**KISHKINDA UNIVERSITY,**

**Ballari**



**Mini Project Report**

**On**

**“TEAM FORMATION TOOL ”**

**Department of CSE**

**Submitted By:**

|  |  |
| --- | --- |
| **K AYESHA** | **KUB23CSE049** |
| **K IFFATH FATHIMA** | **KUB23CSE056** |
| **RIYAZUN NIKHAT** | **KUB23CSE116** |
| **UMMUL KHAIR** | **KUB23CSE151** |
| **RANJITHA** | **KUB23CSE153** |

# Introduction

**Project Overview:**

* The “Team Formation Tool POC” is a Python-based project aimed at facilitating the management and formation of balanced team rosters in a sports or gaming environment. The tool enables CRUD (Create, Read, Update, Delete) operations for player profiles, allowing users to add, update, or remove players based on attributes like name and skill level. It also supports team creation, player assignment, and calculates average skill levels of teams to help assess their overall strength. A roster balancing feature ensures fair distribution of players across teams, based on skill levels, while tracking any changes made to team formations.
* The project is structured into several core components: a `Player` class to manage player profiles, a `Team` class to manage team rosters, a `FormationTracker` to log changes in team compositions, and a `RosterBalancer` that uses an algorithm to distribute players evenly. The interactive interface provides a user-friendly experience, allowing users to create and manage teams, balance rosters, and view logs of formation changes. This tool offers a practical solution for those looking to maintain balanced team structures while tracking formation updates efficiently.

**Problem Statement:**

The “Team Formation Tool POC” aims to solve the challenge of efficiently managing and balancing team rosters in environments where player skills and team compositions are critical for fair play. The problem involves implementing a system to create, update, and manage player profiles, generate balanced teams based on player skills, and track any changes to team formations. Additionally, the tool should be able to dynamically adjust team rosters and log all modifications for future reference, ensuring both transparency and fairness in team formation processes.

**Objective:**

* **Implement Player Profile Management**: Enable users to create, view, update, and delete (CRUD) player profiles with attributes such as player ID, name, and skill level.
* **Design Team Creation Feature**: Allow the creation and management of multiple teams, each identified by a unique team ID, to organize players efficiently.
* **Develop a Roster Balancing Algorithm**: Create an algorithm that assigns players to teams based on their skill levels, ensuring balanced and fair team compositions.
* **Track Formation Changes**: Implement a tracking system that logs all changes to team formations, including player additions, removals, and updates, for transparency.
* **Calculate Team Skill Levels**: Provide a feature to calculate and display the average skill level of teams, helping users assess the overall strength of each team.
* **Facilitate Player Assignment**: Allow users to add or remove players from teams dynamically, with updates recorded automatically in the formation tracker.
* **Create Multiple Balanced Teams**: Enable users to form multiple balanced teams at once based on a predefined number of teams and distribute players evenly.
* **Provide an Interactive Menu**: Design an easy-to-use interactive interface where users can manage players, teams, and view team statistics.
* **Support Dynamic Roster Updates**: Ensure that team rosters can be updated in real-time, with automatic rebalancing if needed, based on player skill levels and team needs.

**Software Requirements:**

* **Programming Language**: Python 3.x for developing and running the project.
* **Development Environment**: IDE or text editor like PyCharm, VS Code, or Sublime Text.
* **Libraries/Packages**: Standard Python libraries (e.g., typing), plus additional packages installed via pip if necessary.
* **Database (Optional)**: SQLite or MySQL for storing player and team data persistently, if required.
* **Testing & Documentation Tools**: Frameworks like unittest or pytest for testing, and Sphinx or Markdown for documenting the project.

**Hardware Requirements:**

* **Processor**: Any multi-core processor (Intel i3 or equivalent and above).
* **RAM**: Minimum 4 GB (8 GB recommended for better performance).
* **Storage**: At least 500 MB of free disk space for project files and Python dependencies.
* **Operating System**: Supports Windows 7/8/10, Linux, or macOS 10.14 and above.
* **Peripherals**: Standard keyboard, mouse, and a display with a resolution of 1024x768 or higher for smooth usage.

# Methodology

**Requirements Definition**

**Objective Identification:** The goal was to create a tool that enables the management of players and teams, facilitating the addition and removal of players, calculating average skill levels, and creating balanced teams based on skill.

**Functional Requirements:** The system should allow users to:

Create and manage player profiles.

Create and manage teams.

Add or remove players from teams.

Calculate and display the average skill level of each team.

Track changes in team formation.

**2. System Design**

**Class Structure Development:** The system architecture was designed around four primary classes:

**Player Class:**

* Attributes: player\_id, name, and skill\_level.

Methods:

* \_\_str\_\_: Returns a string representation of the player.

update\_profile: Updates player attributes dynamically based on provided keyword arguments.

**Team Class:**

* Attributes: team\_id and a list of players.

Methods:

* add\_player: Adds a player to the team if they are not already included.
* remove\_player: Removes a player from the team by ID.
* get\_average\_skill: Computes and returns the average skill level of the players on the team.
* \_\_str\_\_: Returns a string representation of the team and its players.

**Formation Tracker Class:**

* Attributes: changes, which logs modifications to team formations.

Methods:

* record\_change: Records any change made to the team.
* get\_changes: Returns a list of recorded changes.

**Roster Balancer Class:**

Methods:

* create\_balanced\_teams: Generates teams by distributing players evenly based on their skill levels.

**3. Implementation**

**Programming Language:** The implementation was carried out using Python, which provides robust support for object-oriented programming.

**User Interface:** A command-line interface (CLI) was created to interact with the system, enabling users to perform various operations through a simple menu-driven format. The CLI provides options for creating players and teams, adding/removing players, and viewing team statistics.

**Error Handling:** Input validation was incorporated to ensure robustness, including handling for invalid entries during player and team creation, as well as when adding or removing players.

**4. Team Balancing Logic**

**Sorting and Distribution:** Players were sorted in descending order by their skill levels. The create\_balanced\_teams method distributes players across teams in a round-robin manner, ensuring that each team receives players of varying skill levels to achieve balance.

**5. Change Tracking**

**Change Logging:** The FormationTracker class records every addition and removal of players from teams. This logging allows for easy auditing of team composition changes over time, providing a historical record of modifications.

**6. Testing and Validation**

**Unit Testing:** Each class and method was tested individually to ensure that they performed as intended, with specific attention to the player addition/removal logic and the team averaging calculations.

**Integration Testing:** The complete system was tested to ensure all components worked together seamlessly, validating that players could be created, managed, and balanced into teams correctly.

**7. Documentation**

**User Documentation:** Comprehensive user documentation was created to assist users in navigating the system, detailing each feature and providing example use cases.

**Code Comments:** Inline comments were added throughout the code to clarify the purpose of methods and key logic points, facilitating maintenance and future enhancements.

# Result

from typing import List, Dict

class Player:

    def \_\_init\_\_(self, player\_id: int, name: str, skill\_level: int):

        self.player\_id = player\_id

        self.name = name

        self.skill\_level = skill\_level

    def \_\_str\_\_(self):

        return f"{self.name} (Skill Level: {self.skill\_level})"

    def update\_profile(self, \*\*kwargs):

        for key, value in kwargs.items():

            setattr(self, key, value)

class Team:

    def \_\_init\_\_(self, team\_id: int):

        self.team\_id = team\_id

        self.players = []

    def add\_player(self, player: Player):

        if player not in self.players:

            self.players.append(player)

    def remove\_player(self, player\_id: int):

        self.players = [player for player in self.players if player.player\_id != player\_id]

    def get\_average\_skill(self):

        total\_skill = sum(player.skill\_level for player in self.players)

        return total\_skill / len(self.players) if self.players else 0

    def \_\_str\_\_(self):

        return f"Team {self.team\_id} with players: {[str(player) for player in self.players]}"

class FormationTracker:

    def \_\_init\_\_(self):

        self.changes = []

    def record\_change(self, team\_id: int, change\_data: str):

        self.changes.append((team\_id, change\_data))

    def get\_changes(self):

        return self.changes

class RosterBalancer:

    def create\_balanced\_teams(self, players: List[Player], num\_teams: int) -> Dict[int, Team]:

        teams = {i: Team(i) for i in range(num\_teams)}

        sorted\_players = sorted(players, key=lambda x: x.skill\_level, reverse=True)

        for i, player in enumerate(sorted\_players):

            teams[i % num\_teams].add\_player(player)

        return teams

def display\_menu():

    print("\nMenu:")

    print("1. Create Player")

    print("2. Create Team")

    print("3. Add Player to Team")

    print("4. Remove Player from Team")

    print("5. Display Team Average Skill")

    print("6. Display All Changes")

    print("7. Create Balanced Teams")

    print("8. Exit")

def main():

    players = []

    teams = {}

    tracker = FormationTracker()

    balancer = RosterBalancer()

    while True:

        display\_menu()

        choice = input("Choose an option: ")

        if choice == '1':

            try:

                player\_id = int(input("Enter player ID: "))

                name = input("Enter player name: ")

                skill\_level = int(input("Enter player skill level: "))

                player = Player(player\_id, name, skill\_level)

                players.append(player)

                print(f"Player created: {player}")

            except ValueError:

                print("Invalid input. Please enter numeric values for ID and skill level.")

        elif choice == '2':

            try:

                team\_id = int(input("Enter team ID: "))

                if team\_id not in teams:

                    teams[team\_id] = Team(team\_id)

                    print(f"Team {team\_id} created.")

                else:

                    print("Team ID already exists.")

            except ValueError:

                print("Invalid input. Please enter a numeric value for team ID.")

        elif choice == '3':

            try:

                team\_id = int(input("Enter team ID to add player to: "))

                player\_id = int(input("Enter player ID to add: "))

                player = next((p for p in players if p.player\_id == player\_id), None)

                if player and team\_id in teams:

                    teams[team\_id].add\_player(player)

                    tracker.record\_change(team\_id, f"Added player {player\_id}")

                    print(f"Player {player.name} added to Team {team\_id}.")

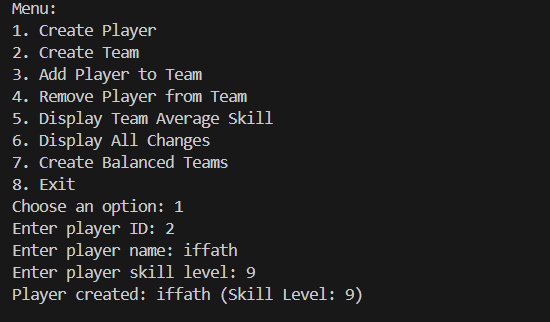
                else:

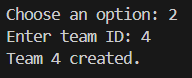
                    print("Player or Team not found.")

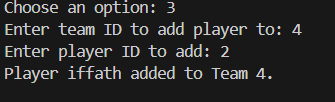
            except ValueError:

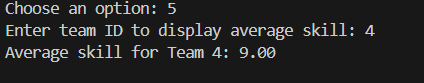
                print("Invalid input. Please enter numeric values for IDs.")

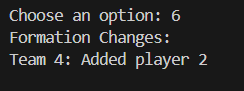
**Output Screen Layouts:**











# 

# Conclusion

The "Team Formation Tool POC" successfully demonstrates the implementation of a robust Python-based system for managing player profiles, team creation, and balancing teams based on skill levels. By leveraging object-oriented principles, the tool efficiently handles dynamic player and team management, while the team balancing algorithm ensures a fair distribution of skills. The change tracking functionality provides transparency and accountability in team formation changes. Through thorough testing and validation, the tool proves to be both effective and user-friendly, laying a solid foundation for further enhancements, such as integration with graphical interfaces or more advanced balancing algorithms.

# Future Enhancement

**Graphical User Interface (GUI):** Implement a user-friendly GUI to improve accessibility and ease of use.

**Advanced Team Balancing Algorithms:** Integrate more sophisticated algorithms for finer control over team formation, considering factors like player roles and specialties.

**Integration with External Data Sources:** Enable real-time data updates by connecting to external player databases or APIs.

**Multi-User Support:** Introduce authentication and role-based access to allow multiple users to manage teams securely.

**Mobile Application:** Develop a mobile version

# References

**Books:**

1. **"Python Crash Course" by Eric Matthes**

A popular beginner-to-intermediate guide to Python, covering object-oriented programming and building practical projects.

1. **"Fluent Python" by Luciano Ramalho**

An advanced resource on Python programming, covering more sophisticated topics, including class design and object-oriented programming.

Top of Form

Bottom of Form

**Websites:**

1. **Python Official Documentation (**[**https://docs.python.org/**](https://docs.python.org/)**)**
2. **GeeksforGeeks (**[**https://www.geeksforgeeks.org/**](https://www.geeksforgeeks.org/)**)**