Project Overview

Built Using UE 5.6

This technical test implements a local split-screen mini football prototype in Unreal Engine 5, built using Blueprints. The goal was to deliver 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 enables closer ball handling. The prototype follows the required WASD scope, and for ease of local testing I also mapped a controller to the second player to quickly validate two-player interactions and input ownership.

A third-person character setup was used as a shortcut to leverage built-in movement and camera behavior, with input overrides applied where needed for gameplay-specific interactions.

Controls and Input Mapping

Keyboard and Mouse

- **WASD:** Move
- **Space:** Jump
- **C:** Enter control mode (subject to duration and cooldown)
- **Left Mouse Button:** Flat shot or pass
- **Right Mouse Button:** Power arc shot
    - Shot strength scales with hold duration, and the arc increases for longer holds

Controller (used for Player 2 testing convenience)

- **Left Stick:** Move
- **Right Stick:** Camera control and aim
- **A:** Jump
- **Y:** Enter control mode
- **X:** Flat shot or pass
- **B:** Power arc shot
    - Strength scales with hold duration, producing a higher and stronger arc the longer the button is held

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 multiplayer logic.

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

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

This reduced conflicting ownership logic and centralized decision-making in a single system.

The overall structure is modular:

- **BP_Player** handles input, movement, aiming, and control requests
- **BP_Ball** owns possession logic, control duration, and cooldown enforcement
- **GameMode / GameState** manage match rules and scoring
- **WBP_Scoreboard** reflects shared game state for both players

C++ vs Blueprint Choices

This prototype was implemented entirely in Blueprints to prioritize speed of iteration and clarity during development.

Why Blueprints Were Sufficient for the Prototype

- Faster iteration for gameplay and physics tuning
- Visual debugging was valuable for ownership timing and state transitions
- Suitable for validating core gameplay rules within a short timeline

Blueprints are not optimized for complex state management or performance-critical systems, but they were sufficient for demonstrating the intended architecture and gameplay behavior at a prototype level.

What I Would Move to C++ and Why

For scalability, determinism, and long-term maintainability, I would prefer to implement the following systems in C++:

- Ball ownership and control state machine, using explicit state objects with constructors and destructors to clearly define ownership lifetimes
- Cooldown and duration logic, enabling more precise control and easier balancing through data-driven parameters
- Physics interaction and shot mechanics, for better control over impulses, charge curves, and collision response

- Critical ownership transitions, guarded using lightweight synchronization or critical-section style logic to prevent conflicting state changes in edge cases
- Replication logic, if expanded to online multiplayer

A C++ implementation would make balancing easier, improve performance, and allow clearer ownership guarantees compared to Blueprint-based state handling.

Key Features Implemented

Ball-Driven Ownership System

- 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

Control Duration and Cooldowns

Each player has:

- A limited control duration while holding the ball
- An individual cooldown before control can be reactivated

This balancing prevents one player from dominating possession and encourages active contesting.

Physics-Driven Ball Interaction and Shooting

The ball uses Chaos physics with manual impulse application for responsiveness. Shot mechanics support both a flat pass/shot and a charged arc shot, with charge duration mapped to impulse strength.

Tradeoffs and Shortcuts

The following tradeoffs were intentional to stay within scope:

- Blueprint-only implementation instead of C++
- Local split-screen only, no networking
- Third-person character template used to accelerate movement and camera setup
- Simplified physics tuning without a custom movement or prediction model
- UI updates driven by polling rather than event-driven architecture

Challenges and Key Learnings

- Chaos Physics in UE5 behaves differently than UE4 and required tuning damping, friction, sleep thresholds, and impulse strength.
- Local multiplayer introduced input ownership edge cases. Ensuring the correct local player and Enhanced Input subsystem were used required debugging and workarounds.
- Separating “wants control” from “has control” was essential to prevent state conflicts.
- Centralizing ownership logic in the ball simplified edge cases compared to distributing possession logic across player actors.

Time Spent

Approximately 15 hours, including:

- Core gameplay setup
- Physics tuning
- Control logic iteration

- UI integration
- Debugging ownership, input, and timing issues

What I Would Improve With More Time

- Rewrite core systems in C++
- Fine-tune gameplay balance for cooldowns, control duration, and shot charge curves
- Replace polling-based UI updates with event-driven updates
- Add clearer visual feedback for control state and cooldown timers
- Improve ball control using predictive physics or assist curves
- Expand to networked multiplayer
- Add animations and feedback for goals and control transitions