

Associate Game Developer Test

Multiplayer State Synchronization

Objective

Develop a simple real-time multiplayer **Coin Collector** game that synchronizes player movement and game state through a central server. Candidates must not rely on engines or frameworks that automatically perform networking or state syncing.

Technology Requirements

- **Permitted languages:** Any (C#, C/C++, JavaScript / Node.js, Python, Go, etc.)
- **Networking constraint:**
 - You may use Raw TCP/UDP sockets, WebSockets, or gRPC
- **Not Allowed:**
 - Networking middleware that auto-syncs state (e.g., Photon, Netcode for GameObjects, Mirror, etc.)

Architecture

The submission must consist of two distinct, running components:

- A central, authoritative **Game Server**.
- One or more connecting **Clients** (players).

Part A: Game Requirements

1. **Lobby:** Two clients must connect to a game server. The server can wait for input or auto-start the game session.
2. **Gameplay:**
 - Each player controls a basic shape (cube, circle, etc.).
 - Coins spawn at random map positions every few seconds.
 - When a player touches a coin, it disappears and that player's score increases.
3. **Server Authority:**
 - The server is authoritative for player positions, coin positions, and scoring.
 - Clients may only send intent or input (e.g., "move left").
 - The server resolves collisions and validates score events.

Part B : Network Quality Simulation

You must simulate degraded network conditions:

- Introduce **~200ms latency** to all data sent and received.

This constraint is critical : the test is not merely functional, but resilience-oriented.

Part C : Evaluation Criteria

1. Smoothness (Interpolation):

Since positional data arrives at delayed intervals, naive rendering causes visible stutter. Candidates must implement entity interpolation such that remote players move smoothly on-screen.

2. Security (Server Authority):

- Clients cannot self-report coin pickups or spoof movement.
- The server must validate proximity and maintain the canonical game state.

Required Output

1. GitHub Repository

- Clean, readable code
- Run instructions and any other details in the README

2. Unlisted Video Demonstration

- Show two client screens simultaneously
- Demonstrate smooth motion (despite 200ms latency)
- Demonstrate correct authoritative scoring

Scoring

We will be valuing the quality of the end result along with effort and understanding. Just ensure the links are accessible and instructions are followed exactly.

When in doubt, make a reasonable assumption and if you feel like it, call it out in the Readme.