Mini Football NGC Interview

Introduction:

Mini football is a technical interview prototype for **Numbered Gaming Co** which relies on physics based objects and player controls to get goals on either side. The game focuses on multiplayer 1v1 scenarios and precise physics collision between the players, world objects and the ball.

Approach and Reasoning:

Used **Unreal Engine 5.6** and built the prototype around a clear server decides, clients display flow so the core gameplay stays consistent when you add replication. I split responsibilities by Unreal patterns:

- **GameMode** for match rules (timer, win condition, spawn rules).
- **GameState** for replicated match data (scores, time remaining, match state).
- **Actors** like Ball / Goal / OutOfBounds as sensors + visuals, with authority checks to ensure only the server drives gameplay.
- Used **Interfaces** to avoid hard casting chains (e.g., ball reacts to `OnScored`, `OnOutOfBounds`, `ApplyKick`) and kept “who owns what” clear to prevent RPC/ownership issues.
- Used **Components** to split functionality from Actors and made it reusable for future actors
- Used **Structs** to split default variables from the Actors to have robust control over values.

Blueprint vs C++:

Blueprints: Used for rapid iteration on gameplay and visuals (goal overlap, Niagara, material changes, UI widgets,Shaders). Blueprints are faster for tuning feel (kick power, Timeline animations, VFX timing, Replication) and are easy to debug with prints and visual graphs.

C++ (if/when scaling):

- Replicated gameplay frameworks (custom GameState/PlayerState logic, score/time replication, match flow) would be cleaner and safer in C++.
- More complex mechanics like dribble forces, ball control, prediction/reconciliation, and custom movement would benefit from C++ performance and maintainability.
- C++ is also better for large-scale refactors (components, replicated properties with RepNotify, clean RPC definitions).

Tradeoffs and Shortcuts Taken:

- Focused on the simplest reliable replication model: **server-authoritative physics** for the ball; clients just receive replicated movement.
- Used straightforward event/RPC patterns over deeper systems (no client-side prediction yet).
- For goal color randomness, used **server picks once → multicast/replicate value** rather than deterministic seeded random across machines.
- Spawning logic initially used a join counter; later recognized a more robust approach is assigning starts deterministically using Tags and Overriding `ChoosePlayerStart`.
- Used some variables within the actors to make the RPC calls more reliable when initializing Replication.

Time Taken:

- Player Controls, Ball Mechanics and Standalone Project development took around **2 to 3 hours** in total.
- Creating VFX for the game took around **3 to 4 hours**.
- When Started Replication building a solid Remote Procedure Calls structure to have a reliable connection between server and client took around **10 to 15 hours** in total.
- Creating UI and adding replication took **2 to 3 hours** in total.
- Building a basic level took around **1 to 2 hours**.

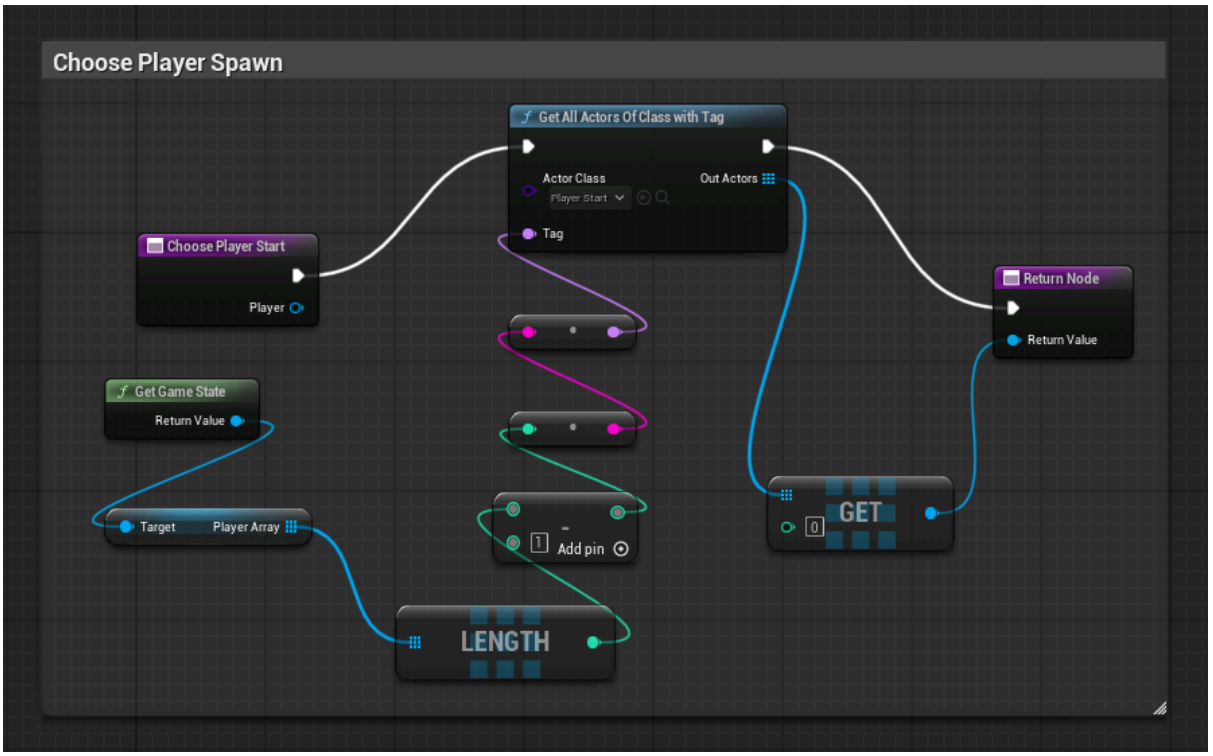
- Abstracting blueprint and edge case tests took around **3 to 5 hours**.

Improvements with more time

- **Prediction & smoothing** for the ball to reduce jitter (better network smoothing, substepping tuning, possibly client prediction + server correction).
- Move gameplay logic into a more **component-based architecture** cleanly:
    - BallPhysicsComponent, GoalLogicComponent, OutOfBoundsComponent, MatchRulesComponent, PlayerInteractionComponent.
- Add proper match state handling:
    - Kickoff, Celebration freeze, Resume play, Overtime, etc.
- Better UI sync:
    - RepNotify-driven updates, late-join support, consistent UI teardown on restart.
- Add **validation/security** on server:
    - range checks for kicks, dribble ownership/lock, cooldowns, power ups.
- More immersive **level design** with proper assets and more realistic animations for Player character and Goal post.
- Improve the Data handling using **Data assets and structs** together for variables and inputs.

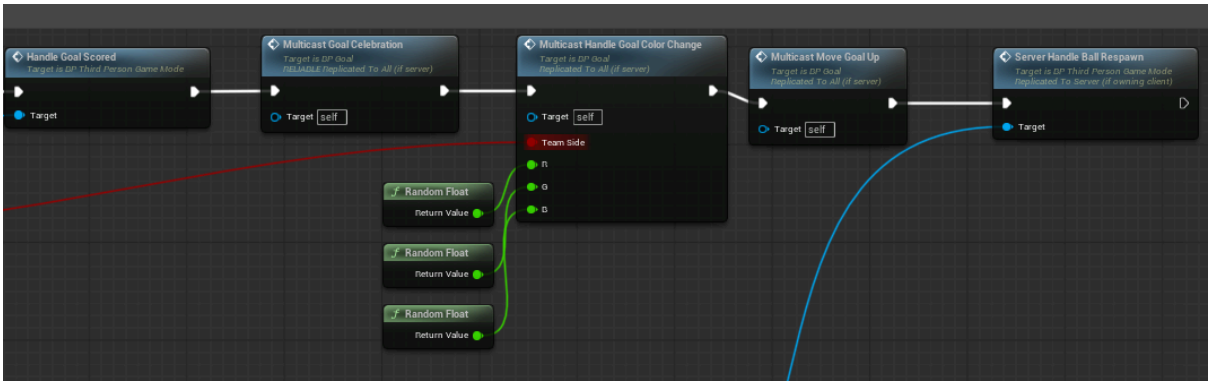
Notes and Screenshots of Key Blueprints structure:

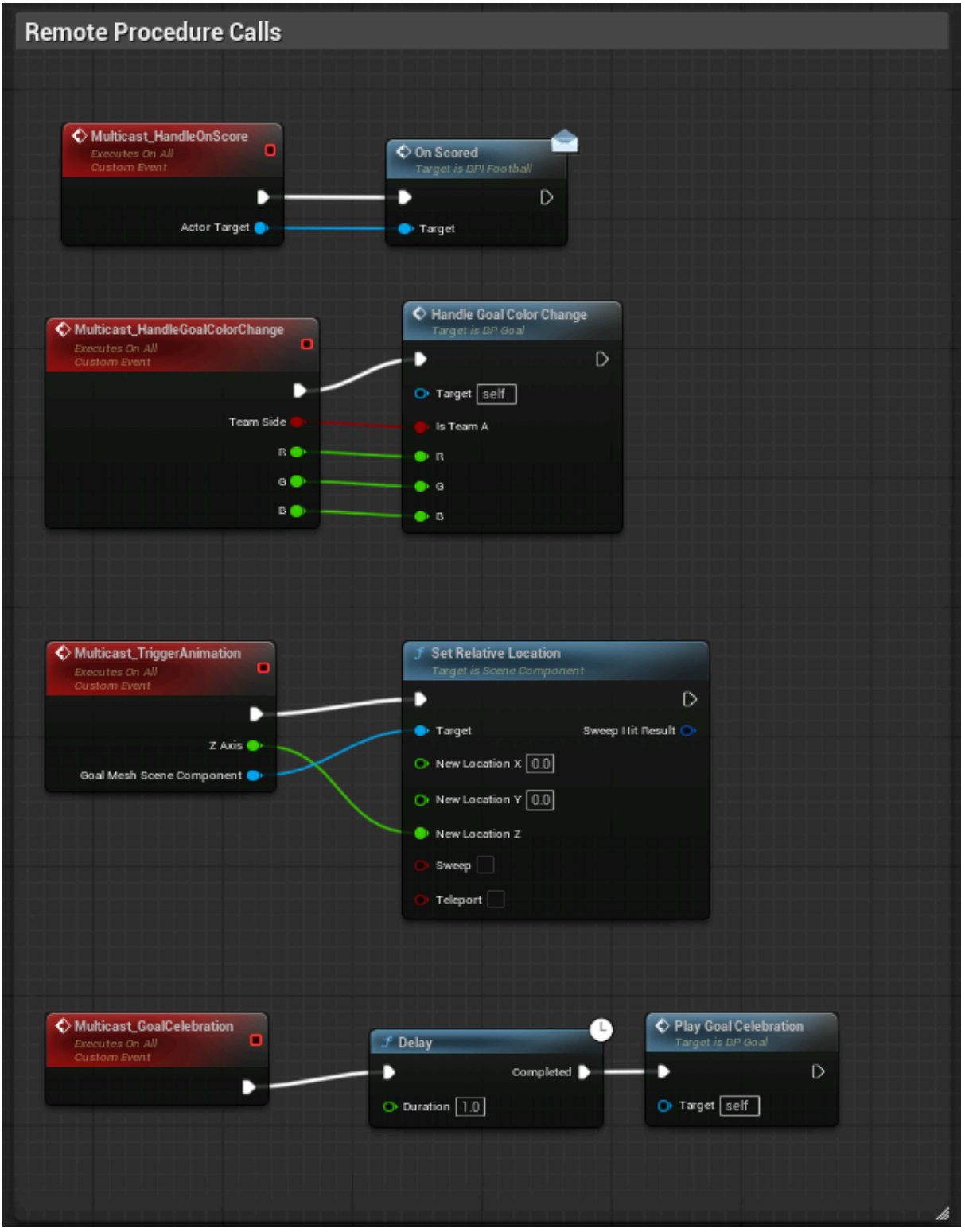
Simple Player Start for player spawn logic:



This is simple player start location choosing logic where you start with overriding the `ChoosePlayerStart` function from the Game Mode Blueprint and inside the graph, you find all the actors of Player Start with Tag (you have to give each Player Start a tag) and find the number of players in world with the Length node and use the Get Node to return value to let the Engine choose the player starts with the tag respectively.

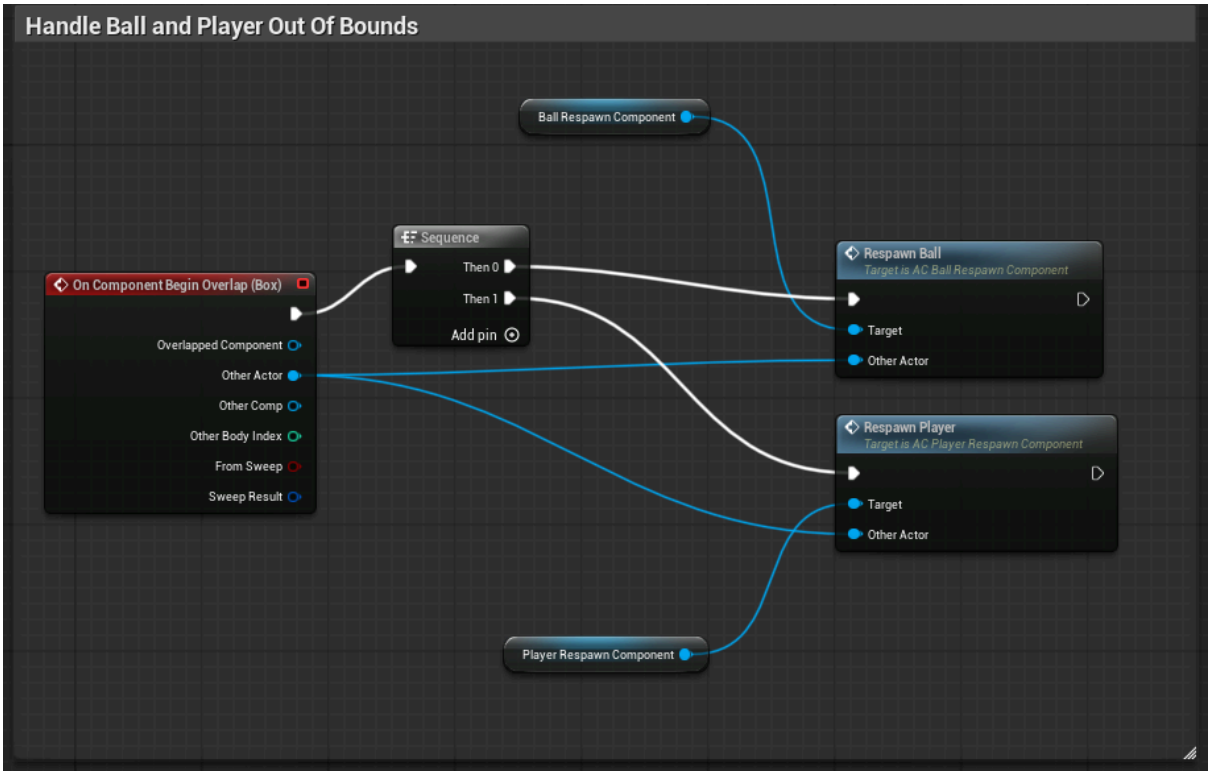
Remote Procedure Calls for Replication:





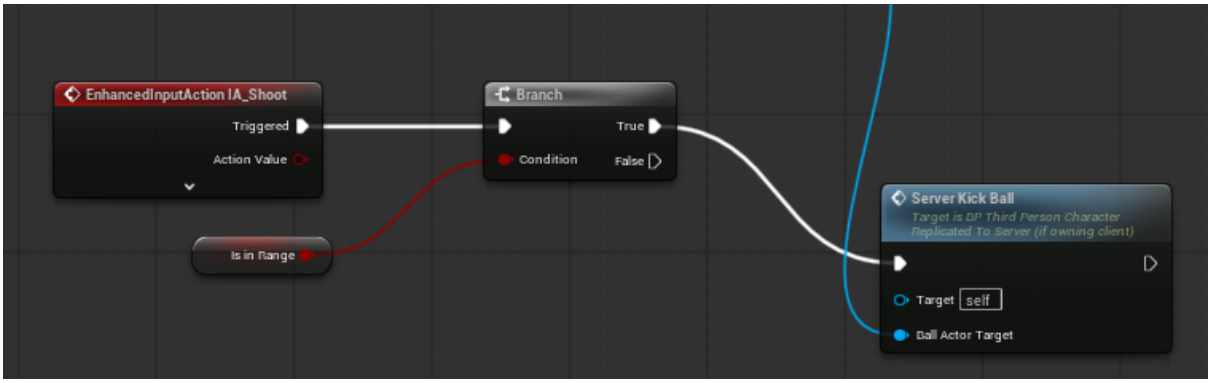
This is a simple Event/RPC system for Goal reactions when the ball collides with the Goal, all these functions are set in Multicast Replication so that all the clients including the server can see what is happening with the Goal.

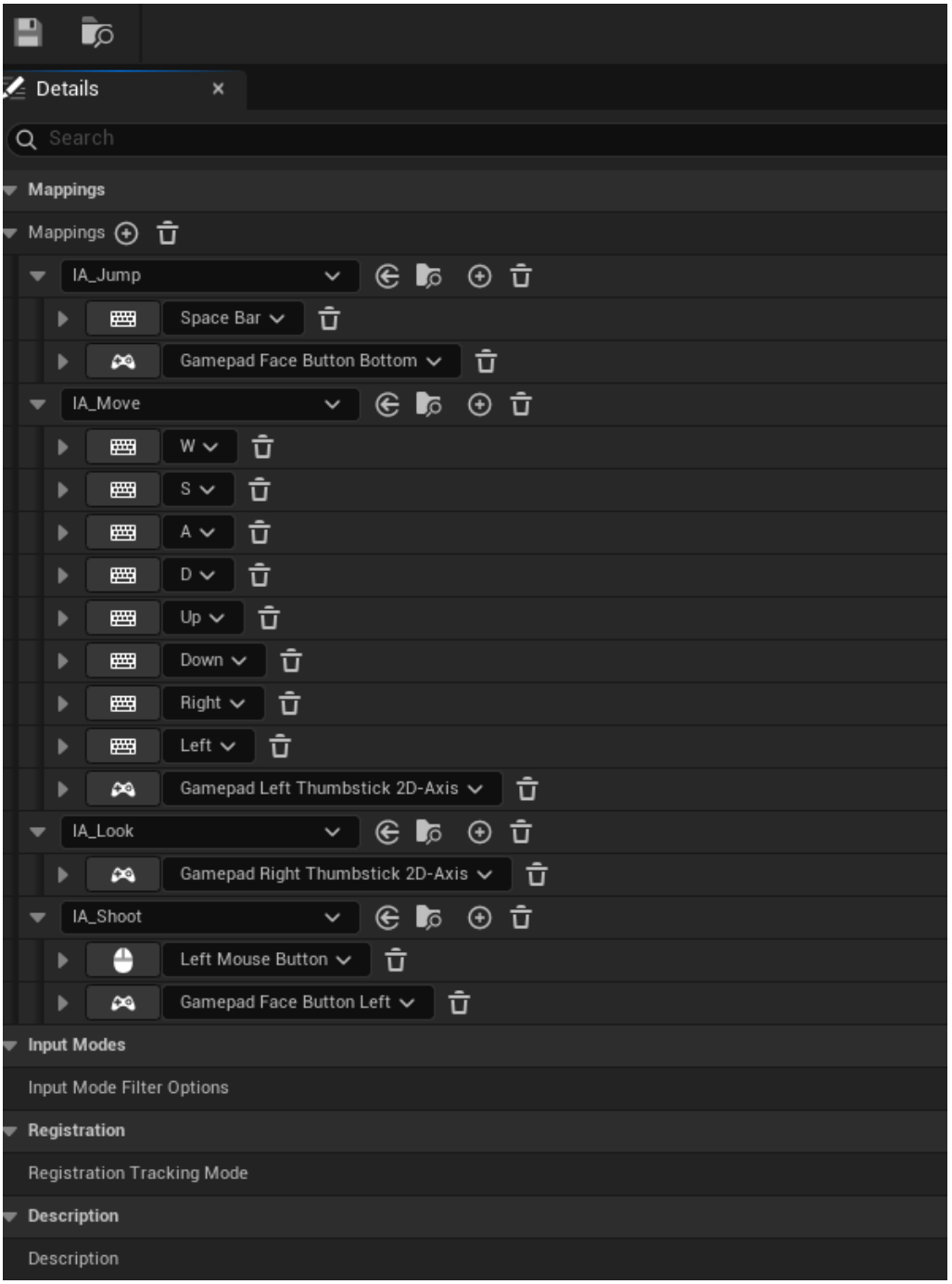
Component Based Respawn Logic:



This is a simple component based respawn logic where I have two actor component blueprints for respawning ball and respawning player which can be re-used when scaling the project in the future.

Enhanced Input System:





Using the Default Enhanced Input system from the Unreal Engine Third Person Template. Added my own inputs for ball shooting using a Gamepad and a keyboard.

Controls:

Use the **keyboard** to move and **mouse** to look around. **WASD** to move and **Left Click** to shoot the ball and **Space** to jump

Use **Gamepad Right Stick** to move and **Left Stick** to look around. **Gamepad face button left** to shoot the ball and **Gamepad face button down** to jump