Project Overview

For this technical test, I implemented a **local split-screen mini football prototype** using **Unreal Engine 5**, built entirely with **Blueprints**. The focus was on creating a functional, readable prototype within scope rather than a fully production-ready system.

The game supports two local players, a shared physics-driven ball, goals on opposite ends of the field, and a scoreboard UI. Players can move, interact with the ball, and temporarily enter a “control mode” that allows closer ball control.

Approach and Reasoning

I scoped the project around **local split-screen gameplay** to keep input handling, player ownership, and camera behavior straightforward while still demonstrating multi-player logic.

A core design decision was making the **ball the authority** over interaction and possession. Instead of each player deciding whether they control the ball, the ball determines:

- Which player is closest
- Whether control can be granted
- When control should end

This reduced conflicting ownership logic and helped centralize decision-making in one place.

The overall structure is modular:

- **BP_Player** handles input, movement, and control requests
- **BP_Ball** owns possession logic and control timing
- **GameMode / GameState** handle match rules and scoring
- **WBP_Scoreboard** reflects game state for both players

C++ vs Blueprint Choices

This prototype was implemented **entirely in Blueprints**.

Why Blueprints

- Faster iteration for gameplay and physics tuning
- Visual debugging was useful for timing, ownership, and state changes
- Ideal for a prototype with evolving rules

What I would move to C++

- Ball ownership and control state machine
- Cooldown and duration timers
- Physics interaction logic
- Replication logic if expanded to online multiplayer

Blueprints worked well for prototyping, but for scalability, determinism, and performance, a C++ implementation would be preferable long-term.

Key Features Implemented

Ball-Driven Ownership System

The most important feature is the **ball-centric ownership model**:

- The ball tracks nearby players using an overlap sphere
- The closest valid player is selected as the control owner
- Control duration and cooldown are enforced per player
- Only one player can control the ball at a time

This avoids race conditions and ensures consistent behavior in split-screen.

Control Duration and Cooldowns

Each player has:

- A limited control duration

- An individual cooldown before re-entering control mode

This keeps gameplay balanced and prevents one player from dominating possession.

Physics-Driven Ball Interaction

The ball uses Chaos physics with manual impulse application for responsiveness. Movement, collisions, and dribbling behavior were tuned to feel controllable without being fully "locked" to the player.

Tradeoffs and Shortcuts

- **Blueprint-only implementation** instead of C++
- Local split-screen only, no networking
- Simplified AI and no goalkeeper logic
- Physics values tuned manually rather than using a full custom movement model
- UI updates driven by state polling rather than event-driven architecture

These were intentional tradeoffs to stay within scope and timeline.

Challenges and Key Learnings

- **Chaos Physics in UE5** behaves differently than UE4. Ball collision and response initially felt inconsistent and required tuning of damping, friction, sleep thresholds, and impulse strength.
- **Local multiplayer setup** introduced edge cases. Creating/initializing the second local player and ensuring correct input ownership (Enhanced Input / local player subsystem) required debugging and a workaround to guarantee the correct local player/controller was used.
- Separating "wants control" vs "has control" was important to avoid state bugs.
- Centralizing ownership logic in the ball simplified edge cases compared to distributing possession logic across players.

Time Spent

Approximately **10 hours**, including:

- Core gameplay setup
- Physics tuning
- Control logic iteration
- UI integration
- Debugging ownership and timing issues

What I Would Improve With More Time

- Rewrite core systems in **C++**
- Replace polling logic with event-driven updates
- Add visual indicators for ball ownership and cooldowns
- Improve ball control using predictive physics or assist curves
- Expand to networked multiplayer
- Add animations and feedback for goals and control transitions