Game Design Document (GDD)

Title: No Need For Speed

Overview of the Game Design: "No Need For Speed" is a networked multiplayer racing game developed using Unity's Netcode for GameObjects (NGO). The game supports multiple players and non-player characters (NPC). The game incorporates dynamic environments, customizable vehicles, and a finish-line mechanic that tracks progress across multiple tours. The emphasis is on fun, accessibility, and a balanced multiplayer experience rather than realistic physics or high-speed precision.

Logo:

NO NEED FOR SPEED

Key Features:

- Multiplayer racing with synchronization across clients.
- Dynamic NPCs with autonomous movement.
- Various player customization options, including vehicle models and emotes.
- A user-friendly interface with a start panel, in-game HUD, and ESC menu.
- Smooth transitions between game states, such as start, join, and finish.

Controls:

Implementation Details of Network Features:

1. **Player Management:**

- Players spawn at designated points using the PlayerSpawner script.
- Player positions and movements synchronize via NetworkObject and ClientNetworkTransform.

2. **NPCs and Movement:**
   - NPCs are managed by NPCManager and follow predefined waypoints with the NPCMovement script.
   - Server-side authority ensures consistent NPC behavior.
3. **Finish Line Tracking:**
   - The FinishLine script tracks progress and communicates with GameCoreManager for race state updates.
4. **User Interface Synchronization:**
   - The GameSceneManager updates ready indicators and countdowns using ClientRpc to synchronize UI.
5. **Game State Transitions:**
   - GameSceneManager handles player connections, disconnections, and scene changes seamlessly.

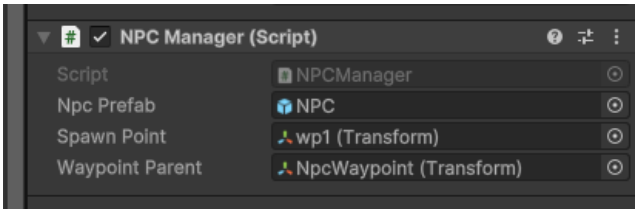
Script Breakdown:

AutoAddPlayerToVcamTargets.cs:

- Automatically assigns the player to the Virtual Camera's tracking target when they join the game.
- Detects players tagged as "Player" or "Player2" and updates the camera dynamically.
- Uses delayed execution to ensure players have spawned before assigning the camera.

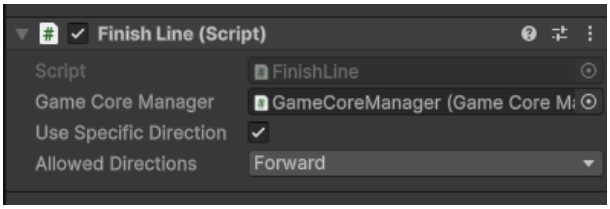
NPCManager.cs:

- Manages NPC spawning and movement toggling.
- Ensures server-side control of NPC instances using NetworkObject components.
- Supports resetting or toggling NPC movement states.



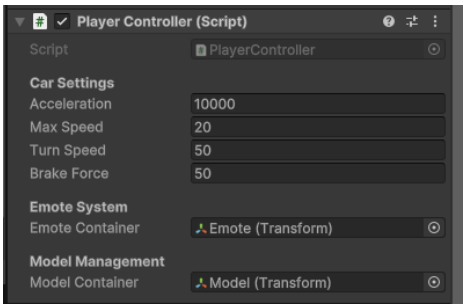
FinishLine.cs:

- Tracks when players or NPCs cross the finish line.
- Validates crossing directions and records progress in a Dictionary.
- Communicates crossing events to GameCoreManager.



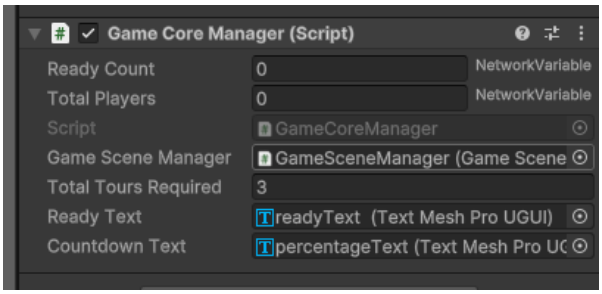
PlayerController.cs:

- Handles player movement, braking, and turning using Rigidbody physics.
- Manages emote and model changes, synchronized via server and client RPCs.



GameCoreManager.cs:

- Centralizes game logic, including player readiness, countdowns, and progress tracking.
- Uses NetworkVariable for shared game states and ClientRpc for UI updates.
- Declares winners and manages end-game states.



ClientNetworkTransform.cs:

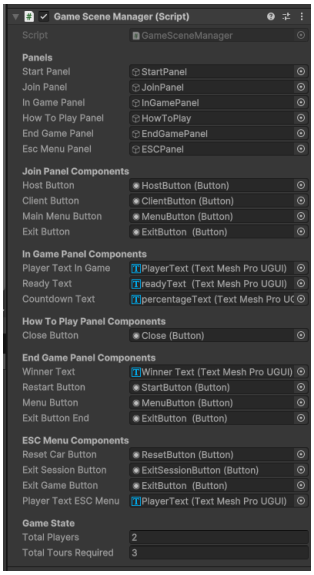
- Implements client-authoritative synchronization for smoother player control.
- Overrides default server authority to reduce latency.

Pit.cs:

- Detects when players enter a pit area and triggers vehicle model changes.
- Relies on player tags for identification and uses PlayerController for updates.

GameSceneManager.cs:

- Manages game scenes, UI panels, and state transitions.
- Handles player connections, disconnections, and game state transitions.
- Integrates Unity's SceneManager for scene control.

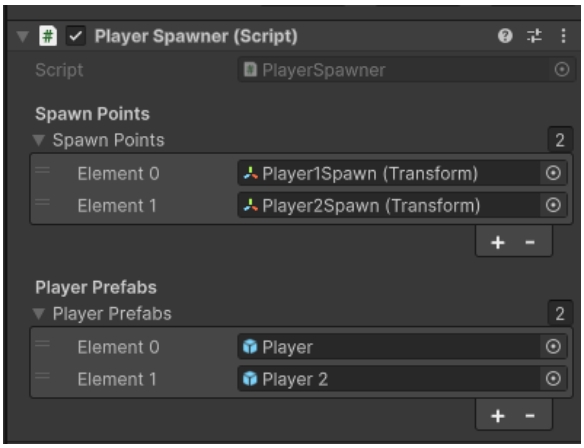


NPCMovement.cs:

- Governs NPC movement along pre-defined waypoints.
- Synchronizes movement states using server and client RPCs.
- Smoothly rotates NPCs toward their next target.

PlayerSpawner.cs:

- Spawns players at designated points based on connected client IDs.
- Supports multiple player prefabs for variety.
- Ensures synchronization using NetworkObject.



Challenges Faced and Solutions Implemented:

1. **Smooth Player Synchronization:**
   - Used ClientNetworkTransform for client-authoritative movement to reduce latency.
2. **Dynamic NPC Behavior:**
   - Managed NPC states server-side and synchronized with ServerRpc and ClientRpc.
3. **Handling Disconnections:**
   - Added callbacks in GameSceneManager to detect and manage disconnections.
4. **UI Synchronization:**
   - Utilized NetworkVariable and ClientRpc for real-time updates.

Reflection on the Learning Experience: This project deepened understanding of multiplayer game development, emphasizing:

- The importance of server authority for fairness.
- Challenges of optimizing network performance without sacrificing features.
- Debugging and planning for networked systems.

Synchronization, Server Authority, and Optimizations:

- **Synchronization:** Achieved using NetworkVariable and ClientRpc to ensure consistent states across clients.
- **Server Authority:** All critical game logic, such as NPC movement and player scoring, is executed server-side to prevent cheating and ensure fairness.
- **Optimizations:**
  - Reduced the frequency of network updates for NPCs.
  - Employed object pooling for frequently instantiated objects like players and NPCs.
  - Minimized unnecessary RPC calls by grouping updates into fewer, consolidated messages.