Game 2: Farkle

T1 Planning and Design based on Mathematical and Declarative Programming techniques:

In designing the Farkle, mathematical structures and declarative programming techniques are pivotal in creating a functional, maintainable, and enjoyable gameplay experience. This game uses various software engineering principles and mathematical concepts to ensure that it operates smoothly and provides an engaging user experience.

Mathematical Structures:

Following mathematical structures were used:

- **Scoring Mechanism:** The scoring mechanism in Farkle is based on mathematical calculations of point combinations derived from dice rolls. Each combination of dice corresponds to a specific point value, defined by a dictionary mapping in the POINTS constant.
- **Randomization:** Randomization plays a crucial role in simulating dice rolls. The roll_dice function generates random dice outcomes based on the number of dice rolled, facilitating the unpredictable nature of the game and adding an element of chance.
- **Decision Making:** The game involves strategic decision-making by players in selecting scoring combinations. Players must analyze their current dice roll and choose the most advantageous combination to maximize their score, demonstrating mathematical reasoning and probability assessment.

Declarative Programming Techniques:

Following declarative programming techniques were used:

- **GUI Layout:** The graphical user interface (GUI) is designed declaratively using the Tkinter library. Widgets such as labels, buttons, and listboxes are defined and arranged in a visually intuitive manner, providing a user-friendly interface for player interaction.
- **Event Handling:** Button clicks and user inputs are mapped to corresponding event handlers, enabling declarative management of player actions. Each button

click triggers a specific action, such as rolling the dice, choosing a scoring combination, or ending the turn, facilitating a smooth user experience.

- **State Management:** The game state, including player scores, current dice roll, and turn status, is managed declaratively within the FarkleGame class. Methods such as update_status and update_players_label update the GUI to reflect the current state of the game, ensuring consistency and clarity.

Software Engineering Principles:

In the making of this game, following software engineering principles were used:

Modularity and Encapsulation: The Farkle game is designed with a modular structure, separating different components such as player management, game logic, and GUI presentation. Each class or function is encapsulated with a specific responsibility, promoting code reusability and maintainability.

Abstraction: The game abstracts away complex scoring rules and player interactions into simple user interfaces. Players interact with the game through buttons and visual cues, shielding them from the underlying implementation details.

Scalability: The game design allows for scalability in terms of the number of players, both human and AI. It prompts the user for the number of players and dynamically creates player instances accordingly, ensuring flexibility in accommodating different game configurations.

The planning and design of the Farkle game incorporate software engineering principles, mathematical concepts, and declarative programming techniques to create an interactive gaming experience. By modularizing the game, abstracting complex rules, using mathematical calculations, and designing a GUI, the game provides a great experience for players.

T2 Program Implementation based on Declarative Programming tools and techniques.

Source Code:

```
'''

Farkle Game:
Programming Language: Python
Interface: GUI based
```

```
Student Name: Aisha Abdi
Student ID: 22160571

'''

import tkinter as tk
from tkinter import messagebox, simpledialog, ttk
import random
from collections import OrderedDict
from operator import itemgetter

__version__ = '0.1.1'

TARGET_SCORE = 10000
POINTS = OrderedDict((
    ('111', 1000),
    ('666', 600),
    ('555', 500),
    ('444', 400),
    ('333', 300),
    ('222', 200),
    ('1', 100),
    ('5', 50),
))


def roll_dice(num):
    return ''.join(sorted(str(random.randint(1, 6)) for _ in range(num)))


class FarkleGame:
    def __init__(self, master):
        self.master = master
        self.master.title("Farkle")
        self.master.geometry("600x800")
        self.players = []
        self.current_player_index = 0
        self.current_score = 0
        self.current_roll = ""
        self.chosen = False
        self.last_round = False

        self.create_widgets()

    def choose_combo(self):
        """Handle choosing a scoring combo."""
        selection = self.combos_listbox.curselection()
        if selection:
            index = selection[0]
```

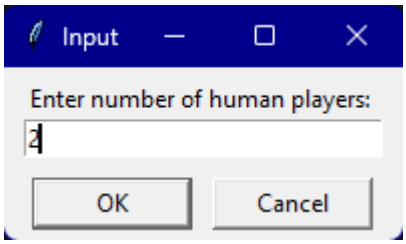
```
            combos = [combo for combo in POINTS if combo in self.current_roll]
            combo = combos[index]
            self.current_roll = self.current_roll.replace(combo, '', 1)
            self.current_score += POINTS[combo]
            self.chosen = True
            self.update_status()
            self.update_combos_listbox()


    def create_widgets(self):
        """Create the main widgets for the game."""
        self.info_label = ttk.Label(self.master, text="Welcome to Farkle!",
font=("Helvetica", 16))
        self.info_label.pack(pady=10)

        self.status_label = ttk.Label(self.master, text="", font=("Helvetica",
12))
        self.status_label.pack(pady=5)

        self.roll_button = ttk.Button(self.master, text="Roll Dice",
command=self.roll_dice_action)
        self.roll_button.pack(pady=5)

        self.combos_listbox = tk.Listbox(self.master, height=10, width=50)
        self.combos_listbox.pack(pady=5)

        self.choose_button = ttk.Button(self.master, text="Choose Combo",
command=self.choose_combo)
        self.choose_button.pack(pady=5)

        self.end_turn_button = ttk.Button(self.master, text="End Turn",
command=self.end_turn)
        self.end_turn_button.pack(pady=5)

        self.players_label = ttk.Label(self.master, text="",
font=("Helvetica", 12))
        self.players_label.pack(pady=10)

        self.start_game()

    def start_game(self):
        """Initialize the game with player inputs."""
        human_players = self.input_to_int("Enter number of human players: ")
        ai_players = self.input_to_int("Enter number of AI players: ")
        if not human_players and not ai_players:
            return

        names = ['Mary AI', 'Bob AI', 'Ben AI', 'Eryn AI', 'John AI', 'Ellen
AI', 'Elizabeth AI', 'Jason AI']
```

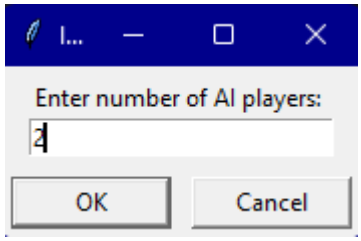
```
        random.shuffle(names)

        for num in range(1, human_players + 1):
            name = self.input_text(f"Player {num} enter your name: ")
            self.players.append({'ai': False, 'score': 0, 'name': name,
'done': False})
        for num in range(1, ai_players + 1):
            name = names[num - 1] if num <= len(names) else str(num)
            self.players.append({'ai': True, 'score': 0, 'name': name, 'done':
False})

        self.update_status()
        self.update_players_label()

    def input_to_int(self, message):
        """Get an integer input fromthe user."""
        while True:
            try:
                player_num = int(self.input_text(message))
                if 0 <= player_num < 100:
                    return player_num
                else:
                    messagebox.showerror("Input Error", "Input must be between
0 and 99.")
            except ValueError:
                messagebox.showerror("Input Error", "Invalid input. Please
enter a number.")

    def input_text(self, message):
        """Get a string input from the user."""
        return simpledialog.askstring("Input", message)

    def update_status(self):
        """Update the status label with the current player's information."""
        print("Current Score:", self.current_score)  # Debug print
        print("Player Score:",
self.players[self.current_player_index]['score'])  # Debug print

        if self.current_roll:
            roll_text = self.current_roll if self.current_roll else 'Hot
dice!'
        else:
            roll_text = 'Click "Roll Dice" to start!'
        player = self.players[self.current_player_index]
        self.status_label.config(
            text=f"{player['name']} has score {self.current_score} and
{player['score']} banked. Dice: {roll_text}"
        )
```

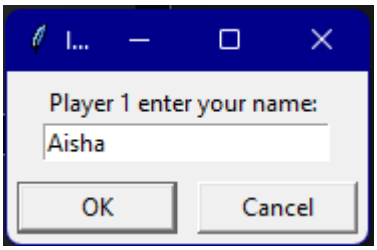
```
    def update_players_label(self):
        """Update the label displaying the players' scores."""
        status = "="*13 + " Status " + "="*14 + "\n"
        for player in self.players:
            status += f"{player['name']} has {player['score']} points.\n"
        self.players_label.config(text=status)

    def roll_dice_action(self):
        """Roll the dice and update the status."""
        self.current_roll = roll_dice(6 if not self.current_roll else
len(self.current_roll))
        self.chosen = False
        self.update_status()
        self.update_combos_listbox()

    def update_combos_listbox(self):
        """Update the listbox with valid scoring combinations."""
        self.combos_listbox.delete(0, tk.END)
        combos = [combo for combo in POINTS if combo in self.current_roll]
        for idx, c in enumerate(combos, 1):
            self.combos_listbox.insert(tk.END, f"({idx}) Remove {c} for
{POINTS[c]} points.")
        if not combos:
            messagebox.showinfo("Farkle", "FARKLED!")
            self.current_score = 0  # Reset score to 0 on Farkle
            self.end_turn()  # End the turn after Farkle

    def end_turn(self):
        """End the current player's turn."""
        self.players[self.current_player_index]['score'] += self.current_score
        self.current_score = 0  # Reset current score to 0 at the end of turn
        self.current_roll = ""  # Reset current roll
        self.chosen = False
        self.current_player_index = (self.current_player_index + 1) %
len(self.players)
        if self.players[self.current_player_index]['score'] >= TARGET_SCORE:
            self.players[self.current_player_index]['done'] = True
            self.last_round = True
        self.update_status()  # Update status after ending turn
        self.update_players_label()
        if all(player['done'] for player in self.players):
            self.end_game()

    def end_game(self):
        """End the game and determine the winner."""
        max_score = max(self.players, key=itemgetter('score'))['score']
        winners = [pl for pl in self.players if pl['score'] == max_score]
```

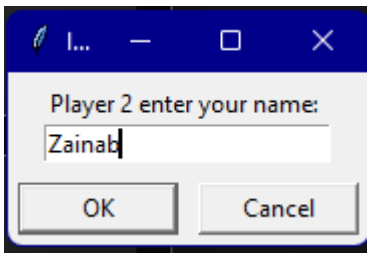
```
        winner_names = ', '.join([winner['name'] for winner in winners])
        messagebox.showinfo("Game Over", f"{winner_names} is the winner with
{max_score} points.")
        self.master.quit()

if __name__ == '__main__':
    root = tk.Tk()
    game = FarkleGame(root)
    root.mainloop()
```

Output Screenshots:

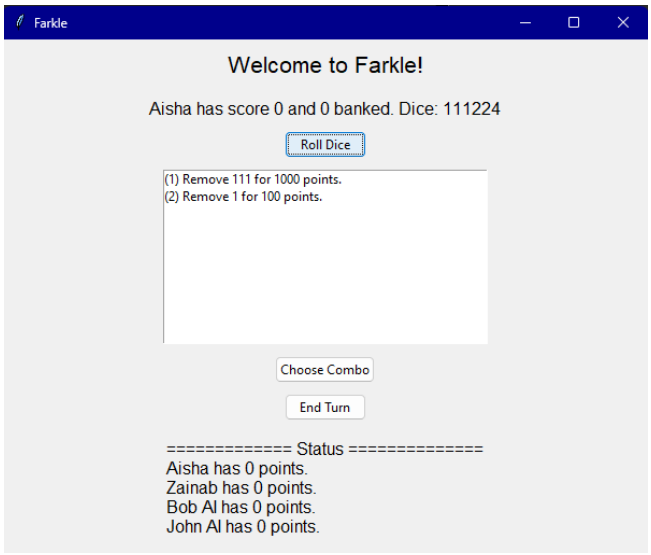


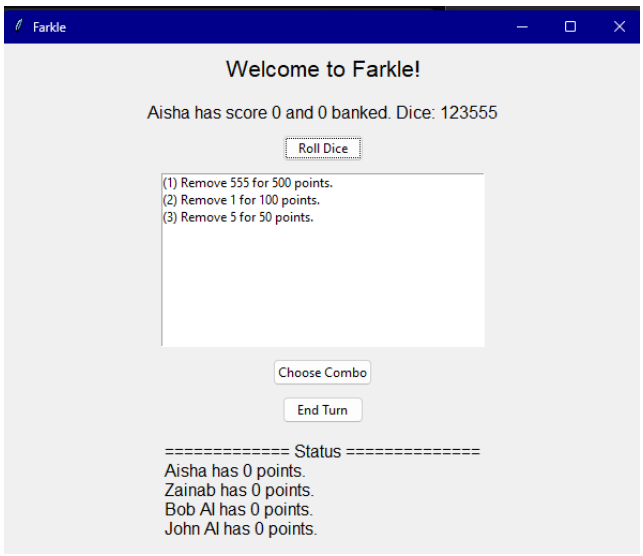


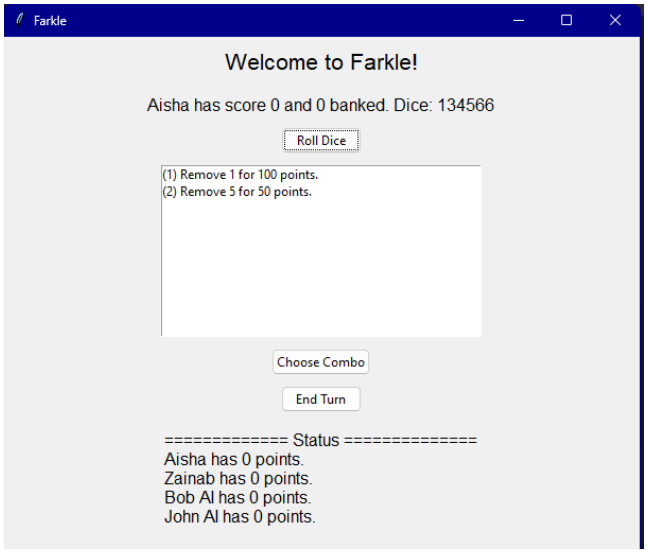


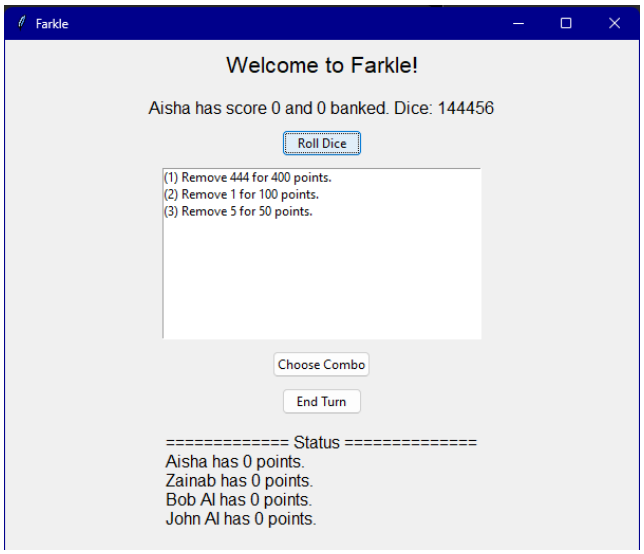



Farkle
Welcome to Farkle!
Aisha has score 0 and 0 banked. Dice: Click "Roll Dice" to start!
Roll Dice
Choose Combo
End Turn
========== Status ==========
Aisha has 0 points.
Zainab has 0 points.
Bob AI has 0 points.
John AI has 0 points.


Farkle
Welcome to Farkle!
Aisha has score 0 and 0 banked. Dice: 111224
Roll Dice
(1) Remove 111 for 1000 points.
(2) Remove 1 for 100 points.
Choose Combo
End Turn
========== Status ==========
Aisha has 0 points.
Zainab has 0 points.
Bob AI has 0 points.
John AI has 0 points.


Farkle
Welcome to Farkle!
Aisha has score 0 and 0 banked. Dice: 123555
Roll Dice
(1) Remove 555 for 500 points.
(2) Remove 1 for 100 points.
(3) Remove 5 for 50 points.
Choose Combo
End Turn
========== Status ==========
Aisha has 0 points.
Zainab has 0 points.
Bob AI has 0 points.
John AI has 0 points.





Declarative Implementation of Farkle in Python:

Implementing the Farkle game in a declarative style involves focusing on the "what" rather than the "how" by using functional programming techniques and maintaining a clean, readable code.

Data Structures and Type Definitions:

To effectively manage game data, the following data structures will be used:

- **Game Board:** The game does not involve a traditional board, but we have a structure to keep track of players and their scores.
- **Player:** Represented as a dictionary with keys for the player's name, score, AI status, and whether they are done with their turn.
- **Dice Roll:** Represented as a string of digits, each representing the outcome of a die roll.

Behaviour Step Implementations:

- **Rolling Dice:** This function simulates rolling a specified number of dice and returns the results as a sorted string.
- **Game Initialization and Player Turn Handling:** Functions to initialize the game with player inputs, handle player turns, validate moves, and update the game state.
- **End Turn and End Game:** Functions to handle the end of a player's turn and to check if the game has ended, declaring the winner.
- **Main Game Logic:** The main function orchestrates the game loop, switching turns between players and checking for a winner after each move.

Areas of Improvement:

- Error handling and input validation could be further refined to enhance user experience.
- Additional features such as AI decision-making logic and a more interactive interface could be developed.

T3 Testing and Verification of Programs via appropriate tools and techniques

---

Test Case ID	Description	Test Steps	Test Data	Expected Results	Actual Results	Pass/Fail
TC1	Verify game initialization and player inputs	1. Run the game. 2. Enter the number of human players. 3. Enter the number of AI players. 4. Enter player names as prompted.	Number of human players, Number of AI players, Player names	Game starts with the specified number of players	As Expected	PASS
TC2	Verify scoring combination selection	1. Roll the dice. 2. Check available scoring combination	Dice roll, Available scoring combinations	Selected scoring combination adds correct	As Expected	PASS

		s. 3. Choose a valid combination.		points to score		
TC3	Verify end of turn functionality	1. Roll the dice. 2. Select scoring combinations until ready to end turn. 3. End turn.	Dice roll, Selected scoring combinations	Player score is updated correctly after ending turn	As Expected	PASS
TC4	Verify Farkle scenario	1. Roll the dice. 2. Choose invalid scoring combinations until a Farkle is encountered.	Dice roll, Invalid scoring combinations	Current score resets to 0 after encountering Farkle	As Expected	PASS
TC5	Verify end of game and winner determination	1. Play the game until a player reaches or exceeds the target score. 2. End the game.	Player scores	Winner is determined correctly based on highest score	As Expected	PASS