



# Projektdokumentation

N. Schönmeyer      H. Staab      N. Henzen

K. Demirel      M. Mayer

Technische Hochschule Aschaffenburg

Fakultät IWIN  
Software Design '22  
Software-Entwicklungsprojekt 2025

Juli 2025

# Inhaltsverzeichnis

<b>1</b>	<b>Glossar</b>	<b>4</b>
<b>2</b>	<b>Einführung</b>	<b>6</b>
2.1	Projektrahmen .....	6
2.2	Projektvision .....	6
2.3	Projektziele .....	6
<b>3</b>	<b>Projektmanagement</b>	<b>7</b>
3.1	Agiles Framework .....	7
<b>4</b>	<b>Architektur und Netzwerklogik</b>	<b>9</b>
4.1	Überblick .....	9
4.2	Unity-Projektarchitektur .....	10
4.3	Szenenablauf .....	11
4.4	Aufbau und Prozesse der FinalMapScene .....	12
4.5	Netzwerkarchitektur .....	13
4.6	Physik- und Ownership-Synchronisation .....	15
4.7	VR-Integration und Pico SDK .....	17
4.8	SpectatorView System .....	18
4.9	Fehlerbehandlung und Stabilität .....	18
4.10	Herausforderungen und Limitationen .....	19
<b>5</b>	<b>Spielmechanik, Gameplay-Design &amp; Unity-Testing</b>	<b>20</b>
5.1	Anforderungen & Ziele der Spielmechaniken .....	20
5.2	Kernmechaniken .....	20
5.3	Zusätzliche Spielfeatures .....	25
5.4	Evolution der Mechaniken .....	31
5.5	Spectator View .....	31
5.6	Tutorial .....	33
<b>6</b>	<b>Raumorientierung</b>	<b>35</b>
6.1	Anforderungen .....	35
6.2	Spatial Mesh .....	36
6.3	Zusätzliche Orientierungshilfen .....	38
6.4	Movement Scaling .....	40
<b>7</b>	<b>Map-Design &amp; Raumorientierung</b>	<b>42</b>
7.1	Visuelle Konzepte und MVP-Basis .....	42
7.2	Konzeptphase: Stadion vs. Canyon .....	44
<b>8</b>	<b>Testing und Debugging</b>	<b>50</b>

8.1	Test-Strategien .....	50
8.2	Debugging-Strategien .....	52
<b>9</b>	<b>Roadmap &amp; Erweiterungsmöglichkeiten</b>	<b>55</b>
9.1	Teleportation einschränken .....	55
9.2	Weitere Developer Shortcuts .....	55
9.3	Hintergrundmusik .....	55
9.4	Tutorial verbessern .....	56
9.5	Barrierefreiheit .....	56
9.6	Items .....	56
9.7	Kameralogik .....	56
9.8	Ballindikator mit Ball verknüpfen .....	57
<b>10</b>	<b>Bedienungsanleitung &amp; Projektsetup</b>	<b>58</b>
10.1	Systemvoraussetzungen .....	58
10.2	Schritt-für-Schritt Projekteinrichtung .....	59
10.3	InputMapping & XR-Toolkit .....	64
10.4	Szenen & Assets .....	67
10.5	Build & Deployment .....	71
10.6	Weiterführende Ressourcen .....	75
<b>11</b>	<b>Git Workflow &amp; Kollaboration</b>	<b>78</b>
11.1	Kernkonzepte .....	78
11.2	Git-Workflow .....	78
11.3	Kollaboration und Konflikte .....	80

# 1 Glossar

**Build & Deployment:** Der Prozess, bei dem das Unity-Projekt in eine eigenständige, ausführbare Anwendung (Build) kompiliert und auf die Pico VR-Brille (Deployment) übertragen wird.

**Client:** Eine der VR-Brillen (die rote) oder die SpectatorView, die einem bestehenden Spiel beitritt, das vom Host gestartet wurde.

**Collider:** Eine Komponente in Unity, die die physikalische Form eines Objekts für Kollisionen definiert. Im Projekt wird dies genutzt, um zu erkennen, wo der Ball aufkommt (z.B. im Spielfeld oder im Fluss).

**Developer Codes:** Spezifische Tastenkombinationen auf den Controllern, die es Entwicklern ermöglichen, Testabläufe zu beschleunigen, z.B. durch das Neuladen des Tutorials oder das Zurücksetzen des Balls.

**GameObject:** Das fundamentale Bauteil jeder Szene in Unity. Es kann alles sein, von sichtbaren Objekten wie dem Spieler oder dem Ball bis zu unsichtbaren Verwaltungsobjekten wie dem ScoreManager.

**Git:** Das Versionskontrollsysteem, das zur Verwaltung des gesamten Projekt-codes und zur Kollaboration im Team verwendet wird.

**Hand-Spion:** Eine Funktion, bei der der Spieler durch ein rundes Fenster in seiner Hand in die reale Welt blicken kann, um sich zu orientieren. Dies wird aktiviert, indem die linke Hand über eine bestimmte Höhe gehalten wird.

**Host:** Die VR-Brille (die blaue), die das Spiel startet und als Server für die Spielsitzung im lokalen Netzwerk agiert.

**Netcode for GameObjects (NGO):** Das offizielle Netzwerk-Framework von Unity, das für die gesamte Multiplayer-Kommunikation im Projekt verantwortlich ist.

**Ownership (Autorität):** Bezeichnet die Kontrolle über ein Netzwerkobjekt, insbesondere den Ball. Die Autorität wird dynamisch an den Spieler übertragen, der den Ball zuletzt geworfen hat, um die Latenz bei der Interaktion zu minimieren.

**Passthrough:** Eine Funktion des Pico SDK, die es ermöglicht, die reale Umgebung durch die Kameras der VR-Brille darzustellen. Dies wird für den „Hand-Spion“ und die durchsichtigen Ränder der Sicht genutzt.

**Pico SDK:** Das „Pico Unity Integration SDK“, eine Softwareschnittstelle, die den Zugriff auf die Hardware-Funktionen der Pico 4 Brillen ermöglicht, wie z.B. Kopf- und Controller-Tracking.

**Prefab:** Ein wiederverwendbarer „Bauplan“ für ein GameObject in Unity. Im Projekt werden Prefabs für den Ball, Spieler-Avatare und UI-Elemente verwendet, um die Modularität zu erhöhen und Merge-Konflikte in Szenen zu vermeiden.

**QuickMatch:** Das selbst entwickelte System zur automatischen Geräteerkennung im lokalen Netzwerk, das auf UDP-Broadcasts basiert und es Spielern erspart, IP-Adressen manuell einzugeben.

**Remote Procedure Call (RPC):** Ein Funktionsaufruf, der über das Netzwerk an den Server oder andere Clients gesendet wird. Dies wird genutzt, um Spielereignisse wie das Erzielen von Punkten synchron zu halten.

**Scene (Szene):** Ein Level oder ein abgeschlossener Bereich des Spiels in Unity, z.B. die StartupScene, ConfigScene oder die Haupt-Spielarena FinalMapScene.

**Spatial Mesh:** Ein 3D-Gitternetz der realen Umgebung, das vom Tiefensensor der Pico-Brille erfasst wird. Es wird im Spiel als transparente, visuelle Hilfe eingeblendet, damit sich Spieler sicher im Raum bewegen können.

**SpectatorView:** Eine eigenständige Anwendung für Windows/macOS, die es Zuschauern erlaubt, dem Spiel als normaler Client beizutreten und das Geschehen aus verschiedenen Kameraperspektiven zu verfolgen.

**UDP Broadcast:** Die Netzwerktechnologie, die vom QuickMatch-System genutzt wird. Der Host sendet periodisch ein Datenpaket an alle Geräte im Netzwerk, wodurch Clients den Host automatisch finden können.

**Unity Transport:** Das zugrundeliegende Netzwerkprotokoll (basierend auf UDP), das von Netcode for GameObjects für die Datenübertragung genutzt wird.

**XR Interaction Toolkit:** Ein Paket von Unity, das die grundlegenden VR-Interaktionen wie Greifen, Werfen und Teleportieren bereitstellt und im Projekt als Basis für die Spielermechanik dient.

## 2 Einführung

Dieses Dokument dient als zentraler Leitfaden für neue Entwickler, um die Softwarearchitektur, das Netzwerkdesign und die Kernmechaniken des Projekts zu verstehen.

GrandSlamVR ist ein VR-Multiplayer-Tennisspiel, das für zwei **Pico 4 Enterprise**-Brillen entwickelt wurde. Es ermöglicht ein vollständig kabelloses Spielerlebnis in einem lokalen Netzwerk, optional ergänzt durch eine **SpectatorView** für Windows und macOS.

### 2.1 Projektrahmen

Das Projekt GrandSlamVR ist im Rahmen des Software-Entwicklungsprojekts im 6. Semester des Studiengangs Software Design der Fakultät IWIN entstanden.

### 2.2 Projektvision

„Unser verteiltes VR „Tennis“ Spiel liefert sofort ab der ersten Runde ein interaktives, dynamisches und immersives Spielerlebnis, in dem es sportliche Aktivität spielerisch mit Virtueller Realität verknüpft. Zudem soll das Spiel zu zweit über das Netzwerk gespielt werden und soll den Spielern kurzfristig ein simples und unterhaltsames Erlebnis bieten ohne übergeordnete Spielprogression.“

### 2.3 Projektziele

„Wir möchten ein VR „Tennis“ Spiel agil innerhalb von 3 Monaten entwickeln, in dem sich zwei Spieler verteilt über das Netzwerk verbinden und gemeinsam „Tennis“ spielen können.“

„Das Spiel soll einfach genug aufzubauen und zu spielen sein, um spontan auf kleineren Veranstaltungen vorgestellt werden zu können.“

„Das Spiel soll intuitiv und in unter 10 Minuten erlernbar sein.“

„Das Spiel performant sein und durchgängig mindestens 60 FPS gewährleisten können, um dem nutzer ein flüssiges Spielerlebnis zu bieten.“

## 3 Projektmanagement

### 3.1 Agiles Framework

Nachdem uns Prof. Biedermann die Aufgabe erklärt hatte, wurde uns schnell klar, dass das Thema technisch ziemlich anspruchsvoll ist: Multiplayer-VR auf der Pico-Brille. Damit wir alle das gleiche Ziel vor Augen hatten, haben wir eine gemeinsame Produktidee festgelegt. Unser VR-Tennisspiel soll ab der ersten Runde Spaß machen – es soll direkt funktionieren, leicht verständlich sein und ein echtes Spielerlebnis bieten. Zwei Personen sollen übers Netzwerk spielen können, ohne lange Erklärungen. Keine Levels oder Fortschrittssysteme – einfach ein spannendes und sportliches Match in der virtuellen Realität

Da Prof. Biedermann möglichst schnell eine funktionierende Version sehen wollte und am Anfang noch vieles unklar war, habe ich vorgeschlagen, das Ganze in **Zwei-Wochen-Sprints** aufzubauen. **Scrum** hat sich dabei als sehr praktisch erwiesen: Damit konnten wir flexibel arbeiten, auf neue Erkenntnisse reagieren und den Überblick behalten. Durch feste Meetings wie Sprint Planning, Weekly Stand-ups, Sprint Review und Retrospektiven wussten wir immer, wo wir stehen. Beim Sprint Planning haben wir die User Stories aus unserem Jira-Backlog durchgesprochen, mit Planning Poker geschätzt und klare Ziele („Definition of Done“) festgelegt. Das Feedback aus dem Sprint Review haben wir direkt in neue Aufgaben übernommen, sodass unser Prototyp Schritt für Schritt stabiler und besser wurde.

Parallel dazu haben wir mit **Jira** alles dokumentiert. Jeder Task, jede Idee oder jeder Bug wurde als eigenes Ticket festgehalten – mit klarer Verantwortung, Aufwandsschätzung und Ziel. So war immer klar, wer was macht. Unsere sechs **Personas** haben uns dabei geholfen, die Anforderungen im Blick zu behalten – z.B. Prof. Biedermann selbst, VR-Neulinge am Open Campus oder Familien. Auf dieser Basis haben wir unsere Use Cases, **funktionalen** und **nicht-funktionalen Anforderungen** definiert, priorisiert und Stück für Stück eingebaut.

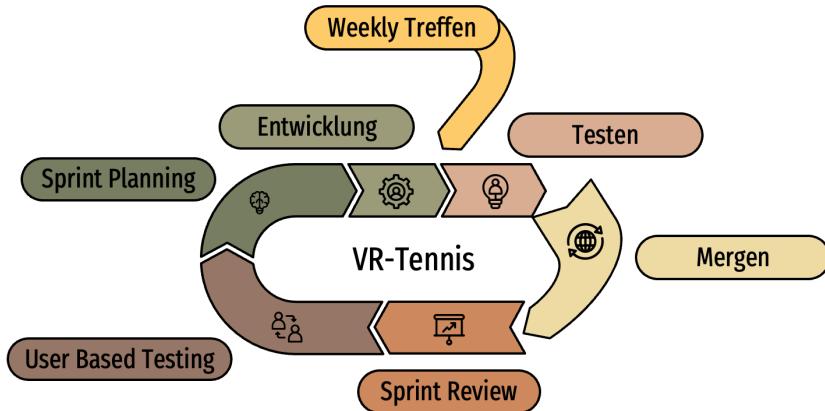
In **Jira** haben wir mit **Epics** gearbeitet – das waren große Themenbereiche, die sich oft über mehrere Sprints gezogen haben. Diese Epics wurden in kleinere **User Stories** aufgeteilt, die in einem Sprint machbar waren (meist 1 bis 5 Tage

Arbeit). Jede Story hatte klare Kriterien, wann sie „fertig“ ist und welchen Nutzen sie bringt. Manche Tickets waren eher technisch oder organisatorisch, aber nötig, um die Stories umzusetzen. So konnten wir die Arbeit feiner planen, besser einschätzen und alles gut aufteilen.

Durch diese klare Struktur konnten wir schon nach wenigen Wochen einen funktionierenden MVP zeigen – und hatten trotzdem noch genug Spielraum für weitere Verbesserungen.

Am Ende jedes Sprints haben wir **User-Based Testings** gemacht. Auch Prof. Biedermann hat als Tester mitgemacht – einmal als Betreuer des Projekts und einmal aus Sicht eines typischen Besuchers. Zusätzlich haben wir Studierende eingeladen, die das Spiel mit der VR-Brille ausprobiert haben. Wir haben ihr Verhalten beobachtet, Interviews geführt und Fragebögen genutzt. Die Ergebnisse haben wir als neue Tickets im Jira eingetragen und direkt im nächsten Sprint umgesetzt.

Diese enge Verbindung von **Planung**, **Persona-Modellierung** und regelmäßigen **Nutzerfeedback** war entscheidend dafür, dass unser Spiel am Ende nicht nur gut funktioniert hat, sondern auch Spaß gemacht hat und die Erwartungen von Prof. Biedermann und unseren Testspielern erfüllt hat.



**Abb. 1:** Inkrementeller Entwicklungsprozess (Scrum)

## 4 Architektur und Netzwerklogik

### 4.1 Überblick

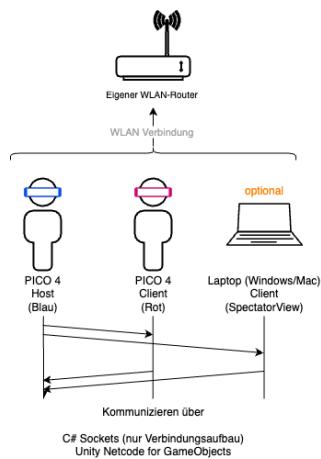
Die technologische Basis bilden **Unity**, das **Pico Unity Integration SDK** für die VR-Funktionalität und **Unity Netcode for GameObjects (NGO)** für die gesamte Netzwerkkommunikation.

#### Kernziele:

- **Serverless Operation:** Das Spiel funktioniert ohne dedizierten PC oder Server; eine der VR-Brillen agiert als Host.
- **Automatische Verbindung (QuickMatch):** Spieler müssen keine IP-Adressen eingeben. Die Geräte finden sich im lokalen Netzwerk automatisch.
- **Faires & Reaktionsschnelles Gameplay:** Eine hybride Netzwerkauftorität mit dynamischem „Ownership“-Transfer für den Ball minimiert die Latenz und sorgt für faire Spielbedingungen.

#### 4.1.1 Aufbau und Komponenten

Das folgende Diagramm visualisiert den typischen Aufbau des Systems in einem dedizierten lokalen Netzwerk:



**Abb. 2:** Überblick über Aufbau und Komponenten

## 4.2 Unity-Projektarchitektur

### 4.2.1 Unity-Grundkonzepte

Ein grundlegendes Verständnis von Unity ist für die Mitarbeit essenziell:

- **Szenen (Scenes):** Eine Szene ist ein Level oder ein abgeschlossener Bereich des Spiels (z.B. ein Menü oder die Spielarena). Sie ist der oberste Container für alle Objekte in diesem Bereich.
- **GameObjects:** Die fundamentalen Bausteine jeder Szene. Ein GameObject kann ein sichtbares 3D-Modell (Spieler, Ball), eine Lichtquelle, eine Kamera oder ein unsichtbares Verwaltungsobjekt (z.B. ein ScoreManager) sein.
- **Komponenten (Components):** GameObjects erhalten ihre Funktionalität durch Komponenten. Dies können eingebaute Komponenten wie ein Rigidbody (für Physik) oder Collider (für Kollisionen) sein. Die eigentliche Spiellogik wird in unseren eigenen **C#-Skripten** geschrieben, die ebenfalls als Komponenten an GameObjects angehängt werden.
- **Prefabs:** Dies sind wiederverwendbare „Baupläne“ für GameObjects. Ein Prefab speichert ein GameObject inklusive all seiner Komponenten, Einstellungen und Kind-Objekte. Unser Ball ist beispielsweise ein Prefab, was es uns erlaubt, ihn einfach in der Szene zu platzieren oder zur Laufzeit zu instanziieren. Änderungen am Prefab wirken sich auf alle seine Instanzen aus.

### 4.2.2 Projekt-Ordnerstruktur

Das Projekt ist zur besseren Übersichtlichkeit und Wartbarkeit logisch strukturiert. Die meisten Ordnernamen sind selbsterklärend, die wichtigsten sollten jedoch hervorgehoben werden. Ausschlaggebend sind diese drei Unterordner des **Assets**-Ordners:

- **Scenes/:** Enthält alle Spielszenen: Startup, Config, Tutorial und FinalMap.
- **Prefabs/:** Beinhaltet alle Prefabs wie den Ball (BallInteractable), Spieler-Avatare und UI-Elemente.
- **Scripts/:** Der Hauptordner für den gesamten C#-Code. Beinhaltet weitere Unterordner zur Kategorisierung der Skripte.

## 4.3 Szenenablauf

Das Spiel führt beim Start durch einen klar definierten Ablauf von Szenen.

Die möglichen Übergänge zwischen den einzelnen Szenen sind der folgenden Abbildung zu entnehmen. Anschließend wird auf die einzelnen Komponenten genauer eingegangen.

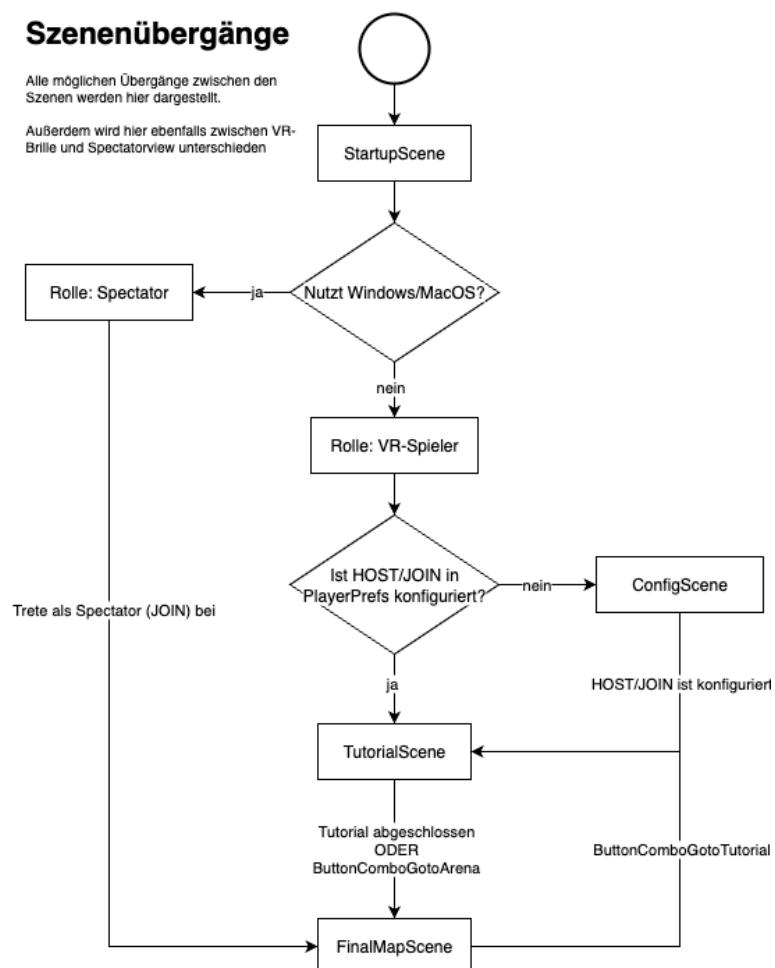


Abb. 3: Übergänge zwischen den Szenen

### 4.3.1 Spielstart-Ablauf

1. **App-Start & Startup Scene:** Beim Start der Anwendung wird die StartupScene geladen. Das Skript `StartupScript.cs` prüft den Player-

Pref-Schlüssel "RunMode", um festzustellen, ob die Anwendung als VR-Brille ("VR") oder als **Spectator** läuft.

2. Der **Spectator** Lädt direkt die FinalMapScene. Jede **VR-Brille** prüft vorab, ob der PlayerPref-Schlüssel "PlayerColor" gesetzt ist, also eine Rolle konfiguriert wurde.

Wenn eine Rolle bereits konfiguriert wurde, wird FinalMapScene direkt geladen. Ist jedoch noch keine Rolle zugewiesen, wird zunächst die ConfigScene geladen.

**Hinweis:** Der RunMode PlayerPref wird durch das AssignPlayerColor.cs Skript zusammen mit der PlayerColor gesetzt, wenn eine Rolle in der ConfigScene ausgewählt wird.

3. **Config Scene:** In dieser Szene wählt der Benutzer eine Farbe, die festlegt, ob die Brille als Host oder Client agiert:

**Blau = Host:** Diese Brille startet das Spiel und fungiert als Server.

**Rot = Client:** Diese Brille tritt einem bestehenden Spiel bei.

Die Auswahl wird im AssignPlayerColor.cs-Skript verarbeitet und in den PlayerPrefs gespeichert.

4. **Tutorial Scene:** Nach der Konfiguration wird die TutorialScene geladen. Hier lernen die Spieler die grundlegenden Mechaniken wie Werfen, Fangen und Teleportation.
5. **Final Map Scene:** Dies ist die Haupt-Spielszene. Hier findet das eigentliche Match statt. Alle Netzwerkverbindungen werden hergestellt, die Spiellogik ist aktiv und die SpectatorView kann sich verbinden.

#### 4.4 Aufbau und Prozesse der FinalMapScene

Im Spiel befindet sich der größte Teil der Spiellogik innerhalb der **Final Map Scene**. Die folgende Abbildung bietet einen guten Überblick über die Objekte und Komponenten, welche in dieser Szene enthalten sind. Die Abbildung ist essenziell, um Vorgänge zu verstehen und wird im Folgenden durch weitere Informationen zusätzlich ergänzt.

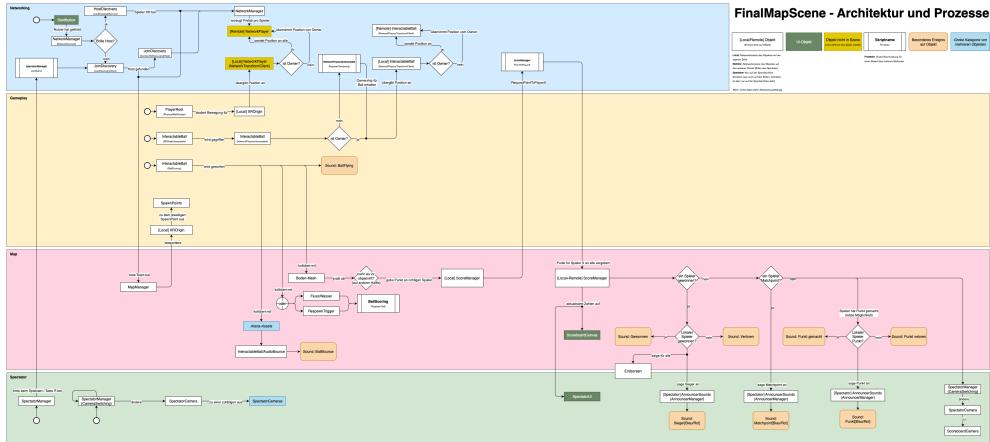


Abb. 4: Architektur und Prozesse der FinalMapScene

Die Abbildung ist in vergrößerter Form ebenfalls im Repository vorzufinden.

## 4.5 Netzwerkarchitektur

### 4.5.1 Client-Server-Modell mit hybrider Autorität

GrandSlamVR nutzt ein Client-Server-Modell mit einer wichtigen Besonderheit: **Eine der VR-Brillen agiert immer als Host (Server)**. Dies ermöglicht den Verzicht auf einen externen PC.

Wir verwenden ein **hybrides Autoritätsmodell**:

- Der **Host** ist für die allgemeine Spiel-Logik und die Zustandsverwaltung zuständig.
- Die **Autorität über kritische Objekte** (insbesondere den Ball) wird **dynamisch** an den jeweils interagierenden Spieler übertragen.

Dieser Ansatz wurde gewählt, um Latenz-Nachteile für den Client zu minimieren und ein faires Spielgefühl zu schaffen. Der Werfer des Balls hat die geringste Verzögerung bei seiner wichtigsten Aktion. Der Nachteil – eine theoretisch höhere Anfälligkeit für Spielmanipulationen (Cheating) – ist jedoch für den derzeit vorgesehenen Einsatzzweck, der aus den Anforderungen hervorgeht (Events an der Hochschule), irrelevant.

#### 4.5.2 Automatische Geräteerkennung (QuickMatch)

Da Unity Netcode for GameObjects keine eingebaute LAN-Discovery-Funktion bietet, haben wir ein eigenes System entwickelt, das auf UDP-Broadcasts basiert.

##### Technischer Ablauf:

1. **Server sendet Broadcast:** Nach dem Start der FinalMapScene beginnt der Host (blaue Brille) über das Skript `LanBroadcastService.cs`, periodisch ein UDP-Paket zu versenden. Dieses Paket enthält standardmäßig das Token "NGO\_HOST" und wird an die Broadcast-Adresse des Netzwerks (255.255.255.255) auf Port 7777 gesendet.
2. **Client wartet:** Der Client (rote Brille) startet über `LanDiscoveryClient.cs` einen Listener auf demselben Port. Dieser läuft in einem separaten Thread, um das Spiel nicht zu blockieren.
3. **Verbindung herstellen:** Sobald der Client ein Paket mit dem korrekten Token empfängt, extrahiert er die IP-Adresse des Absenders. Das `NetworkConnect.cs`-Skript übergibt diese IP an den `UnityTransport`, und `NetworkManager.StartClient()` initiiert die Verbindung zum Host.

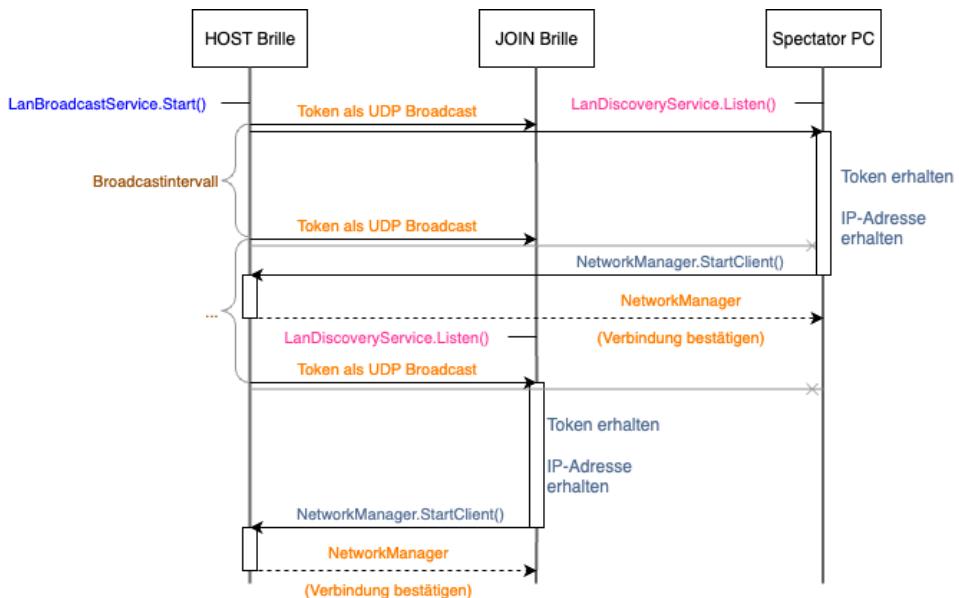


Abb. 5: Sequenzdiagramm zum Verbindungsaufbau

### 4.5.3 Synchronisation via RPCs

Ereignisse wie das Erzielen von Punkten oder das Abspielen von Sounds werden über **Remote Procedure Calls (RPCs)** synchronisiert. Wir verwenden ein einheitliches Pattern, um die Konsistenz sicherzustellen und die server-autoritative Kontrolle über kritische Events zu wahren. Ein Client kann den Server lediglich um die Durchführung einer Aktion bitten; der Server trifft die endgültige Entscheidung und kommuniziert diese an alle. Dies erhöht die Sicherheit und Konsistenz des Spielzustands.

1. **Request-Methode:** Der Gameplay-Code ruft eine einzige Methode auf (z.B. `PointToPlayer1Request()`).
2. **Autoritäts-Check:** Diese Methode prüft, ob sie auf dem Server (Host) oder einem Client ausgeführt wird.
  - **Auf dem Client:** Sie ruft eine `ServerRpc`-Methode auf (z.B. `addPointToPlayer1ServerRpc()`), die nur an den Server gesendet wird.
  - **Auf dem Server:** Sie führt die Logik direkt aus und ruft zusätzlich eine `ClientRpc` (oder `Rpc` an alle) auf.
3. **Server führt aus:** Der Server empfängt den `ServerRpc`, führt die Kernlogik aus und ruft seinerseits einen RPC auf, der an alle Clients (und optional sich selbst) gesendet wird (z.B. `addPointToPlayer1Rpc()`).
4. **Alle führen aus:** Nun führen alle verbundenen Instanzen die finale Methode aus, die die eigentliche Spiellogik enthält (z.B. `addPointToPlayer1()`).

## 4.6 Physik- und Ownership-Synchronisation

### 4.6.1 Dynamischer Ownership-Transfer für den Ball

Um ein reaktionsschnelles Werfen zu ermöglichen, wechselt die Autorität über die Physik des Balls dynamisch: **Der Spieler, der den Ball zuletzt geworfen hat oder hält, besitzt das Ownership.**

**Vorteil:** Der werfende Spieler berechnet die Flugbahn lokal auf seiner Brille, was zu einer direkten, latenzfreien Interaktion führt.

**Mechanismus:** Der Owner sendet die Positions-Updates des Balls an den anderen Spieler, der diese interpoliert darstellt. Beim Fangen wechselt das Ownership.

**Implementierung:** Der Ownership-Wechsel wird über die eingebaute NetworkObject.ChangeOwnership()-Funktion von Unity Netcode realisiert, die typischerweise vom Server nach einer RPC-Anfrage eines Clients aufgerufen wird.

#### 4.6.2 NetworkPhysicsInteractable.cs

Diese Klasse ist das Herzstück der Ball-Interaktion. Sie ist eine Erweiterung von NetworkBaseInteractable und speziell für physikbasierte Objekte konzipiert. Sie verwaltet:

Das **Anfordern des Ownerships** vom Server via RPC.

Die **Wurf-Physik**, die auf der Handgeschwindigkeit des Spielers im Moment des Loslassens basiert.

**Ball-spezifische Logik** wie das Auslösen von Leuchtspuren (triggerTrailsRpc) oder das Abspielen von Sounds.

Im Laufe der Entwicklung wurde diese Klasse sehr spezifisch auf den Ball zugeschnitten. Ein Refactoring wäre nötig, um sie generischer zu machen.

**Bekannte technische Schuld & Einstiegsmöglichkeit:** Die NetworkPhysicsInteractable.cs-Klasse ist ein gutes Beispiel für Code, der organisch gewachsen ist. Eine hervorragende erste Aufgabe für einen neuen Entwickler wäre es, diese Klasse zu refaktorisieren: Die generische Physik- und Ownership-Logik könnte in der Basisklasse verbleiben, während die ballspezifische Logik (Trails, Scoring-Referenzen, spezielle Sounds, haptisches Feedback) in eine neue, erbende Klasse wie BallInteractable.cs ausgelagert wird.

#### 4.6.3 NetworkTransformClient & NetworkPhysicsTransformClient

Der Standard-NetworkTransform von Unity Netcode erlaubt nur dem Server, Positions-Updates zu senden. Da in unserem System auch Clients die Autorität über den Ball haben können, haben wir eine eigene Komponente entwickelt: NetworkTransformClient.

Diese Klasse überschreibt OnIsServerAuthoritative() und gibt false zurück, was client-autoritative Updates ermöglicht.

Der NetworkPhysicsTransformClient nutzt dies für die Ball-Synchronisation:

Der **Owner** schreibt seine lokale Position kontinuierlich in eine NetworkVariable.

**Nicht-Owner** lesen den Wert aus dieser Variable und interpolieren die Position ihres lokalen Ball-Objekts sanft (Vector3.Lerp) auf diesen Zielwert zu. Dies verhindert den „Jitter“-Effekt, der durch die geringere Frequenz von Netzwerk-Updates im Vergleich zur Framerate entsteht.

#### 4.6.4 Spieler-Avatar-Synchronisation

Die Bewegung von Kopf und Händen der Spieler-Avatare wird mit demselben NetworkTransformClient-System synchronisiert. Eine Besonderheit ist hier das **lokale Ausblenden des eigenen Avatars**: Jeder Spieler ist Owner seines eigenen Avatars. Ein Skript stellt sicher, dass die Renderer-Komponenten des eigenen Avatars lokal deaktiviert werden, um die Sicht nicht zu behindern. Der Gegner sieht den Avatar natürlich weiterhin.

### 4.7 VR-Integration und Pico SDK

#### 4.7.1 Rolle des Pico SDK

Das **Pico Unity Integration SDK** ist die Schnittstelle zur Hardware der Pico 4 Enterprise Brillen. Es ermöglicht uns den Zugriff auf Kernfunktionen:

**Head & Motion Tracking:** Präzise Verfolgung der Kopf- und Controller-Bewegungen (6DoF).

**Controller Inputs:** Abfrage von Tasten, Triggern und Joysticks.

**Passthrough (Video Seethrough):** Darstellung der realen Umgebung in der VR-Brille.

**Spatial Mesh:** Vermessung der realen Umgebung und Erstellung eines 3D-Gitternetzes.

#### 4.7.2 SDK-Erweiterungen

Wir haben das Standard-SDK an zwei entscheidenden Stellen für unsere Bedürfnisse erweitert:

**Dynamisches Spatial Mesh:** Standardmäßig wird das Spatial Mesh am Welt-Ursprung generiert. Wir haben den **SpatialMeshManager** so angepasst, dass das Mesh dynamisch um die aktuelle Spielerposition generiert wird und ihm folgt.

Weitere Informationen im Abschnitt: Spatial Mesh.

**Passthrough als Unity Material:** Wir haben die Passthrough-Funktion als wiederverwendbares Unity-Material implementiert. Dies ermöglicht es uns, den Seethrough-Effekt auf beliebige 3D-Objekte (Meshes) anzuwenden.

Weitere Informationen im Abschnitt: Hand Spion.

**Haptisches Feedback:** Für eine verbesserte Immersion wird haptisches Feedback bei wichtigen Interaktionen wie dem Werfen und Fangen des Balls über `PXR_Input.SendHaptic...` ausgelöst.

## 4.8 SpectatorView System

Die SpectatorView ist eine eigenständige Windows/macOS-Anwendung, die es Zuschauern erlaubt, das Spiel zu verfolgen.

**Automatische Erkennung:** Das StartupScript.cs erkennt die Plattform und sorgt dafür, dass die SpectatorView direkt in die FinalMapScene springt.

**Automatische Verbindung:** Ein SpectatorManager-Skript in der Szene startet automatisch den LanDiscoveryClient-Prozess. Die SpectatorView findet den Host im Netzwerk und verbindet sich als regulärer Client, ohne dass eine manuelle Eingabe erforderlich ist. Sie ist **immer ein Client**, da das Spiel so konzipiert ist, dass eine VR-Brille die Host-Rolle übernimmt.

## 4.9 Fehlerbehandlung und Stabilität

### Verbindungsabbrüche:

**Kurzzeitig (< 10s):** Unity Netcode versucht eine automatische Wiederverbindung. Der Spielzustand bleibt in der Regel erhalten.

**Dauerhaft:** Die Verbindung wird sauber getrennt. Eine Neuverbindung ist nach einem Neustart des Matches jederzeit möglich.

**Netzwerk-Infrastruktur:** Öffentliche WLANs oder Uni-Netzwerke (z.B. mit „Client Isolation“) können die UDP-Broadcasts blockieren und die Geräteerkennung verhindern.

**Lösung:** Für einen stabilen Betrieb wird dringend die Verwendung eines **eigenen, dedizierten WLAN-Routers** in einer Standard-Heimnetz-Konfiguration empfohlen. Alle Geräte müssen sich mit demselben Router verbinden.

**Verlust von PlayerPrefs:** Nach einem Build von einem anderen Git-Branch oder einem großen Update können die auf dem Gerät gespeicherten PlayerPrefs

(Host/Client-Einstellung) verloren gehen. In diesem Fall muss die Rolle der Brille über die ConfigScene einfach neu zugewiesen werden.

## **4.10 Herausforderungen und Limitationen**

Jede Architektur bringt gewisse Vor- und Nachteile mit sich. Auch unsere bleibt nicht von Herausforderungen und Limitationen verschont.

### **4.10.1 Umstellen der Build-Targets**

Möchte man auf einem Laptop eine Änderung testen, welche sowohl die Brillen, als auch die SpectatorView betrifft, so muss man das BuildTarget jedes Mal in den Unity-Build-Settings umstellen. Das dauert nicht nur je nach Leistung des Computers relativ lange, sondern Unity behält den Build-Cache für die Android App nach der erneuten Umstellung von SpectatorView zu Android nicht bei.

Das kostet bei jedem größeren Build enorm Zeit. Eine mögliche Alternative wäre, die SpectatorView komplett in ein eingenständiges Projekt auszulagern. Was allerdings Redundanz und weitere neue Probleme mit sich bringt. Aus unserer Sicht bringt eine Auslagerung in ein neues Projekt demnach keinen maßgeblichen Vorteil gegenüber der jetzigen Implementierung.

### **4.10.2 Mergen in Git - Modularer Aufbau der Szene**

Aus bisheriger Erfahrung geht hervor, dass es zu Problemen mit Mergekonflikten führt, wenn mehrere Entwickler gleichzeitig eine Szene in Unity bearbeitet haben und diese Änderungen zusammenführen möchten.

Daher haben wir Teile der Szene in Prefabs ausgelagert und ändern stattdessen die Prefabs statt der eigentlichen Szene. Die Szenendatei bleibt dabei unverändert, die Änderungen werden aber dennoch übernommen. Diesen modularen Aufbau könnte man ebenfalls noch erweitern und weitere Komponenten ebenfalls auslagern.

Allerdings sollten die Prefabs auch nicht gleichzeitig bearbeitet werden, da es zu dem selben Problem führt, also sollten diese so gewählt werden, dass es normalerweise nicht nötig sein sollte, diese gemeinsam zu bearbeiten. Stückelt man aber zu feingranular, wird der Entwicklungs - und Wartungsaufwand unnötig erhöht.

Abschließend ist anzumerken, dass Mergekonflikte in Skripten - im Gegensatz zu Szenen - lösbar sind und kein nennenswertes Problem darstellen.

Hendrik Staab

## 5 Spielmechanik, Gameplay-Design & Unity-Testing

In diesem Abschnitt wird auf die technische Umsetzung der Spielmechaniken und -features eingegangen. Genauere Beschreibungen sind in den jeweils angegebenen Skripten in Form von Kommentaren zu finden.

### 5.1 Anforderungen & Ziele der Spielmechaniken

Aus den Anforderungen an das Produkt ergaben sich für die Spielmechanik diese Prioritäten:

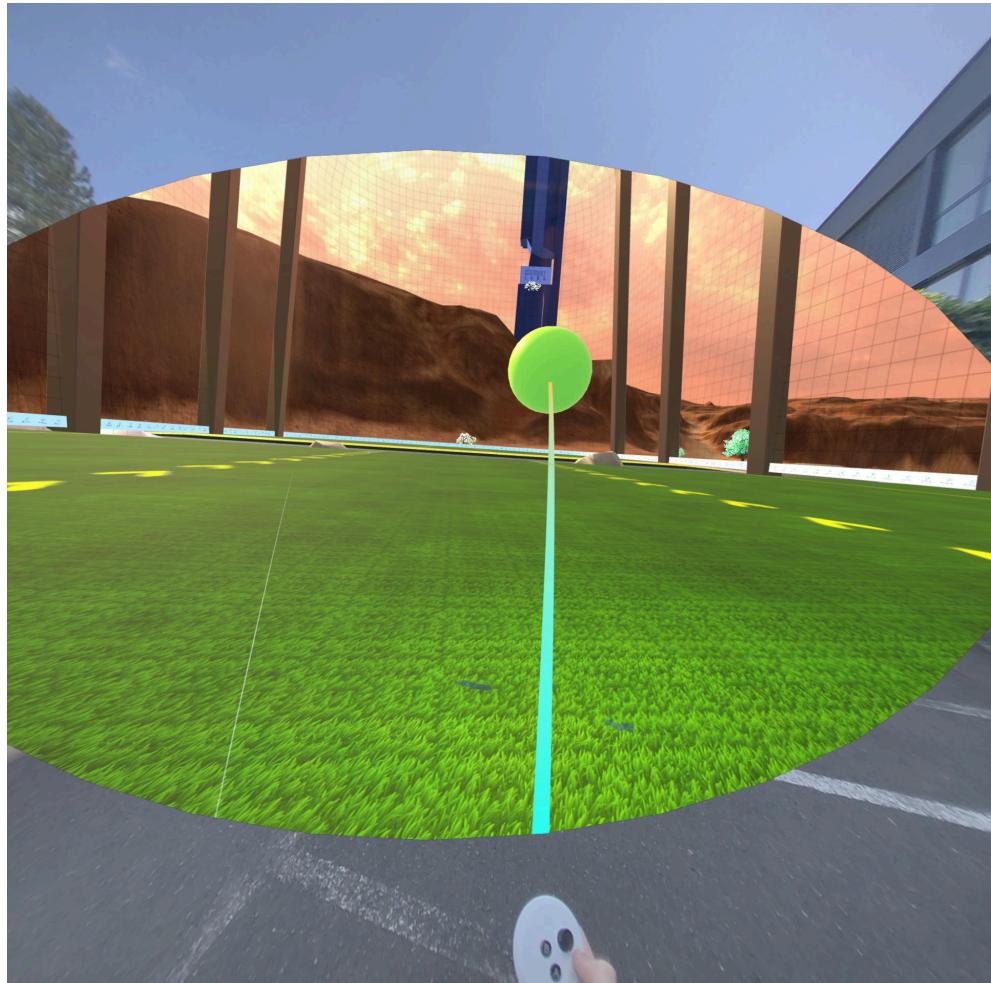
- Verständlichkeit
- Einfachheit
- spannendes Gameplay

Diese leiten sich direkt aus den Anforderungen & Personas ab (siehe Anforderungsanalyse & Personas)

### 5.2 Kernmechaniken

#### 5.2.1 Werfen & Fangen

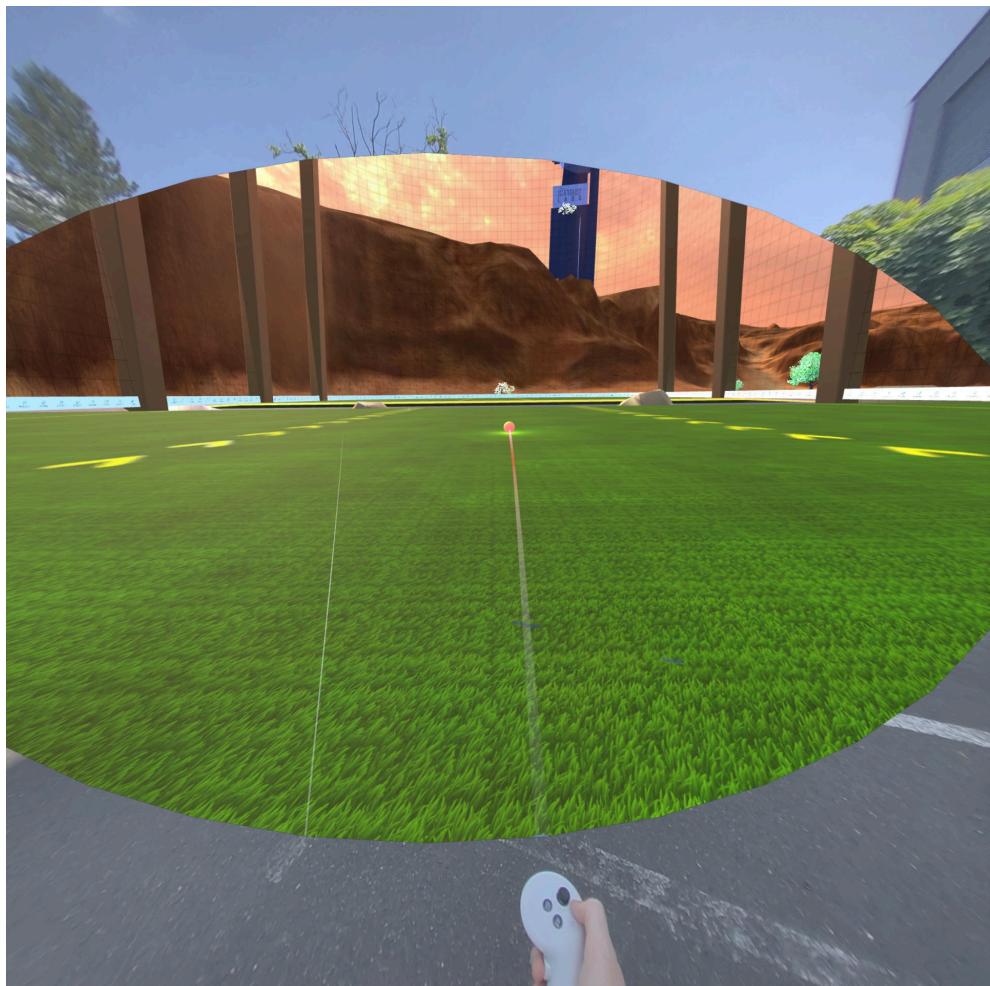
Die Wurfmechanik basiert auf den nativen Physik-Interaktionen des Unity XR Rigs. Ein Wurf wird durch die reale Bewegung der Hand des Spielers erzeugt, wobei das Objekt – ein Ball – durch die XR-Interaktion mithilfe dem gedrückt Halten der seitlichen Trigger-Taste aufgenommen, geworfen und per Netzwerk zwischen den Spielern synchronisiert wird (Details zur Synchronisation siehe Physik- und Ownership-Synchronisation).



Zum Fangen wird wie beim Aufheben die Trigger-Taste gehalten, sobald der Ball in der Luft anvisiert wird. Die Trigger-Taste ist so gewählt, dass sie möglichst optimal ein Greifen im echten Leben simuliert, um so intuitiv wie möglich zu sein. (siehe Abbildung in rot markiert)



Eine der zentralen Herausforderungen bestand darin, die Wurfstärke so zu kalibrieren, dass sie kraftvoll und realistisch wirkt, gleichzeitig aber gut von der Gegenseite gefangen werden kann. Auch das Fangen selbst stellte sich als nicht trivial heraus. Insbesondere die „Güte“ eines Fangs – also wie großzügig das Spiel beim Zielen auf den Ball ist – erforderte genauere Justierung. Ergänzt wurde die Interaktion durch ein visuelles Feedback beim „Hovern“ über dem Ball, bei dem der Ball sich rot färbt, um dem Spieler besseres Feedback zur Interaktionszone zu geben.

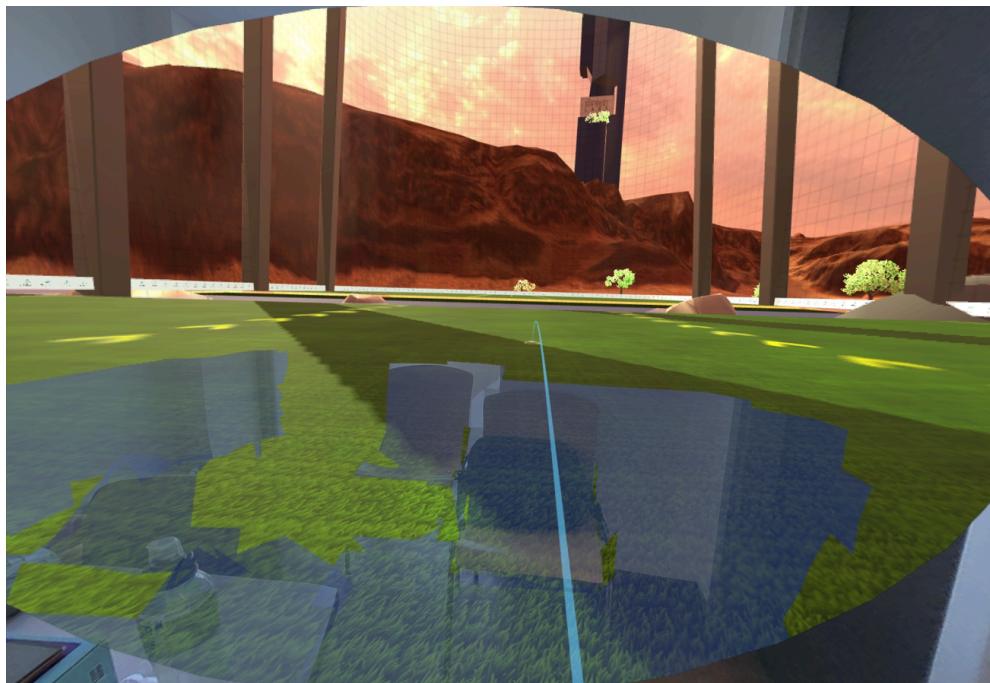


Die finale Implementierung der Wurfstärke basiert auf der durchschnittlichen Geschwindigkeit der Handbewegung in den letzten Frames vor dem Loslassen. Bei besonders schnellen Bewegungen wird zusätzlich ein kraftvoller Soundeffekt abgespielt, um die Intensität des Wurfs auch auditiv zu vermitteln.

Das Werfen und Fangen wird von „NetworkPhysicsInteractable.cs“ und „NetworkPhysicsTransformClient.cs“ umgesetzt.

### 5.2.2 Teleportieren

Im Spiel existiert auch eine Teleportations-Funktion. Wenn der Spieler den rechten Joystick nach vorne bewegt, sieht er eine Linie, die er zielen kann. Beim Loslassen wird er dann an die anvisierte Stelle teleportiert.



Die Teleportation ist mit im XR Interaction Teleport enthalten und funktioniert, indem man einem Objekt das „TeleportationArea.cs“ oder das „TeleportationAnchor.cs“ Skript gibt. Dann kann man dem Skript einen Collider zuweisen, um diesen Collider teleportfähig zu machen.

Im Rahmen des Projekts wurde auch ein „TeleportationCourt.cs“ Skript erstellt. Dieses implementiert „TeleportationArea.cs“ und erweitert es um die Funktion, Teleports zu verhindern, die eine sehr kurze Reichweite hätten.

TeleportationCourt wurde entwickelt, um zur Bewegung im realen Raum zu animieren, da kurze Stecken gelaufen werden sollen. Allerdings ist die Funktion derzeit noch nicht im Einsatz (Siehe Kapitel 8 Roadmap).

### 5.2.3 Kollisionserkennung

Die Erkennung von Treffern, Bounces und Fehlwürfen basiert auf einer Kombination aus Collider-Tags und Logik zur Spielfeldauswertung:

- Der **Boden** besitzt einen Collider mit dem Tag „CourtArea“ und zählt nur dann ein Aufkommen (Bounce) des Balls, wenn der Ball auf der gegnerischen Spielfeldhälfte landet.

- Der **Fluss** ist mit einem „Hindernis“-Tag versehen – landet der Ball dort, erhält der Gegner sofort einen Punkt.
- Eine „BoundingBox“ umgibt das Spielfeld, um Würfe „out of bounds“ zu erkennen. Auch hier wird ein Punkt für den Gegner vergeben.
- Die **Wand** reflektiert den Ball zurück, ohne dabei als gültiger Bounce zu zählen – sie dient also als neutrale Rückprallfläche.

Im Skript „BallScoring.cs“ wird die Kollision des Balls wie gewünscht umgesetzt.

#### **5.2.4 Punktesystem (Scoring, visuelles Feedback)**

Das Spiel ist auf sieben Punkte angesetzt. Punkte werden in zwei Fällen vergeben:

1. Wenn der Ball **drei Mal beim Gegner aufkommt**, ohne dass er gefangen wird.
2. Wenn der Gegner **in den Fluss** oder **außerhalb des Spielfelds** wirft.

Das Skript „BallScoring.cs“ erkennt einen erreichten Punkt und ruft „ScoringManager.cs“ auf, welches dann den Punkt vergibt.

Feedback wird über zwei zentrale Kanäle vermittelt:

- **Auditiv** durch passende Soundeffekte bei Punkten oder Fehlern (siehe Sounddesign).
- **Visuell** durch ein Scoreboard, das in der Mitte des Spielfelds prominent platziert ist und sich bei jedem Punktestand automatisch aktualisiert.

### **5.3 Zusätzliche Spielfeatures**

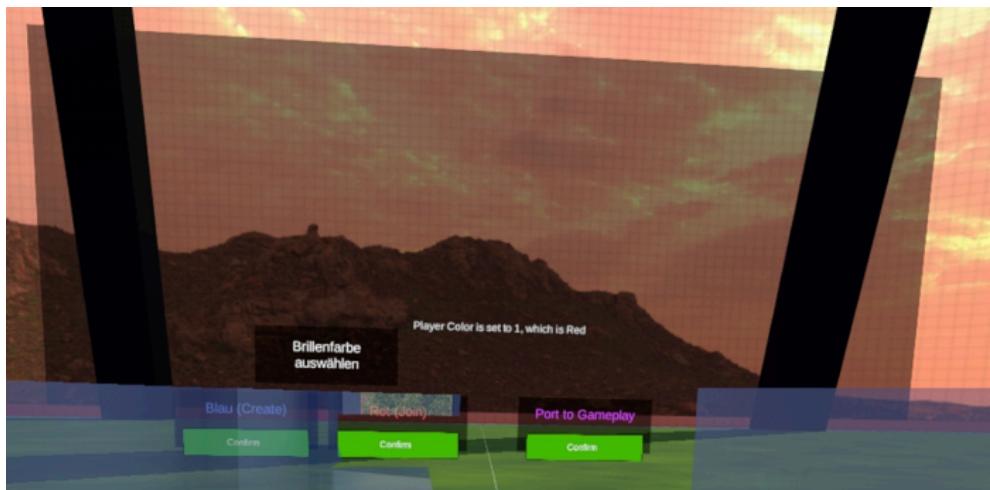
#### **5.3.1 Startup- & Config-Szenen**

Um die Umrüstzeit bei Spielerwechseln zu minimieren, sind den Brillen per Unity PlayerPrefs feste Farben & damit feste Netzwerkrollen zugewiesen.

Um das zu realisieren wurde eine Startup und eine Config Szene eingebaut. Zum Start des Spiels wird immer zunächst die Startup-Szene geladen. Diese enthält nur ein Objekt mit den Skripten „StartupScene.cs“ und „PlatformModeInitializer.cs“. Für jeden Spielstart, wird per „PlatformModeInitializer.cs“ per Unterscheidung der Build Plattform ein PlayerPrefs gesetzt, ob es sich um einen PC-Build (Windows, Mac, Linux) oder

einen Android-Build (VR-Brille) handelt. So kann später zwischen Spieler und Spectator unterschieden werden (siehe Spectator View).

Falls eine Brille in PlayerPrefs keine feste Farbe (Blau oder Rot) gesetzt hat, wird sie statt in die Spielszene zunächst in die Config-Szene geladen. Diese Szene ist nur für Admins bzw. Developer gedacht und dient derzeit lediglich zur Farbauswahl, kann aber bei Bedarf um Adminfunktionen erweitert werden. Der Admin sieht hier 3 Buttons und ein Logfenster, kann dann auf einen der beiden Farbzweisungs-Buttons klicken. Hierzu wird auf dem Logfenster Feedback angezeigt. Danach kann er mit dem dritten Button die Gameplay Szene laden, als wäre das Spiel normal gestartet. Diese Logik findet sich im „AssignPlayerColor.cs“ Skript.



Dieses Prozedere erlaubt die Verwendung eines einzelnen „Spiel starten“ Buttons für beide Spieler, statt einem „Host“ und einem „Join“ Button, von denen der Spieler nur einen drücken muss, was einen Spielstart eines Unerfahrenen Spielers wesentlich erleichtert.

Um eine die Farbe einer Brille zurückzusetzen bzw. zu ändern, muss der Nutzer die Einstellungen der Brille öffnen und unter dem Tab „App-Verwaltung“ die Daten des Spiels löschen (Siehe Kapitel 9 Bedienungsanleitung & Setup).



### 5.3.2 Sounddesign

In GrandSlamVR sind an verschiedenen Stellen Soundclips eingebunden.

Es existieren zum einen Gameplay & Scoring Sounds, zu diesen gehören:

- Punkt gewonnen/verloren
- Match gewonnen/verloren
- normaler Wurf
- starker Wurf
- ein Flugsound des Balls
- ein Bouncesound des Balls

Zudem existiert ein Ambientsound der permanent sowohl für die Spieler, als auch für den Spectator hörbar sind.

Ferner gibt es in der Spectatorview spezielle Soundclips. Diese beinhalten:

- Punkt für das blaue/rote Team
- Matchpoint für das blaue/rote Team
- Sieg für das blaue/rote Team

Die Sounds werden von „ScoringManager.cs“ ausgelöst (siehe Architekturdiagramm)

### **5.3.3 Haptisches Feedback**

Zusätzlich zu den Sounds wird auch an den wichtigsten Stellen im Spiel ein haptisches Feedback über die Controller zurückgegeben. Dies geschieht über zwei Methoden: Die erste Methode nutzt `SendHapticImpulse`, mit der man durch Angabe von Dauer, Stärke und Frequenz ein haptisches Feedback senden kann. Die zweite Methode `SendHapticBuffer` verwendet einen AudioClip und übersetzt diesen in haptisches Feedback. Beide Methoden werden im Script `NetworkPhysicsInteractable.cs` verwendet.

### **5.3.4 Siegbildschirm**

Wenn das Spiel vorüber ist, werden sowohl für beide Spieler, als auch für den Spectator ein Endbildschirm angezeigt der angibt, welche Seite (Blau oder Rot) gewonnen hat.

Dies ist realisiert per Canvas, das bei Rundenende eingeblendet wird und je nach Sieg von Blau oder Rot entweder eine Komponente „VictoryBlue“ oder „VictoryRed“ sichtbar schaltet.

Die Logik hiervon ist im Skript „FinalScreen.cs“ implementiert.

### **5.3.5 Ballindikator**

In der Projektszene findet sich, gebunden an das Ball-Objekt, ein Ballindikator-Objekt. Dieses soll dem Spieler helfen, die Position des Balles besser verfolgen zu können und besteht aus zwei Teilen: die Indikatorfläche und der Indikatorring.

Die Logik des Indikators wird in dem „BallIndicator.cs“ Skript verwaltet, welches auf dem Indikator selbst liegt.

Das Skript setzt die Position beider Indikator Objekte permanent auf dieselben X und Z Positionen des Balls mit Y Koordinate 0. So spiegelt der Indikator zu jedem Zeitpunkt die Position des Balls auf dem Boden wider.

Zudem skaliert das Skript, je nach Höhe des Balls, die Größe des Indikatorrings. Um auch Informationen über die Höhe des Balls zu vermitteln.

So kann der Spieler auf dem Boden zum einen die Position des Balles aus der Position des Indikators, zum Anderen die Höhe des Balles an der Größe des Rings ablesen.

### 5.3.6 Developer Codes

Um das Spiel im Rahmen des Open-Campus möglichst reibungslos spielen zu können, war es essenziell Buttonkombinationen zu haben, die bestimmte Konfigurationsschritte überspringen bzw. in bestimmte Zustände laden können. Daher wurde im Laufe der Projektentwicklung wurden mehrere Entwickler-codes eingebaut, um effizienter zu Testen bzw. das Spiel schneller neuladen zu können:

- Ein Laden des Tutorials: Auf dem Linken Controller beide Trigger, beide Tasten (X, Y) und den Stick für ca. 2 Sekunden eindrücken.



- Ein Teleportieren des Balls auf der eigenen Spielfeldseite: Auf beiden Controllern alle Buchstaben-Tasten (A, B, X, Y) und den linken hinteren Trigger drücken.



### **Neue Developer Codes anlegen**

Um neue Developer Codes anzulegen, muss ein neues Script erstellt werden, welches von `ButtonCombo.cs` erbt.

Um neue Developer Codes zu erstellen, muss ein neues Script angelegt werden, welches von der Basisklasse `ButtonCombo.cs` erbt. Das neu erstellte Script kann dann die Methode `TriggerEvent()` überschreiben. Diese Methode wird automatisch aufgerufen, sobald die festgelegte Tastenkombination erkannt und gedrückt wird. Hier kann die spezifische Funktionalität implementiert werden, die der Developer Code auslösen soll. Eine Beispielimplementation hierfür ist `ButtonComboRespawnBall.cs`.

## 5.4 Evolution der Mechaniken

- Zu Beginn wurde das Spiel mit einer Schlägermechanik konzipiert. Schnell stellte sich jedoch heraus, dass diese Interaktionsform in der VR-Umgebung zu kompliziert war und den Spielfluss hemmte. Aus diesem Grund wurde die Entscheidung getroffen, sich auf eine reine Wurf-/Fangmechanik zu konzentrieren.
- Parallel dazu wurde die Zahl der erlaubten Bounces beim Gegner von zwei auf drei erhöht, um das Spiel etwas zugänglicher und flüssiger zu gestalten.
- Es wurde zusätzlich der Fluss in der Mitte des Spielfelds eingebaut, um zu verhindern, dass Spieler zu flache, kurze Bälle werfen können, die schwer zu fangen sind.
- Auch war zu Beginn der Entwicklung für beide Spieler zum Spielstart zwei Buttons zu sehen „Host“ & „Join“. Hier musste ein Spieler „Host“ ein Spieler „Join“ drücken, was Verständigung erforderte. Dieses System wurde von dem Config Szenen System abgelöst (siehe Startup- & Config-Szenen)

## 5.5 Spectator View

Im Rahmen des Projekts war eine Zuschauersicht enorm wichtig, um bei einem öffentlichen Spielen, wofür das Projekt ausgelegt war, bei Wartenden Langeweile vorzubeugen und Vorfreude auf das Spiel zu erzeugen.

Für die Zuschauerperspektive wurde eine separate Kamera-Logik implementiert.

Beim Start des Spiels unterscheidet das „PlatformModeInitializer.cs“ Skript die Build-Plattform: Wird das Spiel unter Windows oder Mac ausgeführt, wird eine PlayerPreference gesetzt (siehe Startup), die diese Ausführung als Spectator identifiziert.

So kann für den Spectator eine separate Kamera gesetzt werden. Diese Kamera nutzt das Unity **Cinemachine-System** inklusive „Camera Brain“ und mehreren virtuellen Kameras, um Kamerafahrten und Perspektivwechsel zu ermöglichen. Es findet sich im Projektinspektor mit dem Namen „CameraSystem“.

Hier existieren folgende Kameras:

- Eine totale von der Mitte
- Zwei totale aus den jeweiligen Ecken des Spielfelds
- Eine „Rivercam“ in der Mitte des Flusses am Boden
- Zwei Ballkameras, die den Ball aus zwei unterschiedlichen Perspektiven verfolgen

Zwischen diesen Kameras wird in kalibrierbaren Abständen zufällig geschaltet („CameraSwitching.cs“).

Punktgewinne lösen eine automatische Kamerafahrt zum Scoreboard aus, die für ca. zwei Sekunden den neuen Punktestand anzeigt.

Der Zuschauer tritt als normaler Client dem Spiel bei und erhält alle Netzwerk-Updates in Echtzeit, wodurch eine saubere Synchronisation zwischen Spielfluss und Zuschaueransicht gewährleistet ist (Details zur Netzwerkstruktur siehe Netzwerkarchitektur).

In der Spectator Kamera gibt es außerdem am unteren Rand noch eine UI, die den Spielstand anzeigt, damit ein Zuschauer zu jedem Zeitpunkt den Spielstand im Blick hat. Dies ist bei den Spielern nicht der Fall, da das für Spieler die Immersion verringern würde.



Nils Henzen

## **5.6 Tutorial**

Das Ziel des Tutorials ist es, den Nutzern einen schnellen Einstieg zu ermöglichen, da beim Open-Campus nur wenig Zeit für einzelne Personen zur Verfügung steht.

### **5.6.1 Vermittelte Grundmechaniken**

Es werden nur grundlegenden Steuerungsmechaniken vermittelt. Diese werden durch die Darstellung eines Controllers mit blinkenden Elementen verdeutlicht:

- Bewegen (in echt und mit Controller)
- Werfen (Button)
- Teleport (Stick)
- Button-Interaktion (ebenfalls mit dem Werfen-Button, für mehr Einheitlichkeit)

Ein grober Ablauf: Zum Ball laufen -> Ball aufnehmen und halten -> Ball in markierten Bereich werfen -> In einen weiteren Bereich teleportieren -> Zum Button laufen -> Button drücken.

### **5.6.2 Anpassungen durch Nutzer-Feedback**

Durch das User-Testing wurden wichtige Anpassungen vorgenommen. Beispielsweise wurde ein ursprünglich vorhandener, aber als unnötig empfundener Hindernisgraben entfernt. Des Weiteren erfolgte eine Aufteilung des Tutorials in zwei separate Teilbereiche: einen für den Lauf- und Wurfbereich und einen weiteren für den Teleport- und Button-Bereich.

### **5.6.3 Audiovisuelles Feedback**

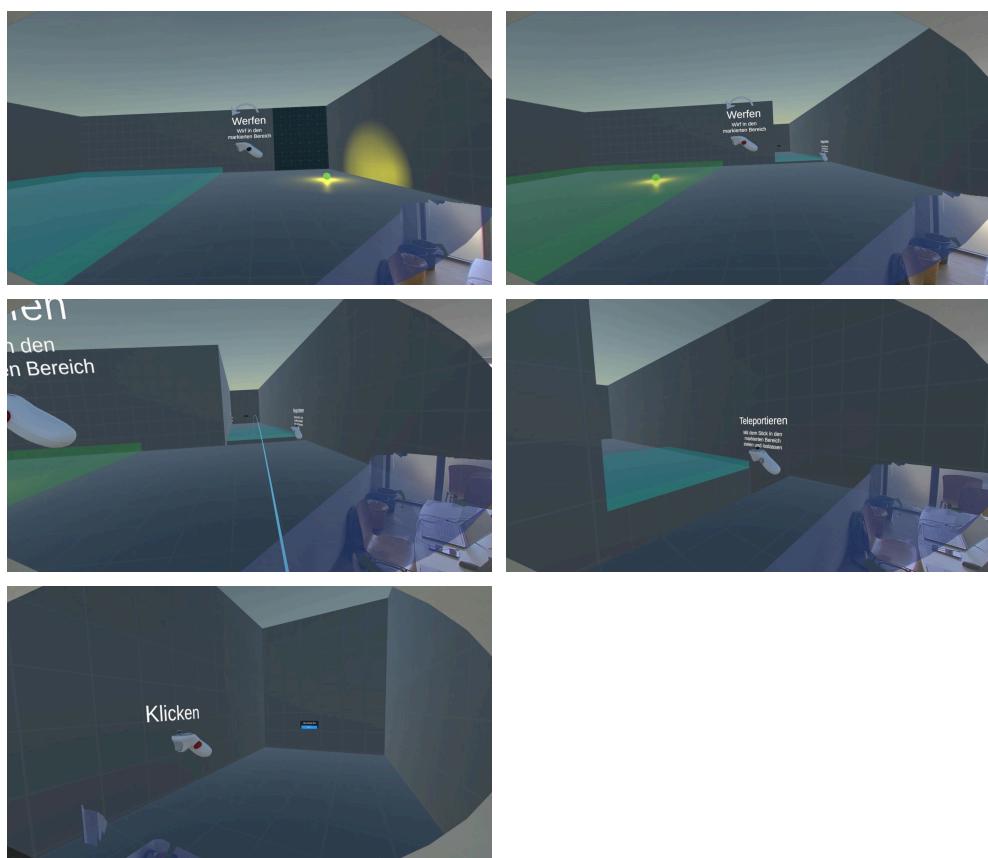
Beim Vervollständigen der Ziele kann der Nutzer durch Farb- und Materialänderungen sowie Audio-Feedback erkennen, wenn eine Station abgeschlossen ist. Der zu hörende Punktesound wird auch im Hauptspiel als Belohnungssound eingesetzt, um eine konsistente Nutzererfahrung zu gewährleisten. Das Skript PlayAudioAfterTimeInArea.cs ist ein Beispiel dafür, wie eine Ballüberprüfung funktioniert.

### **5.6.4 Bodenhöhenanpassung**

Ein wiederkehrendes Problem bei den Pico 4 Ultra Brillen ist die automatische Bodenhöhenerfassung. Die Brillen sind normalerweise in der Lage, die Bodenhöhe automatisch durch die Tiefenkamera zu erkennen und diese dann als

Standard festzulegen. Allerdings ist es nicht möglich, diese Bodenerkennung manuell auszulösen, ohne dabei das Spiel neu zu starten. Für den Open-Campus war es uns deshalb wichtig, einen Workaround zu finden, mit dem Spieler einfach nur durch Betreten des Tutorials die richtige Bodenhöhe haben. Dieser Workaround ist im Skript HeightRecalibrator.cs zu finden und besteht im Grunde genommen nur aus manuellem Umstellen der Bodenhöhenmessungsmethode der Brille. So wird zuerst der TrackingOriginMode auf Device (automatische Erkennung) und anschließend auf Floor (manuelle Höheneinstellung) gewechselt. Diese Abfolge bewirkt, dass die Brille die aktuellen Maße der tragenden Person für die Bodenhöhe nutzt.

#### 5.6.5 Screenshots



**Abb. 6:** Screenshots aus dem Tutorial

## 6 Raumorientierung

### 6.1 Anforderungen

#### 6.1.1 Grundlegende Anforderungen und Herausforderungen

Ein Hauptziel war, dass Nutzer ohne Angst den echten Raum navigieren können. Es war uns wichtig, dass sie sich frei bewegen können, zum Beispiel um einem Ball hinterherzulaufen. Dafür mussten sie unserer Implementierung voll vertrauen, was den Spielspaß deutlich erhöht. Wir planen, das Spiel am Open Campus vorzustellen, wo viele Besucher zum ersten Mal eine VR-Brille tragen werden. Daher ist es entscheidend, dieses Vertrauen von Anfang an aufzubauen.

Anfangs gab es auch die Anforderung, dem Nutzer das Gefühl zu geben, in einem größeren Raum zu stehen, als er tatsächlich ist. Die Umsetzung war eine Herausforderung, da solche Systeme noch nicht weit verbreitet sind. Obwohl es einzelne wissenschaftliche Arbeiten zu Teilbereichen gibt, existieren kaum Projekte, die diese Anforderungen umfassend erfüllen. Auch wir sind hier an die Grenzen des technisch Machbaren gestoßen.

#### 6.1.2 Erwogene Konzepte

In der ersten Planungsphase haben wir verschiedene Konzepte zur Raumorientierung in Betracht gezogen, die aber nicht weiterverfolgt wurden:

- **Omnidirektionales Laufband:** Damit wäre es möglich gewesen, dass sich Nutzer frei bewegen, ohne sich in der echten Welt zu bewegen. Dies hätte jedoch hohe Kosten verursacht.
- **Plattform-Konzept:** Diese Idee entstand aus einem alten Map-Konzept mit fliegenden Inseln. Der Spieler sollte auf einer Plattform spawnen, die genau die Maße seines realen Raums hat. Dieser Ansatz hatte jedoch Nachteile, zum Beispiel bei der Darstellung kleinerer Objekte oder für Personen mit Höhenangst.
- **Redirected Walking:** Hierbei krümmt sich das Spielfeld unterbewusst in die Richtung, in der in der echten Welt am meisten Platz ist. Ein Paper

zeigte, dass der dafür benötigte Platz zu groß wäre, um es effektiv am Open Campus zu nutzen.

- **Teleport-Rotation:** Die Arena rotiert um den Spieler, wenn dieser sich teleportiert. Dieser Ansatz erwies sich jedoch als sehr verwirrend für den Nutzer, da er seine Orientierung verlieren konnte, was auch zu Motion Sickness führen kann.

## 6.2 Spatial Mesh

Das Spatial Mesh scannt den Raum mit dem Tiefensensor der Pico-Brille. Diese Daten werden dann in die virtuelle Umgebung übergeben.

### 6.2.1 Funktionsweise

In Unity wird das Kamerabild der Pico mit einer Maske bzw. einem Material über das Mesh gelegt. So entsteht ein dreidimensionales Bild, in dem der Nutzer sich bewegen und dabei die Außenwelt relativ genau beobachten kann. Diese virtuelle Umgebung wird über das eigentliche Spielgeschehen gelegt. Das Material ist jedoch transparent genug, damit der Spieler das Spiel trotzdem sehen kann. Zudem ist die Höhe des Meshes so angepasst, dass Boden und Decke nicht erfasst werden, was der Immersion zugutekommt. Dadurch dass der Nutzer nun Hindernisse aus der echten Welt in seiner VR-Brille wahrnimmt, kann dieser sich freier und sorgloser bewegen.

### 6.2.2 Visuelle Darstellung

Wir haben uns bewusst gegen ein echtes, solides Mesh entschieden. Hier ging es hauptsächlich darum, dass der Spieler wichtige Informationen aus dem echten Leben noch erkennen kann. Bei einer soliden Mesh-Box müsste er überlegen, ob das ein Tisch oder nur ein Artefakt von jemandem ist, der zuvor dort langgelaufen ist.

- **Initial (Rot):** Zuerst hatten wir Rot für das Material gewählt. Dieses fühlte sich aber oft „im Weg“ an und hinderte die Immersion. Auch der Barrierefreiheit halber (rotes Material auf grünem Grasboden) haben wir die Farbe gewechselt.
- **Final (Blau):** Eine blaue Farbe löste diese Probleme. Sie hebt sich besser vom Boden ab, ist aber trotzdem nicht so aufdringlich. Dies führte dazu,

dass die Tester das Mesh oft einfach vergaßen und es nur in kritischen Momenten bemerkten.

### 6.2.3 Technische Umsetzung

Der `SpatialMeshManager.cs` ist eine angepasste Kopie der vorimplementierten Spatial Mesh Lösung des Pico SDK. Die Code-Anpassungen belaufen sich hierbei auf die dynamische Mesh-Position, die mit dem Spieler mitgeht.

### 6.2.4 Anpassbare Parameter

Das `SpatialMeshManager.cs`-Skript enthält verschiedene Parameter, die das Verhalten und die Darstellung des Spatial Mesh beeinflussen:

- **Headset Offset:** Transform-Referenz für den Headset-Offset.
- **Origin Offset:** Transform-Referenz für einen zusätzlichen Ursprungs-Offset.
- **Mesh Prefab:** Das GameObject-Prefab, das für die Darstellung der einzelnen Mesh-Blöcke verwendet wird.
- **Object Pool MaxSize:** Die maximale Anzahl von Mesh-GameObjects, die im Objekt-Pool gehalten werden.

### 6.2.5 Probleme und Limitationen

Das Spatial Mesh reicht alleine nicht aus, um ein angstfreies Bewegen zu ermöglichen.

**Feine Strukturen:** Dinge wie Wände werden zwar fehlerfrei erfasst, aber Kabel oder kleinere Bälle auf dem Boden gar nicht. Das kann zu gefährlichen Situationen führen.

**Durchsichtige Objekte:** Fenster sind ein Problem, da sie im Mesh oft eher wie ein Loch in die echte Welt dargestellt werden.

**Artefakte:** Es ist beim Testen vorgekommen, dass an Plätzen, an denen zuvor ein Stuhl oder eine Person stand, auch nach dem Entfernen noch ein Teil des Meshes gerendert wurde. Dies führte zu Verwirrung bei Testern.

**Feste Ausrichtung und Höhe:** Technisch bedingt lässt sich das Spatial Mesh leider nicht stark anpassen. Es besitzt immer eine feste Höhe und Rotation. Dies bedeutet, dass die virtuelle Map/Szene sich an das generierte Mesh anpassen muss, indem ihre Höhe auf Null gesetzt und die Rotation entsprechend ausgerichtet wird. Dies erfordert eine sorgfältige Planung des Leveledesigns.

#### 6.2.6 Screenshots

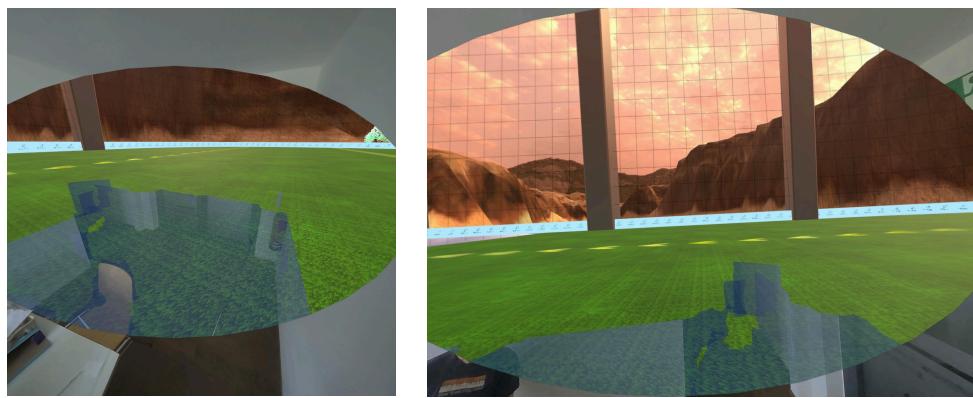
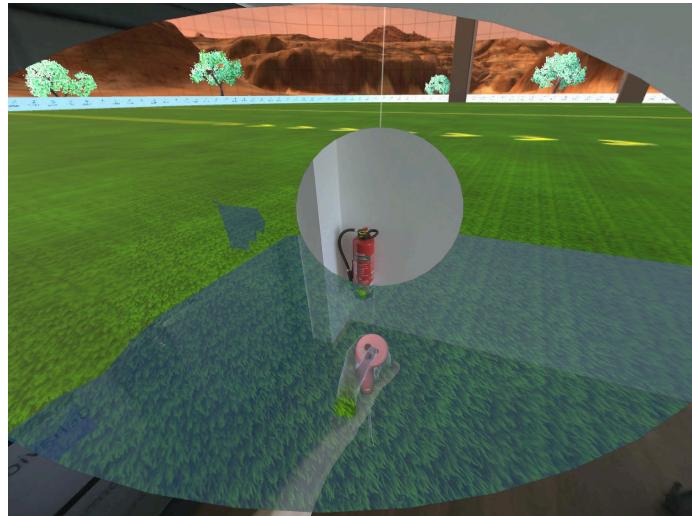


Abb. 7: Spatial Mesh im Spiel

### 6.3 Zusätzliche Orientierungshilfen

#### 6.3.1 Hand-Spion

Der Hand-Spion ist eine Funktion, mit der der Spieler durch ein rundes Fenster in die reale Welt blicken kann.



**Abb. 8:** Hand-Spion im Spiel

**Aktivierung:** Die Funktion wird aktiviert, indem der Spieler seine linke Hand/Controller über einen gewissen Schwellenwert hält (ca. Bauchhöhe).

**Nutzen:** Mit dem Spion kann sich der Nutzer vor dem Spiel und in ruhigen Phasen im Raum umsehen und sich von seiner Sicherheit überzeugen. Er kann auch zur schnellen Vergewisserung zusätzlich zum Spatial Mesh verwendet werden. Tester haben die Einfachheit und intuitive Nutzung des Spions für schnelle Blicke besonders gelobt, was seine Effektivität unterstreicht.

**Technische Grundlage:** Die Pico-Brille besitzt eine allgemeine Passthrough-Funktion, die es ermöglicht, die reale Umgebung durch die Kameralinsen zu sehen. Diese Funktionalität bildet die technische Basis für den Hand-Spion, der quasi als „Lupenmaterial“ einen Ausschnitt der Durchsicht darstellt.

### 6.3.2 Durchschauen am Rand der Brille (Peripheres Fenster)

Der äußere Rand der Brille bietet Durchsicht, was unterbewusst zu einer besseren Orientierung, vor allem beim Rotieren, führt. Dies wurde durch Tester in unserem User-Testing bestätigt.

**Limitation für Brillenträger:** Da der Rand der Brille gleichzeitig auch den Rand der VR-Brille verdeckt, haben viele Brillenträger dieses Feature nicht. Je nach Brillenart kann der von Pico mitgelieferte Brillenaufsatz benutzt werden. Für Personen, bei denen der Aufsatz nicht ausreicht, haben wir aktuell keine Lösung, da die Zielgruppe dafür relativ klein sein sollte und dies weitere Konfiguration beim Open Campus bedeuten würde.

## 6.4 Movement Scaling

Im User-Testing haben wir festgestellt, dass Spieler sich zu wenig bewegt haben. Einerseits hatten sie Angst, andererseits war es auch nicht nötig/belohnend, da der Teleport schneller und einfacher war.

### 6.4.1 Konzept und Umsetzung

Das haben wir geändert, indem wir den Teleport beschränkt und die Eigenbewegung verbessert haben. Wir haben die Grundgeschwindigkeit erhöht und eine Sprint-Mechanik eingebaut, bei der der Spieler durch Bewegen seiner Arme, ähnlich wie im echten Leben, in einen Sprint-Modus gelangt.

Wichtig hierbei ist vor allem die Beschleunigung und das Abbremsen der Geschwindigkeit, um Motion Sickness zu vermeiden. Das Ziel war, den Sprint so schnell wie möglich zu machen, ihn aber trotzdem noch glaubhaft zu halten. Das hilft der Immersion und wirkt gegen Motion Sickness.

Der `PhysicalWalkScaler.cs` setzt diese Logik um. Entscheidend sind `scaleAccelerationSpeed` und `scaleDecelerationSpeed`. Wir interpolieren den Skalierungsfaktor sanft, wobei das Abbremsen schneller ist als das Beschleunigen, um dem Spieler die Kontrolle zu geben.

### 6.4.2 Frame-basierte Sprint-Erkennung

Um den Sprint zu erkennen, analysieren wir die Bewegung der Controller über die letzten `runningDetectionFrames`. Wir speichern die Positionen in einer Liste und summieren die zurückgelegte Distanz. Überschreitet die Summe den `runningDetectionThreshold`, gilt der Spieler als sprintend.

### 6.4.3 Anpassbare Parameter

Das PhysicalWalkScaler.cs-Skript bietet eine Reihe von Parametern zur Feinjustierung des Bewegungserlebnisses:

- **Vr Camera/Player Root:** Referenzen auf die VR-Kamera und das Root-Objekt des Spielers. Diese müssen im Unity Editor korrekt zugewiesen werden, um die Kopfbewegung zu verfolgen und die Spielerbewegung umzusetzen.
- **Default Scale Factor:** Multiplikator für die normale Gehgeschwindigkeit.
- **Include Y Movement:** Bestimmt, ob die vertikale Bewegung (Y-Achse) ebenfalls skaliert wird.
- **Enable Running Detection:** Aktiviert oder deaktiviert die gesamte Sprint-Mechanik.
- **Running Scale Factor:** Multiplikator für die Sprintgeschwindigkeit.
- **LeftController / RightController:** Referenzen auf die Controller-Transforms zur Sprint-Erkennung.
- **Running Detection Threshold:** Schwellenwert für die Sprint-Erkennung.
- **Running Detection Frames:** Anzahl der Frames, über die die Controller-Bewegung für die Sprint-Erkennung gemessen wird.
- **Scale Acceleration Speed:** Geschwindigkeit, mit der der Skalierungsfaktor erhöht wird (z.B. von Gehen zu Sprinten).
- **Scale Deceleration Speed:** Geschwindigkeit, mit der der Skalierungsfaktor reduziert wird (z.B. von Sprinten zu Gehen).

## 7 Map-Design & Raumorientierung

### 7.1 Visuelle Konzepte und MVP-Basis

Als unser erstes MVP lief, trafen wir uns im Team zu einem Brainstorming. Wir legten die wichtigsten Punkte für unsere Spielwelt fest. Der echte Raum muss immer erkennbar sein, damit sich die Spieler sicher bewegen. Die Größe der Arena soll durch klare Referenzpunkte wie Markierungen auf dem Boden oder besondere Objekte sichtbar werden. Wir wollten keine dunklen Szenen, damit vor allem Kinder sich wohlfühlen. Auch Übelkeit oder Angst vor Höhen sollte vermieden werden. Gleichzeitig durfte die Welt nicht zu langweilig sein, sondern sollte die Möglichkeiten von VR zeigen. Aus diesen Ideen fertigten wir erste Skizzen an und hängten sie an unser Visionboard. Dort verglichen wir verschiedene Umgebungen und diskutierten Vor- und Nachteile.

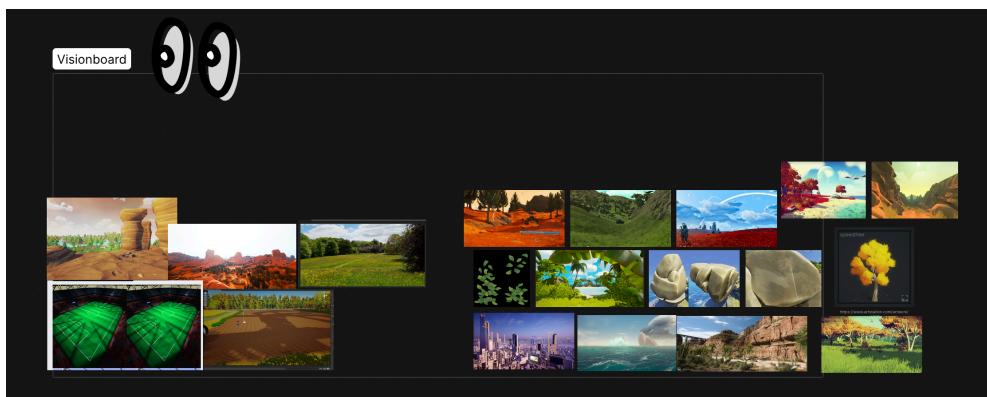


Abb. 9: Visionboard VR Tennis

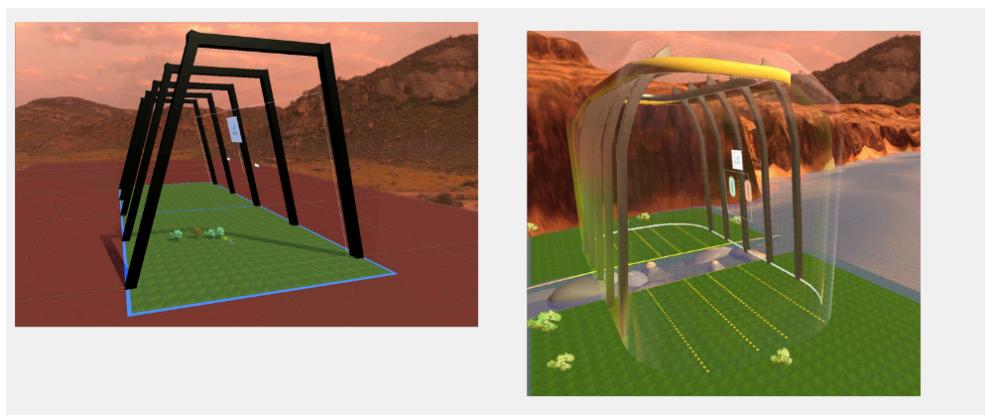


**Abb. 10:** Skizzen VR Tennis

## 7.2 Konzeptphase: Stadion vs. Canyon

Im ersten Prototypen entschieden wir uns für eine **Stadion-Umgebung** mit Tribünen, einem Turm samt Parkplatz und dekorativen Straßenelementen. Ziel war es, den Spieler vertraute Anhaltspunkte zu bieten. In frühen Performance-Tests zeigte sich jedoch schnell: Die vielen Details führten zu instabilen Framerates und lenkten vom eigentlichen Spielgeschehen ab.

Deshalb wechselten wir zu einer reduzierten **Canyon-Welt**: Eine breite Schlucht umrahmt das Spielfeld, lenkt den Blick gezielt aufs Spiel und vermittelt trotzdem ein starkes Raumgefühl. Einzelne Felsen, Bäume und ein zentraler Turm dienen als dezente Orientierungspunkte, ohne visuell zu überfordern.



**Abb. 11:** Stadion vs. Canyon

### 7.2.1 Umsetzungsschritte

Einfaches Netz vs. **Fluss-Trenner** Ursprünglich planten wir ein klassisches **Tennisnetz** zur Spielfeldtrennung. Doch es überzeugte weder visuell noch funktional: Die Sicht auf den Boden dahinter war eingeschränkt, und einige Spieler nutzten es, um den Ball unfair flach darüber zu werfen.

Deshalb ersetzten wir das Netz durch einen breiten **Fluss**. Der Wassergraben wirkt nicht nur moderner und offener, sondern schafft auch eine klare visuelle Trennung. Damit Bälle nicht im Wasser liegen bleiben, haben wir eine unsichtbare Collider-Zone eingebaut: Fällt ein Ball hinein, wird er nach einer Sekunde automatisch an seinen Startpunkt zurückgesetzt. So bleibt das Spiel fair und flüssig.

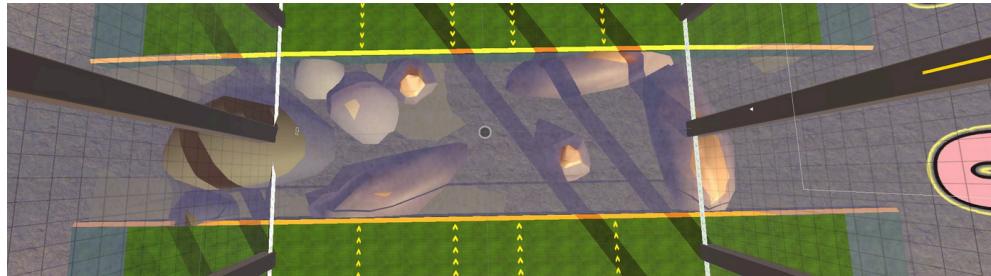


Abb. 12: Fluss - GrandSlam Arena

### 7.2.2 Säulen als Größenreferenz

Anfangs wirkte unsere Arena im Spiel deutlich kleiner, als sie tatsächlich war, weil Orientierungshilfen für das Größenverhältnis fehlten. Unsere Lösung bestand darin, große, sich nach oben verjüngende **Säulen** an den Spielfeldrändern zu platzieren. Wir modellierten diese Säulen in Blender und integrierten sie anschließend als Prefabs in Unity. Direkt nach der Implementierung verbesserten sie das Raumgefühl spürbar und halfen den Spieler, die tatsächliche Ausdehnung der Arena besser einzuschätzen. Um die Tiefenwirkung noch weiter zu verstärken, fügten wir später statische Schattenwürfe hinzu, sodass die Arena sofort deutlich imposanter und majestätischer wirkte.

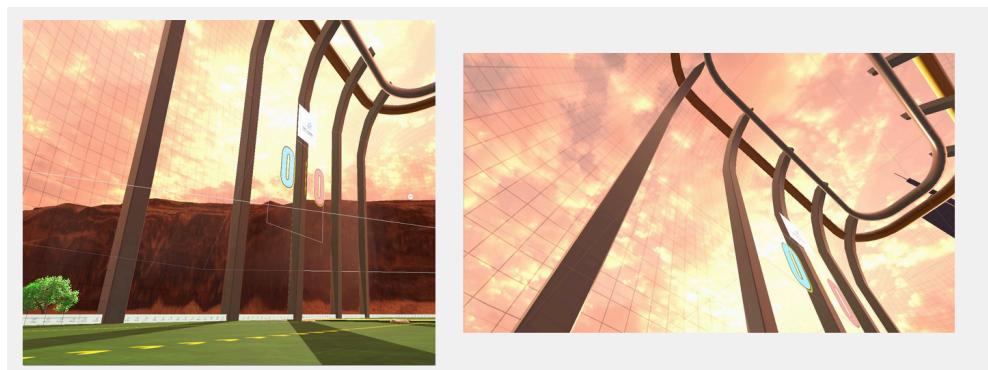
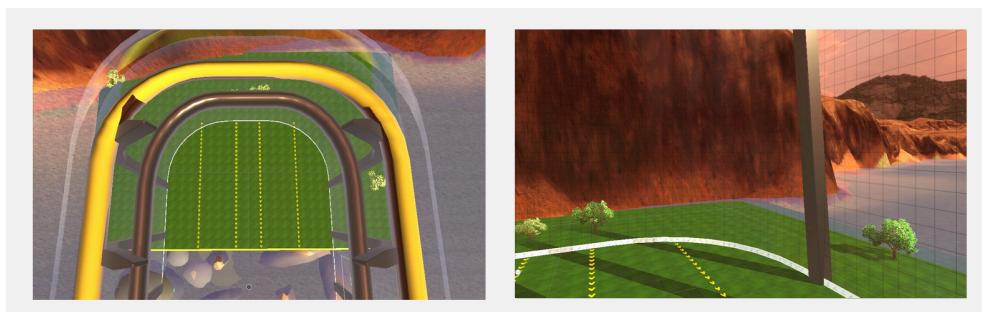


Abb. 13: Säulen - GrandSlam Arena

### 7.2.3 Abrundung der Ecken

Scharfe Kanten wirkten nicht nur unnatürlich, sie sorgten auch für unvorhersehbare Ballabpraller. Deshalb haben wir beschlossen, alle vier Ecken der Spielfläche abzurunden.

Dadurch konnten wir eine realistischere Ballphysik erreichen und das Spiel insgesamt flüssiger gestalten. Viele Tester:innen berichteten, dass sie jetzt leichtere Trickshots landen konnten, die aber immer noch vom Gegner abgewehrt werden konnten. Das machte das Spiel spannend, ohne frustrierend zu werden. Außerdem werden Bälle durch die abgerundeten Ecken häufiger automatisch in Richtung Spielfeldmitte gelenkt, was für ein dynamischeres und besser ausbalanciertes Spielerlebnis sorgt.

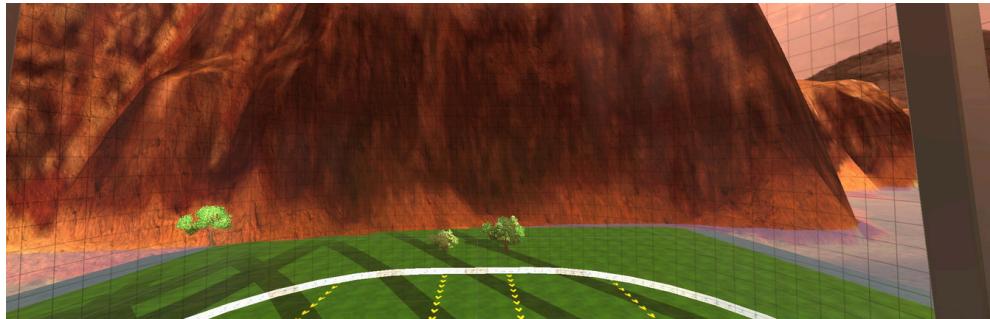


**Abb. 14:** Abrundung - GrandSlam Arena

#### 7.2.4 Gittermuster an der Innenverkleidung

Trotz der Säulen fiel es Spieler immer noch schwer, die Ausmaße der Arena genau abzuschätzen, vor allem in Verbindung mit den abgerundeten Ecken. Auch die Flugbahnen des Balls waren nicht immer klar nachzuvollziehen.

Deshalb haben wir ein dezentes **Gittermuster** an den Innenwänden angebracht. Dieses Raster gibt den Spieler eine zusätzliche visuelle Orientierung und macht es einfacher, Positionen und Bewegungen einzuschätzen, ohne die klare Optik der Arena zu stören.



**Abb. 15:** Gitter - GrandSlam Arena

### 7.2.5 Canyon-LOD-Mesh

Um das ursprünglich geplante Stadion-Setting zu ersetzen, entschieden wir uns für eine **Canyon-Umgebung**. Der Canyon wirkte bewusst neutral und lenkte nicht vom Spielgeschehen ab und vermittelte durch die tiefen Schluchten ein klares Gefühl für Größe, vor allem wenn man vom Turm hinunterschaute.

Ein großer Vorteil war die Performance. Anders als beim Stadion brauchte der Canyon keine aufwändigen Deko-Elemente, um glaubwürdig zu wirken. In Blender haben wir das Gelände mit einer dynamischen Auflösung erstellt. In der unmittelbaren Nähe der Arena zeigt das Mesh viele Details, weiter entfernt wird es stufenweise vereinfacht. Für die Spieler entsteht kein Unterschied, während wir technisch spürbare Vorteile erzielen. Mit der Pico Developer Software haben wir die Framerate gemessen und konnten tatsächlich eine deutliche Verbesserung feststellen. Der Canyon war damit nicht nur optisch passend, sondern auch eine clevere Entscheidung für die Performance.

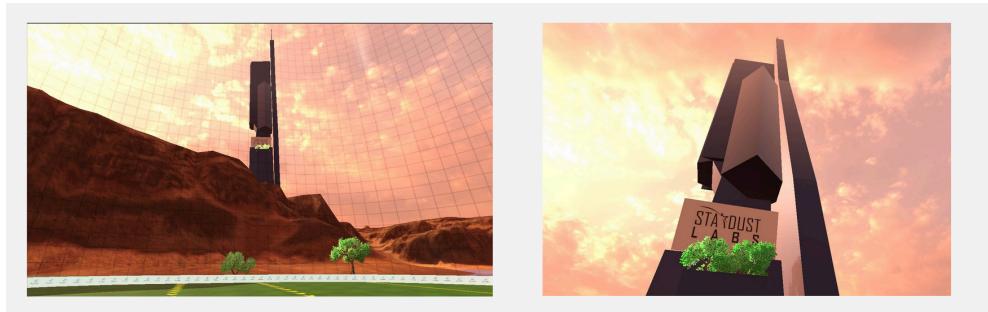
### 7.2.6 Turm als Eyecatcher & Spawnpunkt

Zum Schluss haben wir einen hohen **Turm** am Rand der Schlucht gebaut. Anfangs diente er nur als Orientierungspunkt, später wurde er auch unser Spawnpunkt. Auf dem Balkon des Turms starten die Spieler. Von dort wirkt das Spielfeld erst einmal ziemlich klein, während der Turm riesig erscheint.

Betritt man dann die Arena, kehrt sich dieser Eindruck komplett um. Die Arena fühlt sich plötzlich groß an, und der Turm scheint zu schrumpfen. Das sorgt für einen echten Wow-Effekt, den viele Tester sofort wahrgenommen haben.

Der Turm hat noch einen weiteren Nutzen. Er hilft dabei, die Ausrichtung des Spielfelds besser zu verstehen. Gerade aus der Vogelperspektive ist er sofort

sichtbar und gibt den Spieler:innen eine klare Orientierungshilfe. Visuell auffällig und funktional sinnvoll ist er damit das Highlight unserer Map.

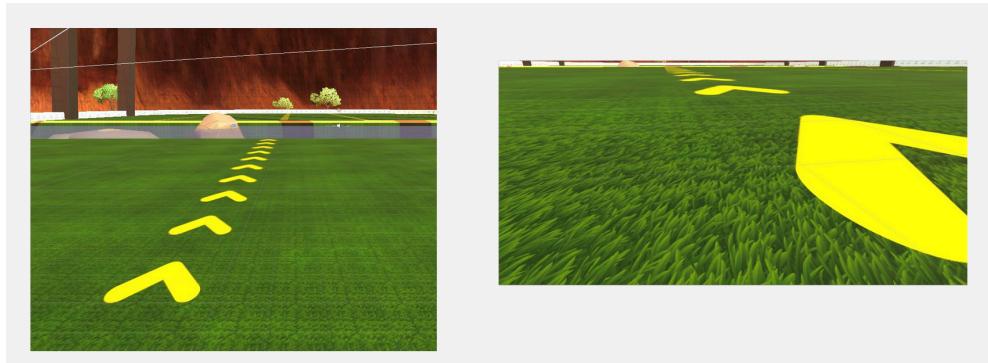


**Abb. 16:** Turm - GrandSlam

### 7.2.7 Orientierungssignifier

Zum Schluss haben wir zwei einfache, aber sehr wirkungsvolle Orientierungshilfen eingebaut. Erstens die **Bodenpfeile**, gelb und dezent auf dem Spielfeldboden (siehe Abb. 17). Sie zeigen klar die Spielrichtung an und helfen besonders nach einem Teleport oder zu Beginn jeder Runde, sich sofort richtig auszurichten.

Zweitens setzten wir auf die **Grastextur** mit einer klaren Wuchsrichtung. Blickt man in die falsche Richtung, wirkt das Gras absichtlich unnatürlich „komisch“. Dieser kleine visuelle Hinweis sorgt dafür, dass man sich automatisch wieder korrekt ausrichtet. Zusammen tragen diese Details stark dazu bei, dass sich das Spielfeld logisch anfühlt und man sich schnell zurechtfindet, ohne lange darüber nachdenken zu müssen.



**Abb. 17:** Orientierungssignifier - GrandSlam

## 8 Testing und Debugging

### 8.1 Test-Strategien

Das Testen in Unity im Zusammenspiel mit VR stellte sich als herausfordernd heraus, insbesondere im Hinblick auf automatisierte Tests. Klassische Unit- und Integrations-Tests erwiesen sich als wenig effizient, da viele Mechaniken auf direkter Spieler-Interaktion beruhen und damit nicht isoliert testbar sind.

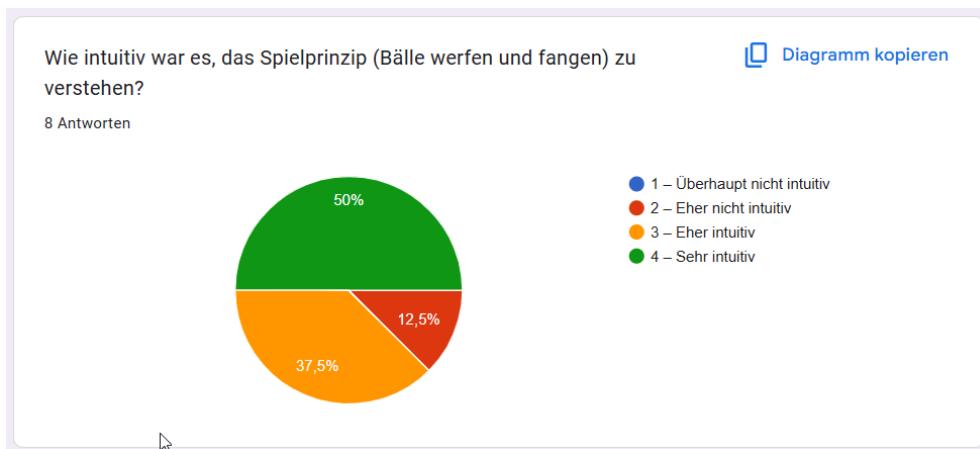
Stattdessen lag der primäre Fokus auf manuellen Test-Sessions, die in verschiedenen Formen durchgeführt wurden, um die Funktionalität, das Spielgefühl und die Benutzererfahrung umfassend zu bewerten.

#### **Interne Teamtests:**

Regelmäßige Tests wurden direkt im Entwicklungsteam durchgeführt. Nach der Implementierung oder Anpassung von Features testeten die Entwickler selbst oder im Pair-Programming-Ansatz, um grundlegende Funktionalität und Integration zu überprüfen. Diese Tests fanden oft mit schnellem „Build-and-Run“-Deployment auf die VR-Brille statt. Besonderes Augenmerk lag hierbei immer auf der Überprüfung der Netzwerksynchronisation.

#### **Umfangreiche User-Testings mit der Kernzielgruppe:**

Zur Bewertung der Benutzerfreundlichkeit, der Intuitivität und des Spielspaßes wurden Test-Sessions mit externen Personen durchgeführt, die der definierten Kernzielgruppe entsprechen. Diese Tests dienten dazu, Feedback von unvoreingenommenen Nutzern zu sammeln, die nicht am Entwicklungsprozess beteiligt waren. Es wurden strukturierte Fragebögen verwendet (Google Forms), um Feedback zu spezifischen Aspekten des Spiels zu erhalten (z. B. zur allgemeinen Bewertung, Intuitivität der Steuerung, Präzision der Interaktionen, Raumorientierung, Schwierigkeitsgrad und Spielspaß).



**Abb. 18:** Auszug aus dem Testformular

### **Wöchentliche Tests mit dem Anfordernden (Product Owner / Professor):**

In regelmäßigen, typischerweise wöchentlichen Terminen wurde der aktuelle Entwicklungsstand dem Anfordernden präsentiert. Diese Tests prüften die Zielerreichung des Projekts und lieferten unmittelbares Feedback.

### **Der Feedback-Loop:**

Ein kritischer Bestandteil der Teststrategie war der strukturierte Umgang mit dem gesammelten Feedback: Feedback aus allen Test-Sessions wurde sorgfältig dokumentiert.

Diskussion und Analyse: Das gesammelte Feedback wurde intern im Team diskutiert und analysiert, um die zugrunde liegenden Probleme oder Verbesserungspotenziale zu verstehen.

Task-Erstellung: Basierend auf der Analyse wurden konkrete Jira-Tasks erstellt, die die identifizierten Bugs, gewünschten Anpassungen oder neuen Features repräsentierten. Diese Tasks wurden mit dem entsprechenden Feedback verknüpft.

Backlog-Management und Priorisierung: Die erstellten Tasks wurden in den Backlog aufgenommen. Das Feedback aus den Tests spielte eine entscheidende Rolle bei der Neupriorisierung des Backlogs, um sicherzustellen, dass die wichtigsten und wirkungsvollsten Aufgaben zuerst angegangen werden.

Vorgang	Pri...	Zu...	Status
VRTENNIS-129 Fenster nur in bestimmten Winkeln	= M	NILS	FERTIG ▾
VRTENNIS-117 Pace aufbauen langsamer machen	= M	NICO	FERTIG ▾
VRTENNIS-118 Säulen Schatten statisch baken	= M	NICO	FERTIG ▾
VRTENNIS-119 Ball indicator größer	= M	NICO	FERTIG ▾
VRTENNIS-127 Jump Button entfernen	= M	NILS	FERTIG ▾
VRTENNIS-120 Angel dicker und Farbe	= M	NICO	FERTIG ▾
VRTENNIS-122 Hover Feedback deutlicher	= M	NICO	FERTIG ▾
VRTENNIS-123 Wurfrichtung manchmal komisch	= M	NICO	FERTIG ▾
VRTENNIS-124 Bug mit Bewegung wird Angel lang fixen	= M	NICO	FERTIG ▾

Abb. 19: In Jira-Tasks umgewandeltes Feedback

Durch diese Kombination aus verschiedenen manuellen Testformen und einem strukturierten Feedback-Loop konnte das Projekt gezielt auf die Herausforderungen der VR-Entwicklung reagieren und die Entwicklung basierend auf realer Spielerfahrung vorangetrieben werden.

## 8.2 Debugging-Strategien

Die Fehlersuche (Debugging) in VR-Multiplayer-Projekten stellt besondere Herausforderungen dar. Standardmäßige Debugging-Methoden, wie das einfache `Debug.Log()` allein, reichen oft nicht aus, um die Ursachen von Problemen in der komplexen Interaktion zwischen VR-Hardware, Spielmechaniken und Netzwerkkommunikation zu identifizieren.

Um diese Herausforderungen zu meistern, wurde ein strukturierter Debugging-Ansatz implementiert, der aus zwei Hauptkomponenten besteht:

- Eine benutzerdefinierte Logging-Klasse (`CustomDebugLogger`): Ermöglicht eine zentralisierte, kontextbezogene und filterbare Ausgabe von Debug-Nachrichten.
- Eine VR-taugliche In-Game-Konsole: Macht Debug-Meldungen direkt auf der Pico VR-Brille sichtbar, ohne dass ein angeschlossener PC für „adb logcat“ benötigt wird.

### **8.2.1 Custom Debug Logger**

Der CustomDebugLogger ist eine statische C#-Klasse, die als zentraler Punkt für alle Debug-Ausgaben im Projekt dient. Anstatt Debug.Log() direkt aufzurufen, werden die Methoden dieser Klasse verwendet.

#### **Struktur der Klasse:**

- Die Klasse CustomDebugLogger.cs dient als Wrapper für die Standard Debug.Log, Debug.LogWarning und Debug.LogError Methoden. Sie fügt zusätzlichen Kontext zu jeder Nachricht hinzu.
- Zentralisierte Steuerung: Durch die statische CurrentLogLevel-Variable kann die Menge der ausgegebenen Nachrichten global gesteuert werden (z.B. nur Fehler und Warnungen in Test-Builds, alles im Entwicklungs-Build).
- Kontextbezogene Ausgaben: Jede Nachricht enthält automatisch den Namen des GameObjects und den Typ des Skripts (MonoBehaviour sender), von dem die Nachricht gesendet wurde. Dies macht die Fehlersuche erheblich einfacher, da man sofort sieht, woher eine Meldung kommt.
- Filterung nach Wichtigkeit (LogLevel): Nachrichten werden in verschiedene Level eingeteilt (None, Error, Warning, Info, Verbose), und über CurrentLogLevel kann eingestellt werden, welche Level ausgegeben werden.

### **8.2.2 Debugging auf der Brille: Die In-Game-Konsole**

Ein wesentliches Problem beim Debugging von VR-Anwendungen auf Standalone-Headsets wie der Pico ist, dass die Standard-Debug.Log-Ausgaben in Builds nicht direkt im Headset sichtbar sind. Man müsste die Brille per USB an einen PC anschließen und adb logcat verwenden, was den Testfluss unterbricht.

Das Asset bietet ein Textfeld, das direkt im VR-Spiel angezeigt wird, und das über ein Skript alle Debug.Log-Aufrufe abfängt und anzeigt.

Da unser CustomDebugLogger intern Standard-Debug.Log() (oder Debug.LogWarning/.LogError) verwendet, werden alle über den CustomDebugLogger gesendeten Nachrichten automatisch von der In-Game-Konsole abgefangen und in der VR-Umgebung angezeigt.

So konnten wir während dem Testen im VR-Spiel auf diesem Textfeld die Debug-Meldungen verfolgen.

### **8.2.3 Audio-Debugging**

Zum Teil wurden auch akustische Signale verwendet, um bestimmte Events im Spiel zu signalisieren (z. B. ein spezifischer Ton beim Ballaufprall, falls er durch die Wand fliegt, oder beim Blockieren eines ungültigen Teleports). Dies lieferte uns unmittelbares Feedback, insbesondere als es darum ging, ob ein Event zum richtigen Zeitpunkt ausgelöst wird.

Hendrik Staab

## **9 Roadmap & Erweiterungsmöglichkeiten**

Im folgenden ist eine Auflistung von Funktionen zu finden, die aus zeitlichen Gründen nicht mehr umgesetzt werden konnten, aber aus Entwicklersicht noch relevant für zukünftige Umsetzung sein könnten

### **9.1 Teleportation einschränken**

Um die Bewegung im realen Raum zu motivieren, war eine Idee innerhalb des Teams, zu verhindern, dass der Spieler kurze Strecken teleportieren kann, da er diese laufen soll. Dies wurde auch im Skript „TeleportationCourt.cs“ umgesetzt. Allerdings steht die Mindestreichweite des Teleports noch auf 0, da keine Zeit zur Justierung war. Dieser sollte mit der Länge der Fangreichweite abgeglichen werden, um Bewegung im realen Raum zu belohnen, aber nicht für zu große Reichweiten verpflichtend zu machen.

### **9.2 Weitere Developer Shortcuts**

Während dem Einsatz im Open-Campus wurde ersichtlich, dass weitere Buttonkombinationen von Vorteil sein könnten. Beispielsweise wäre eine Kombination zum Spiel zurücksetzen gut gewesen, die die Verbindung trennt und den Spieler auf den Turm teleportiert.

### **9.3 Hintergrundmusik**

Aktuell befindet sich im Spiel keine Hintergrundmusik, sondern ein Ambient-sound der in der Arena läuft. Ein Komponieren von Musikstücken, die zum Beispiel vor dem Spiel, nach dem Spiel oder während dem Spiel abgespielt werden könnten, würde das Spielerlebnis positiv beeinflussen.

## **9.4 Tutorial verbessern**

Im Einsatz im Rahmen des Open-Campus wurde ersichtlich, dass auch nach mehrmaligem Testen & Verbessern das Tutorial vor allem für Personen ohne Vorerfahrung mit Videospielen noch zu unintuitiv und verwirrend ist. Das Tutorial muss noch weiter vereinfacht werden. Dazu könnte man:

- Die Notwendigkeit des Laufens mit dem linken Stick entfernen
- Die Richtung in die der Ball zu werfen ist eindeutiger hervorheben
- Der Fortschritt im Tutorial klarer dargestellen
- Das Tutorial im Design der Arena halten
- Im Tutorial überall teleportieren erlauben
- Das Tutorial generell freier gestalten (also den Spieler mehr ausprobieren lassen)

## **9.5 Barrierefreiheit**

Ein wichtiger Punkt, der im Rahmen des Projekts nicht mehr zu implementieren war, war die Unterstützung eines Linkshändermodus. Aktuell sind Linkshänder gezwungen mit dem rechten Stick zu teleportieren, während der linke Stick keine Funktion hat. Es könnte hier entweder zusätzlich der linke Stick mit dem Teleport belegt oder eine Auswahl der dominanten Hand zu Beginn des Spiels möglich gemacht werden.

## **9.6 Items**

Eine Idee, die vor allem zu Beginn des Projekts intern diskutiert wurde, war die Einbindung von Items. Hier kann man sich verschiedene Itemsysteme überlegen. Natürlich sollte je nach Anforderung darauf geachtet werden, weiterhin die Einfachheit zu gewährleisten, falls dies weiterhin erwünscht sein sollte.

## **9.7 Kameralogik**

Im aktuellen Stand des Spiels ist die Kameralogik zufällig (siehe Spectator View). Es kann hier ein Kamerasytem eingebunden werden, das zu bestimmten Zeit-

punkte auf gewisse Kameras schaltet, um die Spectator View noch dynamischer und spannender in Szene zu setzen.

## **9.8 Ballindikator mit Ball verknüpfen**

Der Ballindikator (siehe Ballindikator) befindet sich als eigenständiges Objekt in der Szene. Dies stellt bei einem Ball der diesem zugewiesen wird kein Problem dar. Falls allerdings der Wunsch entsteht mehrere Bälle zur selben Zeit im Spiel zu haben, bietet es sich an den Indikator direkt an den Ball anzubinden, damit ein neugespawnter Ball sofort einen Indikator besitzt.

## 10 Bedienungsanleitung & Projektsetup

### 10.1 Systemvoraussetzungen

Um mit der Entwicklung am Projekt zu beginnen und Builds auf die Pico VR-Brille zu deployen, sind die folgenden Software- und Hardware-Komponenten erforderlich. Die Einhaltung der angegebenen Versionen ist wichtig für die Kompatibilität mit den verwendeten Plugins und Bibliotheken.

#### 10.1.1 Softwareanforderungen

##### Unity Editor:

- Empfohlene Version: Unity 6000.0.47f1
- Erforderliche Module: Stellen Sie sicher, dass die Module „Android Build Support“ (inkl. Android SDK & NDK Tools, OpenJDK) für das Deployment auf die Pico-Brillen installiert sind. Optional können auch Build Supports für Ihre Entwicklungsplattform (Windows, Mac, Linux) installiert werden.

##### Unity Packages

(werden hauptsächlich via manifest.json importiert, müssen aber ggf. im Package Manager verifiziert werden)

- Pico XR Plugin (com.unity.xr.picoxr, empfohlene Version: 3.1.0: Ermöglicht die Interaktion mit der Pico-Hardware.
- XR Interaction Toolkit (com.unity.xr.interaction.toolkit, empfohlene Version: 3.1.2): Stellt das Framework für VR-Interaktionen bereit (Greifen, Bewegen etc.).
- Netcode for GameObjects (NGO) (com.unity.netcode.gameobjects, empfohlene Version: 2.3.2): Die offizielle Unity-Lösung für Multiplayer-Networking.
- Unity Transport (com.unity.transport, wird als Abhängigkeit von NGO installiert): Das zugrundeliegende Netzwerkprotokoll (standardmäßig UTP auf Basis von UDP).
- Input System (com.unity.inputsystem, 1.14.0): Das Input-Management-System von Unity.

##### Weitere Tools:

- Git: Versionskontrollsystem zur Verwaltung des Projekts.

- Android SDK Platform Tools: Enthält das ADB (Android Debug Bridge) Kommandozeilen-Tool, das für das Deployment von APKs auf die Pico-Brille und das Anzeigen von Debug-Logs benötigt wird. Dies wird oft automatisch mit dem Android Build Support in Unity installiert, kann aber auch separat bezogen werden.
- Ein C-Sharp Code-Editor: Für die Bearbeitung der Skripte (z.B. Visual Studio, Visual Studio Code, JetBrains Rider).

### **10.1.2 Hardwareanforderungen**

- Entwicklungs-PC: Ein Computer mit einem unterstützten Betriebssystem (Windows 10/11, macOS, Linux) und ausreichender Leistung (CPU, GPU, RAM), um den Unity Editor flüssig auszuführen und VR-Builds zu erstellen.
- Pico VR-Brille: Mindestens eine, idealerweise zwei (oder mehr) Brillen der Pico 4 Ultra Enterprise.
- USB-C Datenkabel: Ein (idealerweise mehrere) USB-C Kabel, das Daten übertragen kann, wird benötigt, um die Pico-Brille mit dem PC zu verbinden und ADB zu nutzen.
- WLAN-Netzwerk: Ein lokales 5GHz WLAN-Netzwerk ist erforderlich. Das Netzwerk muss nicht zwingend mit dem Internet verbunden sein, die Brillen müssen sich lediglich im selben lokalen Netzwerk befinden. Dafür ist ein geeigneter Router erforderlich.

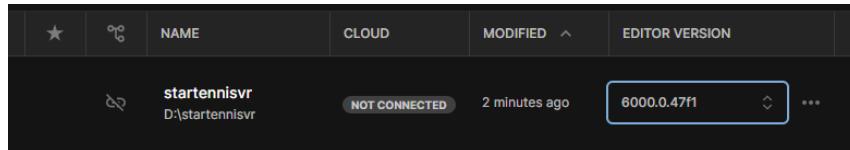
## **10.2 Schritt-für-Schritt Projekteinrichtung**

Dieses Unterkapitel führt Sie durch den Prozess des Imports des Projekts, der Installation notwendiger Unity Packages und der Konfiguration der XR-Einstellungen, um das Projekt für die Pico-Entwicklung vorzubereiten.

### **10.2.1 Unity Projekt importieren via Git**

- Ordner auswählen: Navigieren Sie in Ihrem Datei-Explorer zu dem Speicherort auf Ihrer Festplatte, an dem Sie das Projekt speichern möchten.
- Kommandozeile öffnen: Öffnen Sie eine Kommandozeile, Git Bash oder ein Terminal in diesem Ordner.
- Repository klonen: Führen Sie den folgenden Befehl aus, um das Projekt via SSH herunterzuladen: `git clone git@lv-gitlab.intern.th-ab.de:sep2025/vr-tennis/startennisvr.git`

- Für den Zugang zum Repo ist aktuell der EDU-VPN Client der TH Aschaffenburg notwendig und die Zugriffsrechte aufs Repository müssen noch freigeben werden. (Ansprechpartner Herr Prof.Biedermann)
- Projekt in Unity Hub öffnen: Starten Sie den Unity Hub. Klicken Sie auf den Button „Open“ (oder „Projekt von Festplatte hinzufügen“) und navigieren Sie zu dem Ordner, den Sie gerade geklont haben (dem Hauptordner des startennisvr-Projekts). Wählen Sie diesen Ordner aus.
- Unity Editor starten: Unity Hub sollte nun das Projekt in der korrekten Version (gemäß den Projekt-Einstellungen im Repository) öffnen. Beim ersten Öffnen kann es einige Zeit dauern, da Unity die Pakete auflöst, Skripte kompiliert und die Library neu aufbaut.

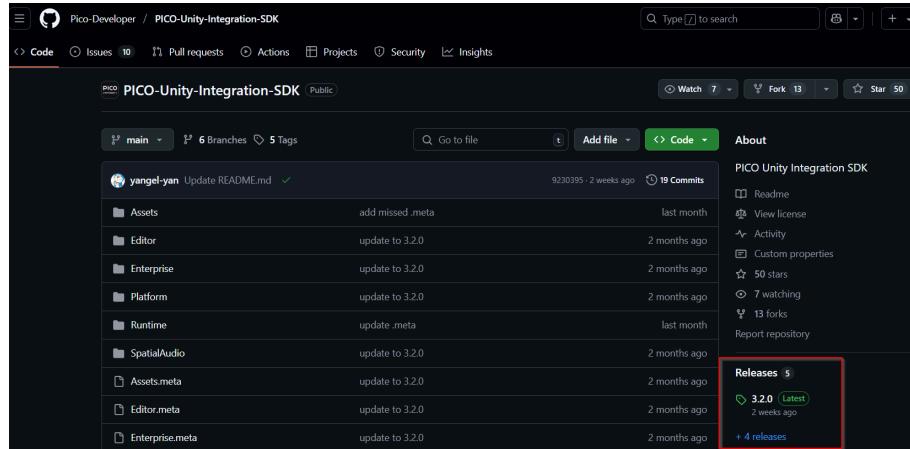


**Abb. 20:** Projekt im Unity Hub

### 10.2.2 Pico Unity Integration SDK

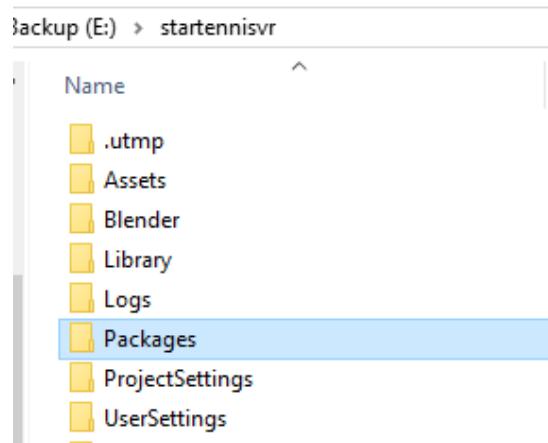
Das Pico Unity Integration SDK, muss manuell von folgender Seite heruntergeladen werden. Auf der rechten Seite kann die passende Version ausgewählt werden.

- <https://github.com/Pico-Developer/PICO-Unity-Integration-SDK>



**Abb. 21:** Unity Integration SDK

Auf der rechten Seite kann die passende Version ausgewählt werden. Anschließend muss es in den „Packages“ Ordner des importierten Git-Projektes kopiert werden.



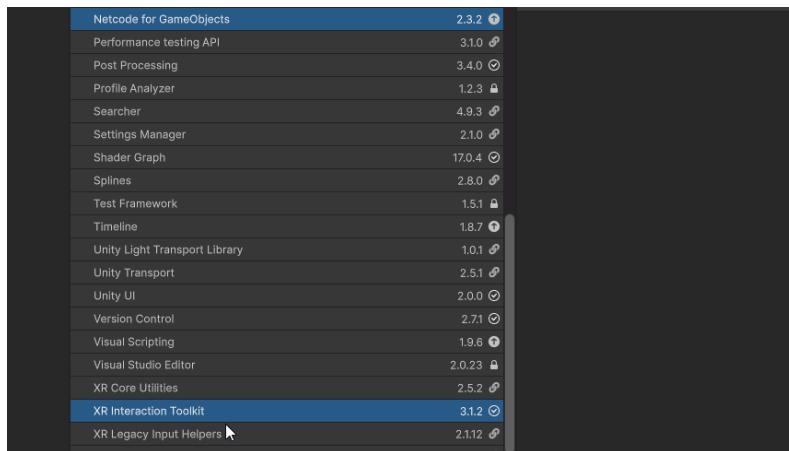
**Abb. 22:** Packages-Ordner

### 10.2.3 Notwendige Unity Packages installieren und verifizieren

Obwohl die meisten benötigten Packages über die manifest.json des Projekts automatisch importiert werden sollten, ist es ratsam, den Package Manager zu überprüfen und sicherzustellen, dass alle notwendigen Pakete in den korrekten Versionen installiert sind.

- Package Manager öffnen: Gehen Sie im Unity Editor Menü zu Window > Package Manager.
- Quelle auswählen: Stellen Sie sicher, dass oben links im Dropdown-Menü „Unity Registry“ ausgewählt ist. Dies zeigt alle von Unity bereitgestellten Pakete an.
- Pakete suchen und installieren/verifizieren: Suchen Sie nacheinander nach den in 5.1 gelisteten Paketen.
- Prüfen Sie auf der rechten Seite die installierte Version. Stellen Sie sicher, dass sie der in angegebenen empfohlenen Version entspricht oder kompatibel ist.
- Wenn ein Paket noch nicht installiert ist, klicken Sie auf den Button „Install“. Wenn eine neuere Version als empfohlen verfügbar ist, aber die Kompatibilität unklar ist, halten Sie sich am besten an die empfohlene Version.

(Hinweis: Unity Transport wird als Abhängigkeit von Netcode for GameObjects installiert und sollte automatisch erscheinen, wenn NGO installiert ist.)

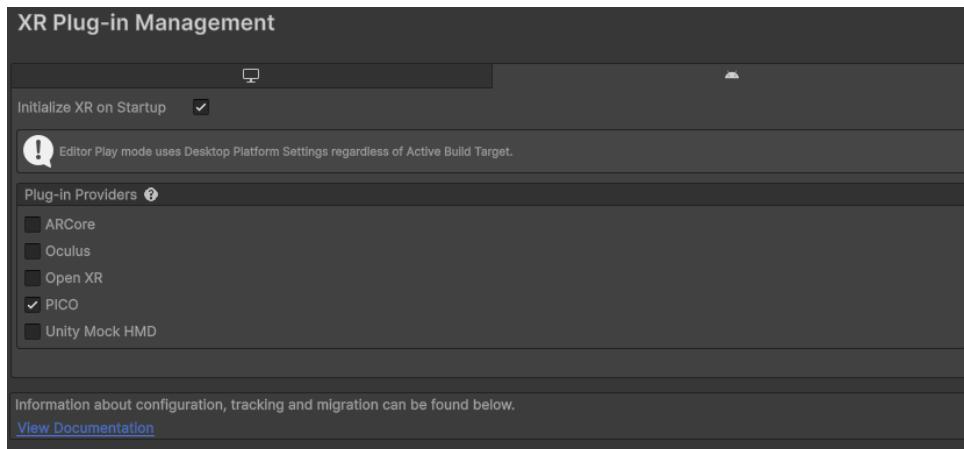


**Abb. 23:** Package Manager Versionskontrolle

#### 10.2.4 XR Plugin Management konfigurieren (Pico XR)

Nachdem das Pico XR Plugin installiert ist, müssen Sie Unity mitteilen, dass es dieses Plugin verwenden soll, insbesondere für Builds auf Android (Pico) und optional für das Testen im Editor auf dem PC.

- XR Plugin Management öffnen: Gehen Sie im Unity Editor Menü zu Edit > Project Settings. Wählen Sie links im Fenster XR Plugin Management.



**Abb. 24:** XR Plugin Management

- Android-Plattform konfigurieren: Wählen Sie oben im XR Plugin Management Fenster das Android-Tab (Symbol eines kleinen Androiden). Setzen Sie einen Haken bei „Pico XR“ in der Liste der Plugin-Provider. Unity wird nun den Pico XR Loader für Android aktivieren.
- PC-Plattform konfigurieren (Optional für Editor-Testing): Wählen Sie das PC, Mac & Linux Standalone-Tab (Symbol eines kleinen Computers). Setzen Sie ebenfalls einen Haken bei „Pico XR“. Dies ermöglicht das Testen im Unity Editor über Pico Link oder den Streaming Assistant.
- Pico XR Einstellungen (Android): Nachdem der Loader aktiviert ist, wählen Sie links im XR Plugin Management Fenster den Punkt „Pico XR“ unterhalb des Android-Tabs aus.



**Abb. 25:** PICO Einstellungen

## 10.3 InputMapping & XR-Toolkit

### 10.3.1 Input System & Action Asset

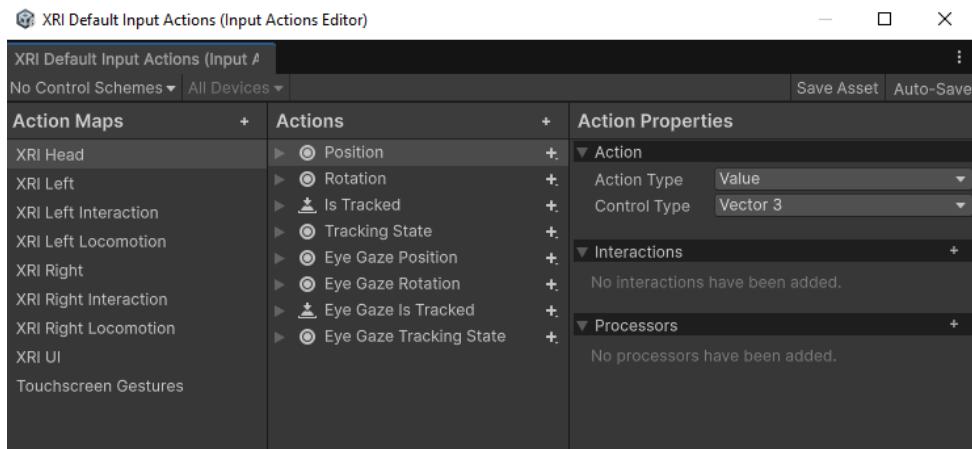
Das Projekt verwendet das moderne Unity Input System, um die Eingaben der VR-Controller zu managen. Logische „Aktionen“ werden hier definiert und mit den physikalischen Eingaben der Controller (Buttons, Achsen) verknüpft.

#### Input System Einstellungen verifizieren:

- Gehen Sie zu Edit > Project Settings. Wählen Sie links im Fenster Input System Package.
- Verifizieren Sie, dass unter „Active Input Handling“ die Option Both oder Input System Package (New) ausgewählt ist. Dies stellt sicher, dass das Input System aktiv ist.

#### Input Actions Asset finden und verstehen:

- Navigieren Sie im Projekt-Fenster zu Ihrem Input Actions Asset. Der Pfad ist Assets/Samples/XR Interaction/Toolkit/3.1.2/StarterAssets/Default Input Actions.inputactions.
- Doppelklicken Sie auf das Asset, um das Input Actions Editor-Fenster zu öffnen.
- Es gibt Action Maps (z.B. „XRI LeftHand“, „XRI RightHand“), Actions (z.B. „Teleport Mode Activate“, „Grab“, „Select“, „Move“) und Bindings, die die Actions mit spezifischen Controller-Eingaben verknüpfen (z.B. „Trigger-Button“, „Thumbstick“).
- Hier wird die Grundlage für alle Controller-Interaktionen gelegt.



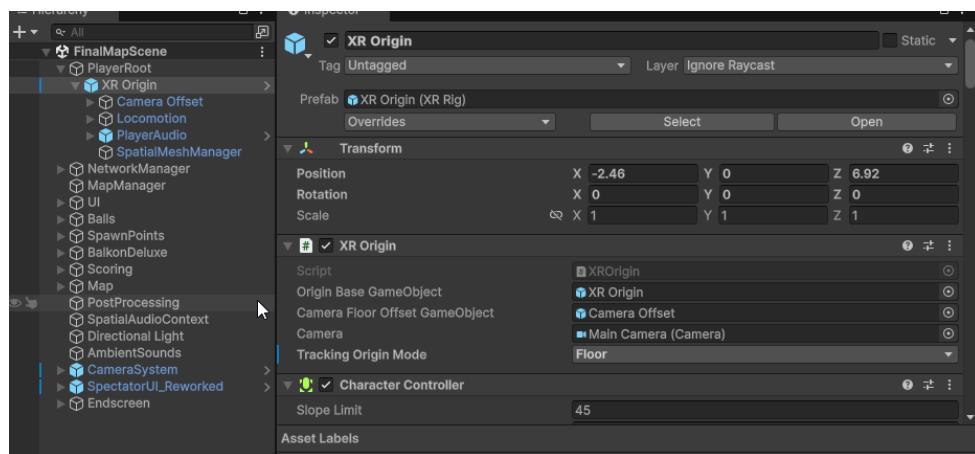
**Abb. 26:** XR Default Input Actions

### 10.3.2 XR Interaction Toolkit Kernkomponenten

Das XR Interaction Toolkit bietet die Komponenten, die es VR-Objekten ermöglichen, auf die Input Actions zu reagieren und Interaktionen auszuführen.

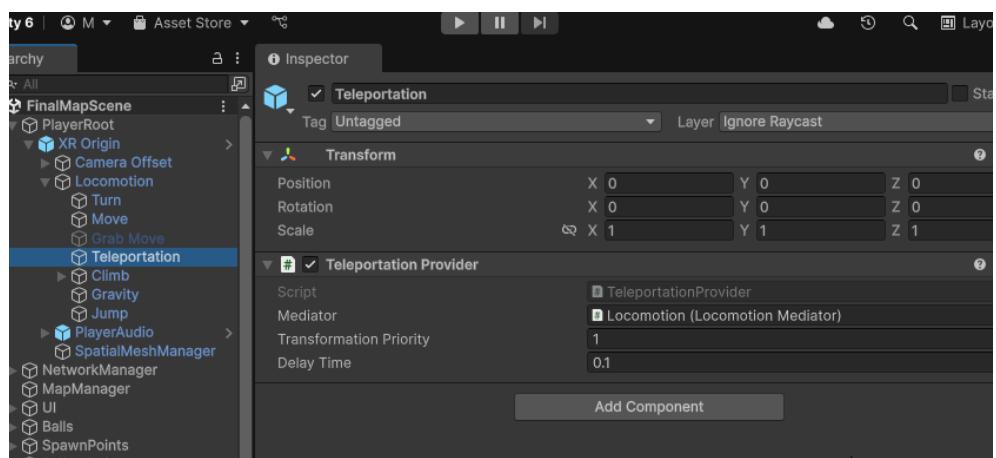
#### XR Origin (XR Rig) und Locomotion System:

- Öffnen Sie Ihre Hauptszene (FinalMapScene.unity).
- Wählen Sie das XR Origin (XR Rig) GameObject in der Hierarchie unter PlayerRoot aus.
- Dies ist das zentrale Objekt für den Spieler (Position im virtuellen Raum, enthält die Kamera und die Controller-GameObjects).



**Abb. 27:** XR Origin

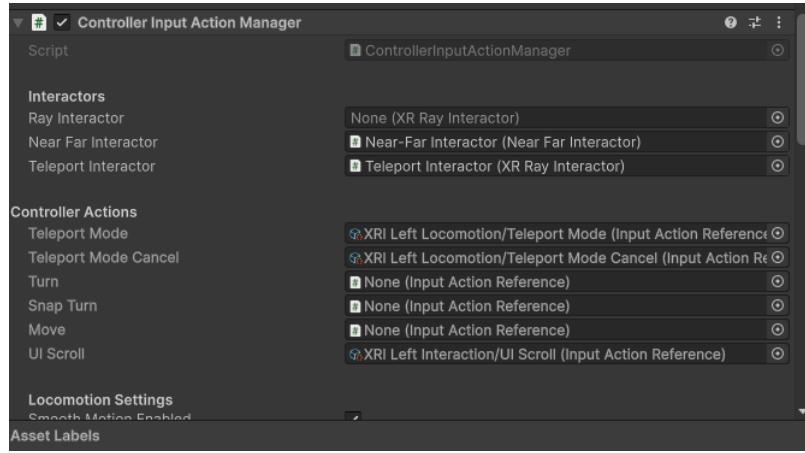
- Wählen Sie das Locomotion System GameObject aus .
- Dies enthält den TeleportationProvider, welcher für die Ausführung von Teleportationsanfragen zuständig ist.



**Abb. 28:** Teleportation Provider

#### **Controller Input Action Manager:**

- Wählen Sie das Left Controller und Right Controller GameObject in der Hierarchie aus.
- Der Controller Input Action Manager(Komponente auf jedem Controller) verbindet die logischen Input Actions mit den physischen Interactors auf dem Controller.



**Abb. 29:** Controller Input Action Manager

- Unter Controller Actions sind die spezifischen Input Actions (z.B. „Teleport Mode“, „Turn“, „Move“,) aus dem Input Actions Asset zugewiesen.
- Unter Interactors sind die physischen Interactor-Komponenten auf dem Controller zugewiesen. (z.B. XR Ray Interactor).

#### Standard-Interactor Komponenten:

- Die Standard-Interactor-Komponenten, können auf den Controllern für verschiedene Interaktionstypen verwendet werden (entweder direkt auf dem Controller oder auf einem Kind-GameObject wie „Teleport Interactor“):
- XR Direct Interactor: Für das direkte Greifen von Objekten, die sich physisch in Reichweite befinden.
- XR Ray Interactor: Für das Zielen auf Objekte über eine Distanz (für Auswahl, UI-Interaktion, Teleportation).
- Die Interaction Layer Mask auf diesen Interaktoren bestimmt, welche Layer von Interactables diese Interaktoren überhaupt „sehen“ können.

## 10.4 Szenen & Assets

Dieses Kapitel bietet einen Überblick über die Struktur der Unity-Szenen im Projekt und die wichtigsten Assets, die für das Spiel relevant sind.

#### 10.4.1 Szenenstruktur

- Hauptszene finden: Die Hauptszene des Spiels befindet sich unter Assets/Scenes/FinalMapScene.unity
- Es gibt weitere Szenen, die sich auch alle unter Assets/Scenes/ befinden. Einen genauen Überblick über die Szenenabfolge finden sie in Kapitel 2.3 Szenenablauf)
- Szenen in Build Settings: Verifizieren Sie, dass die benötigten Szenen in den Build Settings (File > Build Profiles unter „Scenes In Build“ und „Scene List“) enthalten und in der korrekten Reihenfolge sind.

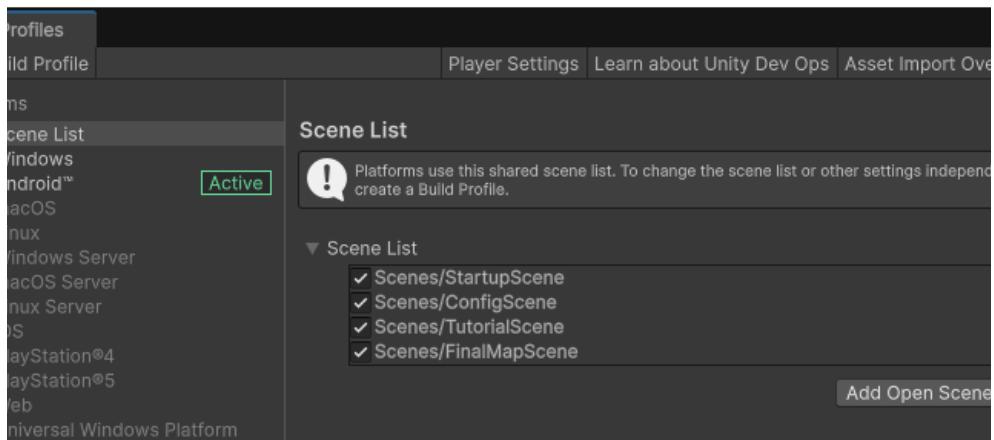
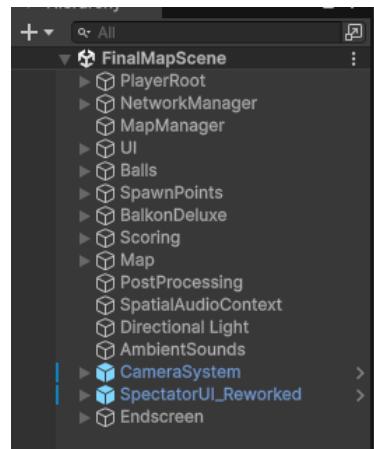


Abb. 30: Scene List

- Top-Level GameObjects: Beim Öffnen der Hauptszene in der Hierarchie-Ansicht sehen Sie die wichtigsten GameObjects auf oberster Ebene.
  - NetworkManager: Enthält die zentrale NetworkManager-Komponente und ist für das Netzwerk-Setup zuständig.
  - Scoring: Enthält das ScoreManager-Skript und die Scoring Sound Objekte, sowie das Scoreboard
  - PlayerRoot: Beinhaltet das XR Origin Rig und das Locomotion System.
  - Map: Enthält alles was mit der Map zu tun hat
  - SpawnPoints: Enthält die Startpunkte für Spieler und Ball
  - Balls: Enthält alles was mit dem Ball zu tun hat
  - UI: Enthält unter anderem den Start-Button

- MapManager: Enthält alles was mit der Steuerung der Map zu tun hat
- BalkonDeluxe: Enthält die Objekte zum Balkon auf dem Turm, auf dem man startet
- CameraSystem: Enthält alles was mit den Kameras des Spectatorview zu tun hat
- SpectatorUI\_Reworked: Enthält die Oberfläche zur Anzeige des Spielstandes in der Spectatorview
- Endscreen: Enthält den Gewinnerbildschirm
- DirectionalLight: Enthält alles was mit der Lichteinstrahlung und dem Schattenwurf zu tun hat
- AmbientSounds: Enthält den AmbientSound für das Hintergrundgeräusch
- SpatialAudioContext:
- PostProcessing: Enthält alles zum Postprocessing



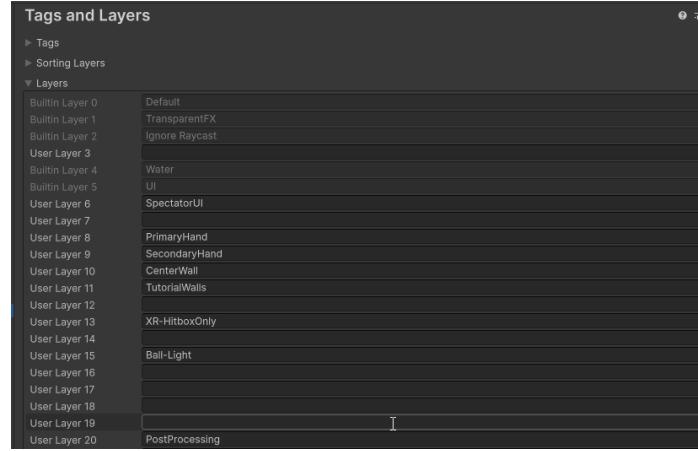
**Abb. 31:** Game Objects

#### 10.4.2 Physik & Layer Konfiguration

Eine korrekte Konfiguration der Physik-Layer ist erforderlich, damit Objekte wie Spieler und Ball korrekt mit der Umgebung interagieren.

##### Layer Überblick:

Gehen Sie zu Edit > Project Settings > Tags and Layers.



**Abb. 32:** Tags and Layers

### Physics Collision Matrix:

Gehen Sie zu Edit > Project Settings > Physics.

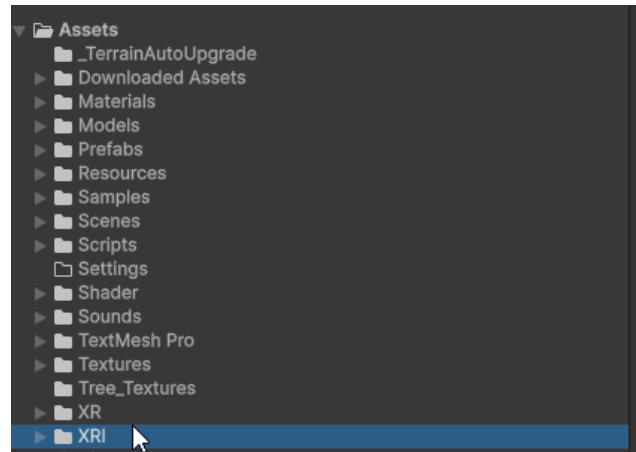
		Layer Collision Matrix																			
		Default																			
		Default	TransparentFX	Ignore Raycast	Water	UI	SpectatorUI	PrimaryHand	SecondaryHand	CenterWall	TutorialWalls	XR-HitboxOnly	Ball-Light	PostProcessing							
Default	✓	✓																			
TransparentFX	✓	✓	✓																		
Ignore Raycast	✓	✓	✓	✓																	
Water	✓	✓	✓	✓	✓																
UI	✓	✓	✓	✓	✓	✓															
SpectatorUI	✓	✓	✓	✓	✓	✓	✓														
PrimaryHand	✓	✓	✓	✓	✓	✓	✓	✓													
SecondaryHand	✓	✓	✓	✓	✓	✓	✓	✓	✓												
CenterWall	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓											
TutorialWalls	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓										
XR-HitboxOnly	✓	✓	✓																		
Ball-Light	✓	✓																			
PostProcessing	✓																				

**Abb. 33:** Collision Matrix

### 10.4.3 Asset-Organisation

- Scenes/: Enthält die Unity-Szenendateien.
- Scripts/: Enthält alle C-Sharp-Skripte, in weiteren Unterordnern (z.B. Scripts/Networking)
- Prefabs/: Enthält wiederverwendbare Prefabs (Spieler, Ball, UI-Elemente etc.).

- Materials/: Enthält Materialien.
- Models/: Enthält 3D-Modelle.
- Textures/: Enthält Textur-Bilder.
- UI/: Enthält UI-spezifische Assets und Prefabs.



**Abb. 34:** Assetstruktur

## 10.5 Build & Deployment

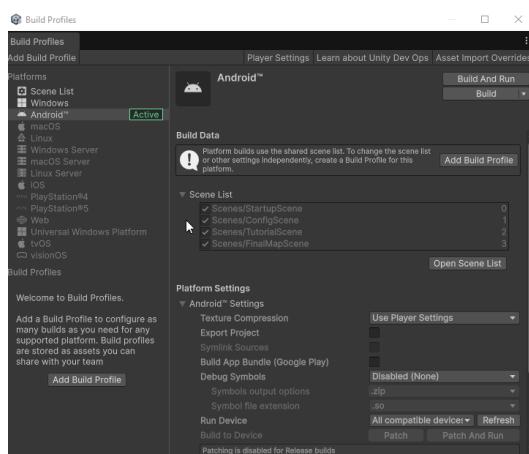
Nachdem das Projekt in Unity eingerichtet und entwickelt wurde, muss es in eine eigenständige ausführbare Anwendung kompiliert (build) und auf die Pico VR-Brille übertragen (deployed) werden.

### 10.5.1 Build Settings

Die Build Settings legen fest, welche Plattformen unterstützt werden, welche Szenen im Build enthalten sind und grundlegende Build-Optionen.

- Build Settings öffnen: Gehen Sie im Unity Editor Menü zu File > Build Settings.
- Szenen hinzufügen: Stellen Sie sicher, dass alle benötigten Szenen für das Spiel in der Liste „Scenes In Build“ enthalten sind. Sie können Szenen aus dem Projekt-Fenster in diese Liste ziehen. Die Reihenfolge der Szenen ist relevant.
- Plattform auswählen: Wählen Sie in der Liste der Plattformen auf der linken Seite „Android“ aus.

- Plattform wechseln: Wenn „Android“ noch nicht Ihre aktive Plattform ist (erkennbar am Unity-Symbol neben dem Namen), klicken Sie auf den Button „Switch Platform“. Dieser Vorgang kann einige Zeit in Anspruch nehmen. Im Falle eines SpectatorView Builds sollte hier allerdings eine PC Build-Plattform ausgewählt werden (Windows, MacOS, Linux).
- Gerät und Architektur konfigurieren: Unter „Run Device“: Wählen Sie „All Compatible Devices“ aus, damit Unity versucht, den Build automatisch auf einem per ADB verbundenen Gerät auszuführen. Oder wählen Sie direkt die PICO Brille aus.

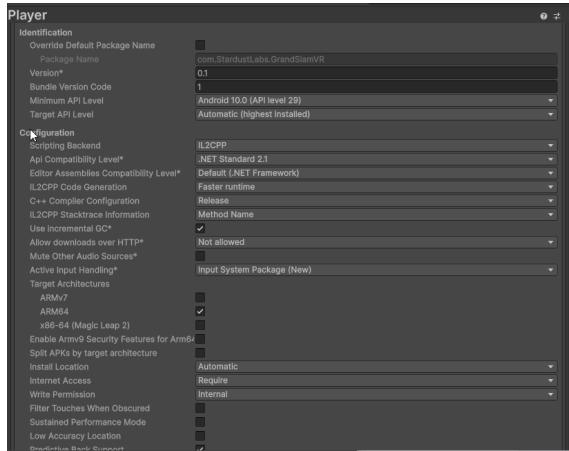


**Abb. 35:** Build Profiles

### 10.5.2 Player Settings

Die Player Settings enthalten wichtige Konfigurationen, die spezifisch für die Zielplattform sind, wie App-Identifikation, Grafikeinstellungen und Signing-Informationen.

- Player Settings öffnen: Gehen Sie im Unity Editor Menü zu Edit > Project Settings. Wählen Sie links im Fenster Player.
- Android-Einstellungen auswählen: Stellen Sie sicher, dass Sie die Einstellungen für die Android-Plattform bearbeiten, indem Sie oben im Fenster das Android-Icon (Symbol eines kleinen Androiden) auswählen.



**Abb. 36:** Player Settings

### Identification:

- Erweitern Sie den Abschnitt „Other Settings“.
- Unter „Identification“:
- Stellen Sie sicher, dass ein „Package Name“ eingetragen ist
- Minimum API Level: Stellen Sie diesen Wert auf die von Pico geforderte minimale Android API Version (29) ein.
- Target API Level: Automatic

### Configuration:

- Bleiben Sie im Abschnitt „Other Settings“.
- Unter „Configuration“:
- Scripting Backend: IL2CPP
- Target Architectures: ARM64. (Entfernen Sie ggf. den Haken bei ARMv7.)

### 10.5.3 Build & Deployment-Prozess

Sobald die Build Settings und Player Settings korrekt konfiguriert sind, können Sie den Build erstellen und auf die Brille bringen. ADB (Android Debug Bridge) wird für die Verbindung und Installation benötigt.

#### ADB vorbereiten und Brille verbinden:

- Stellen Sie sicher, dass die Android SDK Platform Tools installiert sind und ADB in Ihrer Kommandozeile verfügbar ist (siehe 5.1 Systemvoraussetzungen).

- Aktivieren Sie den Entwicklermodus und USB-Debugging auf Ihrer Pico VR-Brille (siehe 5.2.3 XR Plugin Management).
- Verbinden Sie Ihre Pico-Brille mit einem USB-C Datenkabel mit Ihrem Entwicklungs-PC.
- Auf der Brille müssen Sie die Nachfrage nach „USB-Debugging zulassen“ genehmigen und optional „Von diesem Computer immer zulassen“ auswählen.
- Öffnen Sie eine Kommandozeile und überprüfen Sie, ob die Brille erkannt wird: adb devices

Es sollte eine Geräte-ID mit dem Status „device“ angezeigt werden.

#### **APK erstellen und installieren:**

- Gehen Sie zurück zu den Build Settings (File > Build Settings).
- Option A (Build And Run - Empfohlen für schnelles Testen): Wenn die Brille per ADB verbunden und erkannt wird, klicken Sie auf „Build And Run“. Unity wird Sie auffordern, einen Ordner zum Speichern der APK zu wählen. Wählen Sie einen Speicherort (z.B. einen Builds-Ordner in Ihrem Projekt, der in .gitignore ausgeschlossen sein sollte). Unity baut die APK, signiert sie mit Ihrem Keystore und installiert sie automatisch über ADB auf der Brille. Die App sollte danach starten.
- Option B (Build - Für manuelle Installation oder Weitergabe): Wenn Sie nur die APK-Datei erstellen möchten, klicken Sie auf „Build“. Wählen Sie den Speicherort und Dateinamen für die APK. Unity baut und signiert die APK. Anschließend können Sie die generierte .apk-Datei manuell über ADB installieren:
  - adb install [Pfad zum Namen Ihrer .apk Datei]

Testen auf der Brille: Nach erfolgreicher Installation sollten Sie die App auf der Pico-Brille in Ihrer App-Bibliothek finden und starten können.

#### **10.5.4 PC Standalone Build für Spectator View**

Zusätzlich zum Android-Build für die Pico-Brillen wird ein separater Build für PC (Windows, Mac oder Linux) benötigt, um die Spectator View des Spiels auszuführen. Diese Anwendung wird auf einem PC gestartet und verbindet sich ebenfalls als Client mit dem Host (auf einer der Brillen).

#### **Plattform für PC-Build auswählen:**

Gehen Sie zu File > Build Settings.

- Wählen Sie in der Liste der Plattformen „PC, Mac & Linux Standalone“ aus.
- Klicken Sie auf „Switch Platform“. Dieser Vorgang kann erneut einige Zeit dauern.

#### **Szenen für PC-Build konfigurieren:**

Stellen Sie sicher, dass die korrekte(n) Szene(n) für die Spectator View in der Liste „Scenes In Build“ enthalten sind (StartupScene & FinalMapScene). Entfernen Sie ggf. Szenen, die nur für andere Build-Typen relevant sind.

#### **Settings (PC Standalone):**

Gehen Sie zu Edit > Project Settings > Player.

- Wählen Sie das PC, Mac & Linux Standalone-Tab (Computer-Icon).
- Identification: Prüfen Sie den Company Name und Product Name.
- Resolution and Presentation: Konfigurieren Sie die Fenster-Einstellungen (Fullscreen Mode, Default Screen Width/Height etc.) für die PC-Anwendung.
- Other Settings: Überprüfen Sie die Scripting Backend („IL2CPP“ für PC) und Architecture (x86-64 für 64-Bit-PCs).

#### **PC-Build erstellen:**

Gehen Sie zurück zu den Build Settings (File > Build Settings).

- Klicken Sie auf „Build“ (oder „Build And Run“, wenn Sie es direkt starten möchten).
- Wählen Sie einen Zielordner auf Ihrem PC und einen Namen für die ausführbare Datei (.exe unter Windows).
- Nachdem der Build abgeschlossen ist, finden Sie die ausführbare Datei im Zielordner.

Um die Spectator View zu starten, führen Sie einfach die generierte Datei aus. Diese Instanz wird sich dann als Client mit einem aktiven Host verbinden.

## **10.6 Weiterführende Ressourcen**

### **Unity Manual & Scripting API:**

Das offizielle Handbuch und die API-Referenz von Unity sind unerlässlich für alle allgemeinen Fragen zur Unity Engine, Skripting und eingebauten Komponenten.

- <https://docs.unity3d.com/Manual/> (Unity Manual)
- <https://docs.unity3d.com/ScriptReference/> (Scripting API)

### **XR Interaction Toolkit Dokumentation:**

Detaillierte Informationen zur Einrichtung und Nutzung des XR Interaction Toolkits, einschließlich der Interaktoren, Interactables, Locomotion und des Input System.

- <https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@latest>

### **Netcode for GameObjects (NGO)**

Alles zur Netzwerkprogrammierung mit NGO, einschließlich NetworkManager, NetworkObjects, NetworkVariables, RPCs, Netzwerk-Transporten und Discovery.

- <https://docs.unity.com/netcode/current/>

### **Unity Input System Dokumentation:**

Erklärung des modernen Input Systems, Action Assets, Bindings und der Verarbeitung von Eingaben.

<https://docs.unity3d.com/Packages/com.unity.inputsystem@latest>

**Pico XR SDK Dokumentation:** Spezifische Dokumentation für die Entwicklung auf Pico-Hardware mit Unity. Enthält Informationen zum Plugin, zu den Einstellungen und besonderen Pico-Features.

- <https://developer.picoxr.com/>

### **Git Dokumentation:**

Umfassende Ressourcen zur Nutzung von Git, Branches, Commits, Pullen, Pushen und dem Lösen von Konflikten.

- <https://git-scm.com/doc>

### **ADB (Android Debug Bridge) Dokumentation:**

Anleitung zur Nutzung des ADB-Tools für Geräteverwaltung, Installation von APKs und Anzeigen von Logs auf Android-Geräten.

- <https://developer.android.com/tools/adb>

## 11 Git Workflow & Kollaboration

### 11.1 Kernkonzepte

- Repository: Das zentrale Verzeichnis, das alle Projektdateien sowie die gesamte Historie der Änderungen speichert. Unser Repository wird auf GitLab gehostet.
- Commit: Eine „Momentaufnahme“ des Projekts zu einem bestimmten Zeitpunkt. Jeder Commit enthält eine Beschreibung der vorgenommenen Änderungen und einen eindeutigen Hash (ID). Commits sind die Bausteine der Projekthistorie.
- Branch: Ein unabhängiger Entwicklungszweig. Der Hauptentwicklungs- zweig ist typischerweise develop . Für neue Features, Bugfixes oder Aufgaben wird für gewöhnlich ein eigener „Feature Branch“ erstellt.
- Remote: Die Version des Repositories, die auf einem externen Server gespeichert ist (in unserem Fall auf GitLab, standardmäßig oft als origin bezeichnet).

### 11.2 Git-Workflow

Der tägliche Arbeitsablauf folgt einem Branching-Modell, bei dem jede Aufgabe in einem eigenen Zweig entwickelt wird.

#### Arbeit starten (Aktuellen Stand holen):

- Zu Beginn eines jeden Arbeitstages oder vor dem Start einer neuen Aufgabe ist es wichtig, den neuesten Stand des Hauptentwicklungs- zweigs (develop) zu holen.
- Wechseln Sie zum Hauptzweig und holen Sie die neuesten Änderungen vom Remote:
  - git switch develop
  - git pull origin develop
- Stellen Sie sicher, dass Sie keine ungespeicherten lokalen Änderungen haben, die dadurch überschrieben würden (git status prüfen, ggf. commit- ten oder stashen).

#### Feature Branch erstellen:

-Für jede neue Aufgabe (basierend auf einem Jira-Ticket) wird ein neuer Branch vom aktuellen Stand des Hauptzweigs (develop) erstellt.

- Der Branch-Name sollte das entsprechende Jira-Ticket referenzieren und kurz die Aufgabe beschreiben (z.B. VRTENNIS-56-Teleportation-beschränken).
- Erstellen Sie den neuen Branch und wechseln Sie direkt darauf:
  - git switch -c VRTENNIS-XX-Kurzbeschreibung

Oder :

- git checkout -b VRTENNIS-XX-Kurzbeschreibung

(Ersetzen Sie VRTENNIS-XX-Kurzbeschreibung durch den tatsächlichen Branch-Namen.)

#### **Entwickeln und Committen:**

- Arbeiten Sie an Ihrer Aufgabe in Unity und Ihrem Code-Editor.
- Committen Sie Ihre Änderungen regelmäßig! Immer wenn Sie einen sinnvollen Fortschritt erzielt haben, erstellen Sie einen Commit. Dies dient als Ihre lokale Sicherung und dokumentiert die Schritte.
- Prüfen Sie den Status: Sehen Sie, welche Dateien geändert oder neu hinzugefügt wurden:
  - git status
- Änderungen hinzufügen (Staging Area): Wählen Sie die Änderungen aus, die in den nächsten Commit aufgenommen werden sollen.
  - git add . Fügt alle Änderungen im aktuellen Verzeichnis hinzu

Oder spezifische Dateien:

- git add Assets/Scripts/DeinSkript.cs
- Commit erstellen: Erstellen Sie den Commit mit einer aussagekräftigen Commit-Nachricht. Beginnen Sie die Nachricht mit der Ticket-ID.
  - git commit -m „VRTENNIS-XX: Implementierung der Teleport-Mindestdistanz“

#### **Änderungen teilen (Pushen):**

- Wenn Sie Ihren Fortschritt sichern und für andere Teammitglieder sichtbar machen möchten, pushen Sie Ihre lokalen Commits auf den Remote-Branch auf GitLab.
  - git push origin Ihr Branchname

- Beim ersten Push eines neuen Branchs müssen Sie eventuell den Upstream-Branch setzen (-u):
  - ▶ `git push -u origin Ihr Branchname`

(Zukünftige Pushes auf diesen Branch können dann einfach `git push` sein.)

#### **Arbeit integrieren (Merge Request):**

- Nach Abschluss und Testen Ihrer Aufgabe, erstellen Sie auf der GitLab-Webseite einen Merge Request (MR) (manchmal auch Pull Request genannt) von Ihrem Feature Branch (VRTENNIS-XX-Kurzbeschreibung) in den Zielbranch (develop).
- Der MR dient als Diskussionsplattform für Code-Reviews durch andere Teammitglieder.
- Nach erfolgreichem Review und Behebung eventueller Kommentare und Fehler wird der Branch in den Zielbranch gemitigt.

### **11.3 Kollaboration und Konflikte**

- Regelmäßiges Pullen: Auch während der Arbeit an einem Feature Branch kann es sinnvoll sein, gelegentlich die neuesten Änderungen vom Hauptzweig (develop) in Ihren Feature Branch zu mergen (`git pull origin develop`), um Konflikte frühzeitig zu erkennen und zu lösen.
- Merge-Konflikte: Konflikte treten auf, wenn zwei Entwickler gleichzeitig Änderungen an derselben Stelle in einer Datei vornehmen. Git kann dies nicht automatisch zusammenführen.
- Wenn `git pull` oder das Mergen eines MRs Konflikte meldet, müssen diese manuell im Code-Editor gelöst werden. Git markiert die Konfliktstellen in den Dateien.
- Nach dem Lösen der Konflikte müssen die geänderten Dateien neu committed werden.