Football Game Simulator – Project 1

"Design the Game. Code Your Imagination!"

Project Objective

You will build a console-based football game simulator in Dart. This project will help you master Object-Oriented Programming (OOP) by modeling a football match, designing your own classes and actors, and using game theory to simulate real strategic competition. You are invited to add as many creative features, actors, or rules as you wish!

Project Requirements

1. Build the Core Classes (Actors)

For each class, include all required properties, and add any additional or creative properties or methods you invent:

Player:

- Required: name, position, id, power (random)
- Open: (e.g., age, nationality, stamina, skills, jersey number, etc.—add your own!)

Trainer:

- Required: name, experience (random)
- Open: (e.g., experience, strategy specialty, motivation, etc.—add your own!)

Team:

- Required: team name, trainer, list of players
- Open: (e.g., mascot, city, team colors, motto, etc.—add your own!)

Stadium:

- Required: name, location
- Open: (e.g., capacity, weather, altitude, turf type, etc.—add your own!)

Game:

- Manages team creation, stadium setup, simulation logic, and match outcome.
- Open: (Add your own methods, helper classes, or match events.)

Other Actor(s) for Innovation:

- Invent and implement at least one new actor or system of your own!
- Collect input for your new actor/system if needed.
  Integrate it into the simulation and show its effect in the output.

2. Enable Strategic Decision-Making (Game Theory)

Before the match, each trainer chooses a strategy (e.g., Offensive, Defensive, Balanced, Counter-Attack, Pressing, Park-the-Bus, etc.—add your own if you wish!).
Strategies interact based on a payoff matrix:
- Each strategy gains or loses effectiveness depending on the opponent's choice.
- Use a function or matrix to look up the bonus for each team based on the chosen strategies.
Example: Offensive vs Defensive: Offensive gets +5, Defensive gets +10.
Balanced vs Balanced: both get +7.
Design and balance your own matrix!

3. Create Teams Player

For each team:
- Enter team name and any additional property.
- Enter trainer name and any additional property.
- Choose a team strategy (prompt user).
- Enter the number of players.
- For each player, input all required and extra properties.

4. Create the Stadium

Input stadium name, location, and any extra properties (capacity, weather, etc.).

5. Invent Your Own Actor, System, or Feature

You must invent and add at least one new actor, system, or game feature (referee, fans, weather system, match event, commentator, media, etc.).
Collect input for your new actor/system if needed.
Integrate it into the simulation and show its effect in the output.

6. Simulate the Game (Game Theory in Action!)

For each team, calculate total power as:
sum of player powers + trainer power + strategy bonus (from matrix) + random factor + (any extra bonuses from your innovations)
Use your invented actors/properties to influence the game outcome.
Print all teams, strategies, powers, stadium info, match events, and the winner.
Add any creative outputs: commentary, simulated goals, cards, injuries, etc.

7. Creativity is Open!

There is no limit to how many extra properties, actors, rules, or features you invent.
At the top of your code, briefly describe any new actors, systems, or logic you have created.

Submission Checklist

- Dart code file (.dart) with all classes and properties (required + extras).
- At least one invented actor, system, or feature, described at the top.
- User input for all properties and game decisions (including strategy selection).
- Console output showing all teams, strategies, actors, events, and the winner.
- Sample output included as a comment.
- Bonus: add any creative features or innovations for extra credit!

Evaluation Criteria

- OOP design: clear classes, relationships, encapsulation, and extensibility.
- Game theory: strategic choices and a working payoff matrix.
- Input/output: interactive and complete.
- Simulation logic: creativity and fairness.
- Integration: use of your invented actor/system and any other open features.
- Code readability and comments.

Note:
If you do not fully understand any part of the project requirements or have difficulty implementing a specific instruction, you are allowed to create your own mini-project idea instead.
Your custom project should still follow the main principles of object-oriented programming, demonstrate your creativity in class design and properties, and show meaningful output in the console.

Design the Game ..... Code Your Imagination!

بالتوفيق