PropHunt Technical Write-up

Highrise Studio Take-Home Assessment

Game Design Document & Code Implementation

Core Game Loop & State Machine

PropHunt implements a server-authoritative finite state machine: `LOBBY → HIDING → HUNTING → ROUND_END → LOBBY`

- **LOBBY**: Dynamic ready system with quick-start (all ready = 5s countdown) or standard countdown (minimum ready)
- **HIDING**: Props teleport to arena, select disguises (35s default, auto-advance when all hidden)
- **HUNTING**: Hunters teleport with 5s delay, tag props via TapHandler interaction (240s, exponential miss penalties)
- **ROUND_END**: EndRoundScore UI with leaderboard, winner overlay, role preservation (15s)

Multiplayer Architecture

Built on Highrise Studio SDK using Unity 6000.0.58.f2 with URP 14.0.9:

- **Network Sync**: Global Event Pattern for Module↔UI communication, NetworkValues for state/timer/counts
- **Server Validation**: TapHandler-based interaction, cooldown (≥0.5s), phase/role validation, possession verification
- **Edge Cases**: Mid-game join → auto-spectator, duplicate tag prevention, eliminated props show original role in scores

Scoring System

Points-based system with tie-breaker logic. Props: +10/5s + 100 survival bonus. Hunters: +120/tag, exponential miss penalty (-10, -20, -40, -80...), accuracy bonus at round end. Zone-based multipliers (NearSpawn 1.5x, Mid 1.0x, Far 0.6x) implemented but currently disabled.

Network Communication Patterns

- **Global Events**: `_G.PH_EndRoundScoresEvent`, `_G.PH_StateChangedEvent`, `_G.PH_PropsCountEvent` for cross-script-type communication

- **NetworkValue Listeners**: Role changes, score updates, game state sync via Changed:Connect callbacks
- **RemoteFunction**: Client→Server validation (tag requests, possession requests, ready toggle)
- **FireAllClients**: Broadcast pattern for real-time UI updates (props count, phase transitions, scores)

Code Architecture Highlights

- Global Event Pattern solves Module/UI Event isolation (critical discovery during EndRoundScore implementation)
- Server-authoritative validation with client-side prediction for responsive gameplay
- One-Prop Rule with static props, possession system with server-side validation
- Unified logging system (PropHuntLogger) with per-system toggles via Unity Inspector
- TapHandler-based interaction system for prop selection and tagging
- Original role tracking prevents eliminated props from displaying as "Spectator" in final scores

---

Technical Art & Implementation

VFX System Architecture

Implemented `PropHuntVFXManager.lua` with DevBasics Tweens library for procedural animations:

- **Phase Transitions**: Camera movements between lobby/arena, UI fade overlays for state changes, synchronized with game state machine
- **Possession System**: Prop appearance with emissive highlight, prop scaling to match original size
- **Tag Effects**: Hit VFX with particle burst and prop scaling animation, miss VFX with visual feedback for invalid targets.
- **Round End VFX**: Smooth transitions to end-game UI, winner overlay animations, leaderboard fade-in effects

Phase Transition Implementation
VFX Manager coordinates state-synchronized effects:

- LOBBY→HIDING: Fade overlay before teleport, camera transition to arena, UI element fade-out sequence
- HIDING→HUNTING: 5-second countdown with visual timer, hunter teleport delay with fade effect
- HUNTING→ROUND_END: Victory/defeat screen transitions, leaderboard display
- ROUND_END→LOBBY: Arena cleanup, teleport back to lobby, UI element restore with fade-in

Possession VFX Pipeline
Multi-stage effect sequence using DevBasics Tweens:

1. Position transfer to prop transform
2. Prop scale-up animation matching player size
3. Visual confirmation feedback to client

Tag VFX System

- **Successful Tag**: Particle system spawn at contact point, Prop Scale pulse, hunter score popup indicator.
- **Miss Tag**: Rejection particle effect, Prop Scale Pulse

Custom Shader PipelineBasic Prop Shader: Custom Shader Graph material using ORME (Occlusion, Roughness, Metallic, Emissive) single texture workflow. Features world-aligned UV projection with triplanar blend for seamless texturing across arbitrary prop geometry. Exposes albedo tint/strength controls plus per-material adjustments for roughness, metallic, and emissive properties via material inspector.

GodrayUnlit Shader: Multi-beam volumetric lighting with procedural generation, UV offset controls, dual fade system for atmospheric depth. Configurable beam count (1-10), spacing, width parameters via material inspector. Additive blending pipeline optimized for mobile URP rendering.

Technical Art Features:

- Emissive material states for possessed props (toggled via VFX Manager)
- Screen-space UI transitions with DevBasics Tweens easing functions

Unity Asset Pipeline
Structured hierarchy in `Assets/PropHunt/`:

- **Prefab System**: 30+ prop prefabs with standardized collision meshes, "Possessable" tag for runtime detection, TapHandler components for interaction, scaling metadata for possession system
- **Zone Volumes**: BoxCollider trigger volumes with ZoneVolume.lua scripts, zone weight configuration (NearSpawn 1.5x, Mid 1.0x, Far 0.6x), visual debug overlays in Scene view
- **Spawn System**: LobbySpawn/ArenaSpawn empty GameObjects, position-based teleportation (50-100 units separation), camera transition anchor points for VFX
- **VFX Assets**: Particle prefabs for tag/possession effects, fade overlay UI elements, emissive material variants for props

Performance Optimizations

- Single texture atlas for all props with optimized UVs for quality
- Lightmap resolution set to 1024×1024 for optimal lighting quality

Future Enhancements (1 Week Extension):

- **Sound FX**: Audio feedback for possession, tagging, phase transitions, and UI interactions
- **Dynamic Spawn System**: Random spawn location generation for teleport destinations using NavMesh surface sampling. Instead of fixed spawn points, the system would use a central anchor point with configurable radius distance, polling valid NavMesh positions within the radius to generate randomized spawn locations each round, ensuring players spawn on walkable surfaces.
- **Performance Optimization**: Further optimization based on platform-specific technical constraint