float(self.amount_entry.get())

        # Simulate match to determine winner
        _, _, winner = self.football_match.simulate_match(self.match_name)

        if winner == "Draw":
            messagebox.showinfo("Draw", f"The match ended in a draw.\nNo winners or
losers.")
        elif team_name == winner:
            payout = amount * 2
            self.menu_window.update_balance(payout)  # Update balance with winning
amount
            messagebox.showinfo("Congratulations!", f"You won {payout} units.\nMatch
Winner: {winner}")
        else:
            self.menu_window.update_balance(-amount)  # Deduct bet amount from balance
            messagebox.showinfo("Sorry!", f"You lost your bet.\nMatch Winner: {winner}")

        self.bet_window.geometry("300x150")  # Adjust size
        self.bet_window.destroy()
class PlayerInfoWindow:
    def get_player_info_window(self):
        self.player_info_window = tk.Tk()
        self.player_info_window.title("Player Information")

        welcome_label = tk.Label(self.player_info_window, text="Welcome to Sports Win
Betting Football", font=("Arial", 14, "bold"))
        welcome_label.pack(pady=10)

        info_label = tk.Label(self.player_info_window, text="--- Player Information ---",
font=("Arial", 12, "bold"))
        info_label.pack(pady=10)
```

```
name_label = tk.Label(self.player_info_window, text="Enter your name:")
        name_label.pack()
        self.name_entry = tk.Entry(self.player_info_window)
        self.name_entry.pack()

        age_label = tk.Label(self.player_info_window, text="Enter your age:")
        age_label.pack()
        self.age_entry = tk.Entry(self.player_info_window)
        self.age_entry.pack()

        submit_button = tk.Button(self.player_info_window, text="Submit",
command=self.submit_info)
        submit_button.pack(pady=20)

    def submit_info(self):
        player_name = self.name_entry.get()
        player_age = int(self.age_entry.get())

        if player_age < 21:
            messagebox.showerror("Error", "You must be at least 21 years old to play this
game.")
            self.player_info_window.destroy()  # Close the current window
            self.get_player_info_window()  # Reopen the player info window
        else:
            self.player_info_window.destroy()
            MenuWindow(player_name)
```

These were the GUI classes:

class MenuWindow:

```
    def __init__(self, player_name):

        self.player_name = player_name

        self.menu_window = tk.Tk()

        self.menu_window.title("Main Menu")

        self.menu_window.geometry("600x400")
```

```
self.match_names = [
        "Germany vs. France",
        "Brazil vs. Argentina",
        "England vs. Germany",
        "France vs. Portugal",
        "Korea vs. Japan",
        "Costa Rica vs. Mexico",
        "Australia vs. Netherlands"

    ]
    self.current_match_index = 0  # Initialize current match index

    self.tab_control = ttk.Notebook(self.menu_window)

    self.game_tab = tk.Frame(self.tab_control)
    self.tab_control.add(self.game_tab, text="Game")
    self.tab_control.pack(expand=1, fill="both")

    menu_label = tk.Label(self.game_tab, text=f"Welcome, {self.player_name}!",
font=("Arial", 14, "bold"))
    menu_label.pack(pady=20)

    self.match_label = tk.Label(self.game_tab, text="Scheduled Match of the Day:")
    self.match_label.pack()

    self.match_info_label = tk.Label(self.game_tab, text="Italy vs. Spain")
    self.match_info_label.pack()
```

```
start_game_button = tk.Button(self.game_tab, text="Place Bet", command=self.start_game)
        start_game_button.pack(pady=10)

        play_again_button = tk.Button(self.game_tab, text="Play Again",
command=self.play_again)
        play_again_button.pack(pady=10)

        quit_game_button = tk.Button(self.game_tab, text="Quit Game",
command=self.quit_game)
        quit_game_button.pack(pady=10)

        # Initialize bankroll
        self.bankroll = 1000  # Starting balance

        # Display initial balance
        self.balance_label = tk.Label(self.game_tab, text=f"Bankroll: {self.bankroll} units")
        self.balance_label.pack()

        # Create FootballMatch instance
        self.football_match = FootballMatch()

    def update_balance(self, amount):
        self.bankroll += amount
        self.balance_label.config(text=f"Bankroll: {self.bankroll} units")

    def start_game(self):
        match_name = self.match_info_label.cget("text")
        BetWindow(match_name, self, self.football_match)  # Pass FootballMatch instance
```

```
def play_again(self):
        new_match_name = self.match_names[self.current_match_index]
        self.current_match_index = (self.current_match_index + 1) % len(self.match_names)
        self.match_info_label.config(text=new_match_name)
        BetWindow(new_match_name, self, self.football_match)  # Pass FootballMatch instance


    def quit_game(self):
        self.menu_window.destroy()


class GameWindow:
    def __init__(self):
        self.intro_window = None
        self.player_info_window = PlayerInfoWindow()


    def game_introduction(self):
        self.intro_window = tk.Tk()
        self.intro_window.title("Sports Win Betting Football!")


        intro_label = tk.Label(self.intro_window, text="--- Sports Win Betting Football! ---", font=("Arial", 16, "bold"))
        intro_label.pack(pady=20)


        welcome_label = tk.Label(self.intro_window, text="Welcome to Sports Win Betting Football! You will be betting on a football match between two teams.", wraplength=400)
        welcome_label.pack(pady=10)


        reminder_label = tk.Label(self.intro_window, text="*Reminder: This game is only played for ages 21 years old and above.", font=("Arial", 10, "italic"))
        reminder_label.pack(pady=10)
```

```
def next_step():
    self.intro_window.destroy()
    self.player_info_window.get_player_info_window()

next_button = tk.Button(self.intro_window, text="Start Betting", command=next_step)
next_button.pack(pady=20)

self.intro_window.mainloop()
```