Player Info HUD System - Master Implementation Plan

1. Project Overview

Objective: Extend the existing HUD system with race progress tracking, skill feedback, Fan Service mission integration, and opponent information display.

Timeline: 1-2 weeks (5-10 working days)
Team Size: 2 developers (1 C++ engineer, 1 UI developer)
Target Milestone: MVP ready for playtesting

---

2. Task Breakdown

Phase 1: Foundation & Core Extensions

TASK-001: Extend UPrototypeRacingUI Base Class

Priority: P0 (Critical)
Effort: 2 days
Owner: C++ Engineer
Dependencies: None

Subtasks:

- [ ] Add new Blueprint Implementable Events:
  - `OnRaceProgressUpdate`
  - `OnPlayerPositionUpdate`
  - `OnRaceTimerUpdate`
  - `OnLapChanged`
  - `OnFanServiceProgressChanged`
  - `OnSkillActivated`
  - ...
- [ ] Add event binding methods:
  - `BindToRaceEvents(ARaceTrackManager* RaceManager)`
  - `BindToVehicleEvents(ASimulatePhysicsCar* Vehicle)`
  - `BindToFanServiceEvents(UFanServiceSubsystem* FanService)`
  - ...
- [ ] Implement event handler methods:
  - `HandleRankingUpdate(const TArray<FPlayerRaceState>& PlayerRaceStates)`
  - `HandleLapCompleted(const FString& VehicleId, const int& LapNumber)`
  - `HandleTimeUpdate(const int& TimeRemaining)`
  - `HandleDriftPointChanged(float DriftPoint)`
  - `HandleFanServiceProgressUpdate(const FFanServiceProgressData& ProgressData)`
  - ...
- [ ] Update `NativeConstruct()` to auto-bind events on widget creation
- [ ] Add null checks and error handling for all event bindings

Acceptance Criteria:

- All new events compile without errors
- Events fire correctly when subscribed in Blueprint
- No crashes when RaceTrackManager/Vehicle/FanService are null

Testing:

- Unit test: Verify event binding/unbinding
- Integration test: Trigger events from RaceTrackManager and verify HUD receives them

---

TASK-002: Create Data Structures

Priority: P0 (Critical)
Effort: 1 day
Owner: C++ Engineer
Dependencies: None

Subtasks:

- ☐ Create `FRaceProgressInfo` struct
- ☐ Create `FSkillPopupData` struct
- ☐ Create `ESkillType` enum (PerfectDrift, AirborneBoost, NearMiss, CheckpointBonus, SpeedDemon)
- ☐ Create `FSkillReward` struct with skill type mapping
- ☐ Add BlueprintType specifiers to all structs/enums
- ☐ Document all struct fields with UPROPERTY comments

Acceptance Criteria:

- All structs/enums visible in Blueprint editor
- Structs can be created and modified in Blueprint
- No compilation warnings

Testing:

- Create test Blueprint that instantiates each struct
- Verify all fields are editable in Blueprint

TASK-003: Implement Race Progress Calculation

Priority: P0 (Critical)
Effort: 2 days
Owner: C++ Engineer
Dependencies: TASK-001

Subtasks:

- ☐ Add `CalculateRaceProgress()` method to `ARaceTrackManager`:

```
float CalculateRaceProgress(const FString& VehicleId);
```

- [ ] Implement progress calculation logic:
  - Get `ASimulatePhysicsCar::TotalDistance` for player vehicle
  - Get `ARoadGuide::SplineComponent->GetSplineLength()` for track
  - Calculate: `Progress% = (TotalDistance / SplineLength) * 100`
- [ ] Add progress update event to `OnCheckpointPassed` broadcast
- [ ] Handle edge cases:
  - Spline not found → return 0%
  - Vehicle not found → return 0%
  - Division by zero (SplineLength = 0) → return 0%
- [ ] Add logging for debugging progress calculation

Acceptance Criteria:

- Progress percentage updates every checkpoint pass
- Progress resets to 0% on race restart
- Progress reaches 100% at finish line
- No crashes when spline is missing

Testing:

- Drive full lap and verify progress goes 0% → 100%
- Test with multiple vehicles (ensure correct vehicle tracked)
- Test with missing spline (should not crash)

TASK-004: Implement Time Attack Countdown Timer

Priority: P0 (Critical)
Effort: 1.5 days
Owner: C++ Engineer
Dependencies: TASK-001

Subtasks:

- [ ] Verify existing `ARaceTrackManager::OnAttackTimeCountdown()` implementation
- [ ] Add checkpoint bonus logic (+20s on checkpoint pass)
- [ ] Broadcast `OnTimeAttackUpdate` event every second
- [ ] Add timer expiration check (when `AttackTimeCountdown <= 0`)
- [ ] Trigger race end when timer reaches 0
- [ ] Add timer format conversion utility:

```
FString FormatTimeMMSSMS(float TimeInSeconds);
```

Acceptance Criteria:

- Timer counts down from initial value (e.g., 60s)
- Timer increases by +20s when passing checkpoints
- Timer stops at 0:00.00 and triggers race end
- Timer format displays correctly (MM:SS.MS)

Testing:

- Start Time Attack race, verify countdown starts
- Pass checkpoint, verify +20s bonus applied
- Let timer reach 0, verify race ends
- Test timer format with various values (0.5s, 59.9s, 120.5s)

Phase 2: UI Components

TASK-005: UI Components Implementation (All Widgets)

Priority: P0 (Critical)
Effort: 5 days
Owner: UI Developer
Dependencies: TASK-001, TASK-002, TASK-003, TASK-004

Scope: UI Developer will break down and implement all HUD widgets including:

- Race Progress Info Widget (mode-specific display)
- Skill Popup Widget with object pooling
- Fan Service Widget (Progress Bar & CheckBox variants)
- Drift Points integration with Skill Popup
- All animations and visual polish

High-Level Requirements:

- ☐ Create `WBP_RaceProgressInfo` with mode-specific display (Sprint %, Time Attack countdown, Circuit laps)
- ☐ Create `WBP_SkillPopup` with object pool (max 5 concurrent, 3s accumulation window)
- ☐ Create `WBP_FanServiceWidget` with Progress Bar and CheckBox variants
- ☐ Integrate drift point tracking with skill popup system
- ☐ Implement all animations (appear, accumulate, disappear, milestone, warning)
- ☐ Position all widgets according to Figma design
- ☐ Ensure mobile readability (720p on 5-inch screen)

Acceptance Criteria:

- All widgets display correctly in all race modes
- Skill popup accumulates points within 3s window
- Fan Service widget switches between Progress Bar/CheckBox based on mission type
- All animations play smoothly at 60 FPS
- Widgets are readable on mobile devices

Testing:

- Test all widgets in Sprint, Time Attack, and Circuit modes
- Test skill popup with rapid activations (verify accumulation)
- Test Fan Service widget with all 6 mission types
- Verify mobile readability on target devices

Note: UI Developer will create detailed subtask breakdown during implementation

---

Phase 3: Advanced Features

TASK-006: Implement Opponent Info Widget (World-Space)

Priority: P2 (Medium)
Effort: 3 days
Owner: C++ Engineer + UI Designer
Dependencies: TASK-001

Subtasks (C++ - Day 1-2):

- [ ] Add `UWidgetComponent* OpponentInfoWidget` to `ASimulatePhysicsCar.h`
- [ ] Create opponent info widget in `BeginPlay()` for AI vehicles only
- [ ] Attach widget to vehicle root component with offset (0, 0, 150cm above vehicle)
- [ ] Set widget space to `EWidgetSpace::Screen` (always faces camera)
- [ ] Create `UOpponentInfoManager` class to manage visibility updates:
  - `UpdateOpponentVisibility(ASimulatePhysicsCar* PlayerCar)`
  - `GetNearestOpponents(int32 MaxCount) → TArray<ASimulatePhysicsCar*>`
- [ ] Implement distance-based visibility logic:

```
float Distance = FVector::Distance(PlayerLocation, OpponentLocation);
if (Distance >= 2000.0f) { // 20m
    Widget->SetVisibility(ESlateVisibility::Hidden);
} else if (Distance >= 1000.0f) { // 10m
    Widget->ShowPosition(Opponent->Ranking);
} else if (Distance >= 500.0f) { // 5m
    Widget->TransitionToName(Opponent->PlayerName);
} else {
    Widget->ShowName(Opponent->PlayerName);
}
```

- [ ] Add timer-based update (5 Hz) instead of tick
- [ ] Implement frustum culling (hide widgets outside camera view)

Subtasks (Blueprint - Day 3):

- [ ] Create `WBP_OpponentInfo` Blueprint widget
- [ ] Design widget hierarchy (InfoContainer, PositionText, NameText)
- [ ] Create animations:
  - FadeInAnim: Fade in when entering 20m range
  - TransitionAnim: Crossfade between position and name

    - FadeOutAnim: Fade out when exiting 20m range
- [ ] Implement `ShowPosition()`, `ShowName()`, `TransitionToName()` methods
- [ ] Add text outline/shadow for readability against backgrounds

Acceptance Criteria:

- Widget appears when opponent enters 20m range
- Widget shows position when 10-20m away
- Widget transitions to name when 5-10m away
- Widget shows name only when <5m away
- Widget disappears when opponent exits 20m range
- Max 8 opponent widgets rendered simultaneously
- Update frequency: 5 Hz (every 0.2s)

Testing:

- Drive near AI opponent, verify widget appears at 20m
- Approach opponent, verify transition from position to name
- Drive away, verify widget disappears at 20m
- Test with 10 opponents, verify only nearest 8 shown

TASK-007: Add Player Position Conditional Display

Priority: P0 (Critical)
Effort: 0.5 days
Owner: C++ Engineer
Dependencies: TASK-001

Subtasks:

- [ ] Add position display logic to `HandleRankingUpdate()`:

```
void UPrototypeRacingUI::HandleRankingUpdate(const TArray<FPlayerRaceState>&
PlayerRaceStates) {
    int32 TotalRacers = PlayerRaceStates.Num();
    bool bShowPosition = TotalRacers > 1; // Hide in Time Attack (single
player)

    if (bShowPosition) {
        FPlayerRaceState* PlayerState =
GetPlayerRaceState(PlayerRaceStates);
        OnPlayerPositionUpdate(PlayerState->Ranking, TotalRacers);
    }
}
```

- [ ] Update `WBP_PrototypeRacingUI` Blueprint to hide position widget when `TotalRacers == 1`

Acceptance Criteria:

- Position widget hidden in Time Attack mode
- Position widget visible in Sprint/Circuit modes with AI opponents
- No crashes when `PlayerRaceStates` is empty

Testing:

- Start Time Attack race, verify position widget hidden
- Start Circuit race with 5 AI opponents, verify position widget shows "1st/6"

Phase 4: Polish & Optimization

TASK-008: Performance Optimization & Profiling

Priority: P1 (High)
Effort: 2 days
Owner: C++ Engineer
Dependencies: All previous tasks

Subtasks:

- [ ] Profile HUD performance on target devices (Android, iPhone)
- [ ] Measure UI frame time using `stat unit` and `stat scenerendering`
- [ ] Optimize draw calls:
  - Combine UI textures into single atlas
  - Reduce material count to max 3
  - Collapse widget hierarchy (max 3 levels deep)
- [ ] Optimize update frequencies:
  - Speed/NOS: 60 Hz
  - Race progress: 30 Hz
  - Position/lap: 10 Hz
  - Opponent info: 5 Hz
- [ ] Add widget pooling for all dynamic widgets
- [ ] Implement frustum culling for world-space widgets
- [ ] Add distance-based LOD for opponent info (hide details when far)
- [ ] Profile memory usage and ensure <50MB for all HUD components

Acceptance Criteria:

- UI frame time <5ms on high-end Android devices
- Total draw calls <50 per frame
- Memory usage <50MB
- 60 FPS maintained during normal gameplay
- No frame drops when all HUD components active

Testing:

- Run profiling session on target devices
- Capture frame time data for 5-minute race
- Verify no frame drops below 60 FPS (high-end) or 30 FPS (mid-range)

TASK-009: Integration Testing & Bug Fixes

Priority: P0 (Critical)
Effort: 2 days
Owner: Both Engineers
Dependencies: All previous tasks

Subtasks:

- ☐ Test all race modes (Sprint, Time Attack, Circuit)
- ☐ Test all Fan Service mission types (6 types)
- ☐ Test all skill types (5 types)
- ☐ Test edge cases:
  - Race restart mid-race
  - Player finishes before AI opponents
  - Timer reaches 0 in Time Attack
  - Multiple skills activated simultaneously
  - Fan Service mission completed/failed
- ☐ Fix all critical bugs (crashes, incorrect data display)
- ☐ Fix all high-priority bugs (visual glitches, performance issues)
- ☐ Document known issues (low-priority bugs deferred to next sprint)

Acceptance Criteria:

- Zero critical bugs remaining
- Zero high-priority bugs remaining
- All edge cases handled gracefully (no crashes)
- Test coverage >80% for all HUD components

Testing:

- Run full regression test suite
- Perform manual playtesting session (30 minutes per race mode)
- Verify all requirements from `player-info-hud-overview.md` met

---

3. Dependencies & Sequencing

Critical Path

```
TASK-001 (Extend Base Class)
  ↓
TASK-003 (Race Progress Calculation) → TASK-005 (UI Components)
  ↓
TASK-004 (Countdown Timer) → TASK-005 (UI Components)
  ↓
TASK-007 (Position Conditional Display)
  ↓
TASK-009 (Integration Testing)
```

Parallel Tracks

- **Track A (Foundation)**: TASK-001 → TASK-002 → TASK-003 → TASK-004 (Week 1)
- **Track B (UI Components)**: TASK-005 (Week 1-2, starts after TASK-001 complete)
- **Track C (Advanced Features)**: TASK-006 → TASK-007 (Week 2)
- **Track D (Optimization)**: TASK-008 → TASK-009 (Week 2)

Recommendation:

- **Week 1**: C++ Engineer completes TASK-001 to TASK-004, UI Developer starts TASK-005 after TASK-001
- **Week 2**: C++ Engineer handles TASK-006 to TASK-009, UI Developer completes TASK-005

Document Status: Draft - Ready for implementation
Next Steps: Begin TASK-001 (Extend UPrototypeRacingUI Class)