UEFA Champions League Simulation Design Document

Allen Roman
CS 4632 W01

March 2025

1 Introduction

This document outlines the design of a comprehensive UEFA Champions League simulation system. The system replicates football matches with high fidelity using discrete-event simulation and probability models. The simulation operates at a second-by-second resolution, modeling individual player actions, ball physics, and match events in detail.

2 System Architecture

The simulation adopts an object-oriented architecture with clear separation of concerns. Figure 1 shows the high-level design, consisting of four main modules: State Representation, Simulation Engine, Action Outcome System, and Player Decision Framework.

2.1 State Representation Module

The state representation module contains classes that define and maintain the current state of all entities in the simulation. These classes serve as the data model for the entire system.

2.1.1 Ball Class

The Ball class encapsulates all aspects of the match ball:

- **Physical State:** Position coordinates (x,y), height above ground, velocity vector, and speed.
- **Status Attributes:** Current physical state (on ground, in air, stationary), action being performed (passed, shot, loose), and field zone location.
- **Possession Information:** References to the team and player currently in possession.

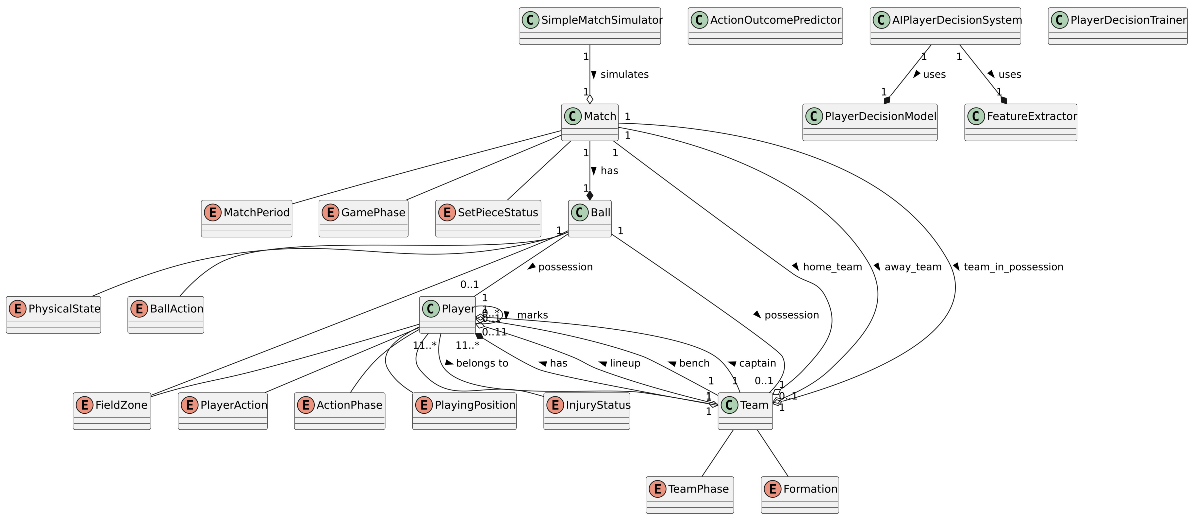

Figure 1: Simple UML Diagram of the System Architecture

- **Behavior:** Updates its position based on velocity and time step, maintains field zone awareness, and synchronizes position with possessing player.

2.1.2 Player Class

The Player class represents individual football players, the primary actors in the simulation:

- **Identity:** Player ID, name, assigned position, and team affiliation.
- **Attributes:** Performance capabilities (passing, shooting, defending, stamina, etc.) that affect action success rates.
- **Physical State:** Position, orientation, velocity, speed, and acceleration on the pitch.
- **Action State:** Current action being performed, action phase (starting, executing, finishing), action target, and availability for new actions.
- **Physiological State:** Current stamina, fatigue level, injury status, and sprint availability.
- **Behavior:** Updates position, maintains stamina, performs actions, and tracks distance to the ball.

2.1.3 Team Class

The Team class manages groups of players and team-level information:

- **Identity:** Team ID and name.
- **Match State:** Possession status, goals scored, and goals conceded.
- **Player Management:** Collections of all players, active lineup, and substitutes.
- **Tactical Setup:** Formation, phase of play, and tactical parameters (pressing intensity, defensive line height, width, tempo, etc.).
- **Behavior:** Facilitates player substitutions, changes formations, and updates tactical parameters.

2.1.4 Match Class

The Match class serves as the container for the entire simulation:

- **Core Components:** References to the competing teams and the ball.
- **Match State:** Current clock time, match period, game phase, and set piece status.
- **Possession Tracking:** Reference to the team currently in possession.
- **Match History:** Chronological record of all significant events.
- **Behavior:** Advances the match clock, facilitates possession changes, and logs match events.

2.1.5 Enumeration Types

The system uses several enumeration types to standardize categorical states:

- **PhysicalState:** Defines ball states like ON_GROUND, IN_AIR, STATIONARY.
- **BallAction:** Represents what's happening with the ball (PASSED, SHOT, LOOSE, etc.).
- **FieldZone:** Divides the pitch into tactical zones (DEFENSIVE_THIRD, LEFT_WING, etc.).
- **PlayerAction:** Comprehensive set of actions a player can perform (PASS, TACKLE, PROVIDE_SUPPORT, etc.).
- **ActionPhase:** Represents the progression of an action (STARTING, EXECUTING, FINISHING).
- **PlayingPosition:** Standard football positions (GK, CB, CM, ST, etc.).
- **TeamPhase:** Team tactical phases (ATTACKING, DEFENDING, TRANSITION).

- **MatchPeriod:** Match time divisions (FIRST_HALF, HALFTIME, etc.).
- **GamePhase:** Current state of play (OPEN_PLAY, SET_PIECE, KICKOFF, etc.).

2.2 Simulation Engine Module

2.2.1 SimpleMatchSimulator Class

This class drives the entire simulation using SimPy's discrete-event framework:

- **Initialization:** Sets up the match with teams, players, and initial conditions.
- **Time Management:** Uses SimPy environment to advance time in one-second increments.
- **Process Coordination:** Manages the main match process and coordinates player processes.
- **State Updates:** Updates all player positions, actions, and stamina levels each second.
- **Ball Updates:** Manages ball movement and applies simplified physics.
- **Event Processing:** Detects and handles events like goals, fouls, and out-of-bounds.
- **Period Transitions:** Handles halftime, kickoffs, and end-of-match conditions.

The simulator orchestrates all simulation components, ensuring that events occur in the correct sequence and that state updates are properly synchronized across all entities.

2.3 Action Outcome System Module

2.3.1 ActionOutcomePredictor Class

This component acts as the physics and probability engine that determines the results of player actions:

- **Base Probabilities:** Defines baseline success rates for different actions (passes, shots, tackles, etc.).
- **Contextual Adjustment:** Modifies success probabilities based on:
  - Player attributes (passing skill, shooting accuracy, etc.)
  - Context factors (distance, pressure from opponents, angles)
  - Physical condition (player stamina and fatigue)

- **Specialized Predictors:** Contains algorithms for specific actions:
  - Pass outcome prediction (success, interception, misplaced)
  - Shot outcome prediction (on target, saved, goal, wide)
  - Tackle outcome prediction (successful, foul, missed)
  - Dribble outcome prediction (successful, dispossessed, lost control)
- **Success Determination:** Uses weighted random chance to determine outcomes while respecting the calculated probabilities.

This class adds realism by ensuring that actions have appropriate success rates based on player skills and match context, rather than being purely random or deterministic.

2.4 Player Decision Framework Module

2.4.1 PlayerDecisionModel Class

This component determines what actions players should take:

- **Context Analysis:** Evaluates the current match state from a player's perspective.
- **Action Selection:** Chooses appropriate action based on player position, ball proximity, team possession, and tactics.
- **Decision Rules:** Currently uses a simple rule-based approach with conditional logic:
  - If player has ball: choose between pass, shoot, dribble based on position
  - If team has possession but player doesn't have ball: provide support, make runs
  - If defending: press, mark, or cover space based on proximity to ball
- **Target Selection:** For actions requiring a target (passing, marking), identifies appropriate targets based on position and context.

This module is currently implemented with straightforward decision rules but provides the structure for more sophisticated decision-making in the future.

2.4.2 AIPlayerDecisionSystem Class

This higher-level class integrates decision-making components:

- **State Tracking:** Maintains a short history of recent player states to provide temporal context.
- **Decision Coordination:** Ensures players make decisions that are tactically coherent as a team.

- **Interface:** Provides a clean interface between the simulation engine and the decision models.

3 Module Interactions

The system modules interact in specific ways to create a cohesive simulation:

3.1 State Updates Cycle

The core interaction cycle occurs each simulation second:

1. **SimpleMatchSimulator** advances the clock by one second
2. **Match** time is updated and period transitions are checked
3. For each **Player**:
    - Position and physical state are updated based on current velocity and actions
    - Stamina is reduced based on activity level
    - Distance to ball is recalculated
    - If available for action, new action is determined through **PlayerDecisionModel**
    - Action is initiated if applicable
4. The **Ball** position and state are updated
5. Events (goals, out of bounds) are processed and logged

3.2 Action Execution Flow

When a player performs an action:

1. **Player.start_action()** is called with the chosen action type and target
2. Player's action state is updated (current_action, action_phase, action_target)
3. Player's available_for_action flag is set to false
4. **ActionOutcomePredictor** calculates success probability and determines outcome
5. Results are applied to game state:
    - For passes: ball possession may transfer to another player
    - For shots: goals may be scored
    - For tackles: possession may change or fouls may occur
6. After action completes, player becomes available for next action

3.3 Ball Possession Management

Ball possession changes are coordinated across multiple objects:

1. **Match.switch_possession()** is called with new team and player
2. The target team's possession flag is set to true
3. The other team's possession flag is set to false
4. **Ball.set_possession()** updates ball's team and player references
5. Ball position is synchronized with possessing player
6. Player's has_ball flag is set to true

3.4 Event Processing

When significant events occur:

1. Event is detected (e.g., ball crossing goal line)
2. **Match.record_event()** creates an event record with time, type, players involved
3. Appropriate state changes are made (e.g., incrementing goal count)
4. For major events like goals, match state is reset (e.g., ball to center, kick-off)

4 Design Rationale

4.1 Discrete-Event Simulation Approach

The discrete-event simulation paradigm was chosen for its ability to model complex systems with temporal dependencies. Football matches have natural discrete events (passes, shots, tackles) that can be modeled as occurring at specific time points, with the system state evolving between these events.

SimPy was selected as the simulation framework because:

- It provides robust tools for managing the simulation timeline
- It supports process-based modeling that maps well to player behaviors
- It can coordinate concurrent processes, similar to how multiple players act simultaneously in a real match

4.2 Comprehensive State Tracking

The design emphasizes detailed state tracking across all entities. This approach:

- Enables realistic simulation of complex interactions between players and the ball
- Provides rich contextual information for decision-making algorithms
- Supports post-simulation analysis and statistics generation

4.3 Separation of Concerns

The architecture follows a clear separation of concerns:

- State classes (Ball, Player, Team, Match) focus exclusively on representing information
- Simulation logic is contained in the SimpleMatchSimulator class
- Action outcome determination is encapsulated in the ActionOutcomePredictor
- Decision-making is isolated in the Player Decision Framework

This separation enables modular development and testing, and also allows for enhancements to any component without affecting others.

4.4 Probabilistic Outcome Model

The use of probability-based outcome determination for player actions reflects the inherent uncertainty in football:

- Even highly skilled players occasionally fail at simple tasks
- Context significantly affects success rates
- Physical condition impacts performance
- Weighted randomization creates natural variability in outcomes

This approach creates more realistic and engaging simulations than either purely random or purely deterministic models.

5 Conclusion

The UEFA Champions League simulation design represents a comprehensive approach to modeling football matches through discrete-event simulation. By dividing the system into state representation, simulation engine, action outcome, and player decision modules, the architecture achieves both realism and maintainability.

The detailed tracking of states across all entities, combined with the probabilistic action outcome system, enables the simulation to capture the complex and dynamic nature of football. The second-by-second advancement of match time allows for precise modeling of player actions and interactions, creating a foundation for realistic match simulation.