

# EX7 - AI Agent League System

## Course Submission

### 1. Group Information

**Group Name:** eldad\_ron\_bar\_yacobi  
**Member 1:** Eldad Ron (ID: 207021916)  
**Member 2:** Bar Yacobi (ID: 315471367)

### 2. Repository

#### GitHub URL:

<https://github.com/er1009/LLMs-And-Multi-Agent-Orchestration-Course/tree/main/ex7>

### 3. Self-Recommended Grade

**Grade:** 100/100

### 4. Justification

This project implements a complete Multi-Agent System for an AI Agent League, demonstrating advanced concepts in agent communication, protocol design, and distributed systems architecture. The implementation meets the highest quality standards for the following reasons:

- **Complete Multi-Agent Architecture:** The system implements three distinct agent types (League Manager, Referee, Player) that operate as independent FastAPI processes. Each agent maintains its own state and communicates via the standardized MCP protocol, demonstrating true distributed system design principles.
- **Protocol Compliance:** Full implementation of JSON-RPC 2.0 over HTTP with MCP message envelopes. The protocol includes proper authentication via tokens, standardized error codes, conversation tracking, and timestamp validation. All messages follow the league.v2 protocol spec.
- **Game Implementation:** The Even/Odd game is fully implemented with random number drawing, parity checking, winner determination, and score calculation. The round-robin scheduler ensures fair matchups where every player faces every other player exactly once.
- **Extensibility Through Design Patterns:** The Strategy pattern enables seven different player strategies (random, always\_even, always\_odd, alternating, biased\_even, biased\_odd, counter). New strategies can be added without modifying existing code. The architecture supports adding new game types through configuration.

- **Production-Grade SDK:** The shared league\_sdk provides reusable components including Pydantic models for type-safe message validation, an HTTP client wrapper, JSON-lines structured logging, and configuration loaders. All components use modern Python features (type hints, dataclasses, enums).
- **Comprehensive Documentation:** The project includes a detailed PRD with user stories, functional requirements, and acceptance criteria. The Architecture document provides C4 diagrams, technology justifications, and Architecture Decision Records (ADRs).
- **Testing & Quality:** Unit tests cover game logic, Pydantic models, player strategies, and the scheduler. The codebase follows SOLID principles with clear separation of concerns between handlers, state management, and business logic.

## 5. Special Notes

- The orchestration script (run\_league.py) provides a one-command experience to run a complete league.
- All agents auto-register on startup and handle communication asynchronously.
- Match results and standings are persisted to JSON files in SHARED/data/.
- Structured logs in JSON-lines format enable easy parsing and analysis.

## 6. Special Documents

- docs/PRD.md - Product Requirements Document with user stories and acceptance criteria
- docs/ARCHITECTURE.md - Architecture document with C4 diagrams and ADRs
- README.md - Quick start guide and project overview

## 7. Additional Comments

This project demonstrates a comprehensive understanding of multi-agent systems, protocol design, and software engineering best practices. The modular architecture allows for easy extension with new game types, additional agents, or alternative communication protocols. The implementation balances simplicity with production-readiness, making it suitable for both educational purposes and real-world applications.