```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>GRIDIRON 26: Pro-Framework Engine</title>
    <script src="https://cdnjs.cloudflare.com"></script>
    <style>
        :root { --primary: #00ff88; --secondary: #ff4400; }
        body { margin: 0; background: #000; color: #fff; font-family: 'Segoe UI', sans-serif; overflow:
hidden; }
        #hud { position: absolute; top: 0; width: 100%; display: flex; justify-content: space-between;
padding: 15px; pointer-events: none; z-index: 5; }
        .stat-box { background: rgba(10, 10, 10, 0.9); padding: 10px 15px; border-left: 3px solid
var(--primary); font-size: 14px; }
        #menu-overlay { position: absolute; inset: 0; background: rgba(0,0,0,0.95); display: flex;
align-items: center; justify-content: center; z-index: 100; pointer-events: all; }
        .panel { background: #111; padding: 25px; border: 1px solid #333; width: 600px;
max-height: 90vh; overflow-y: auto; border-radius: 8px; }
        .btn { background: var(--primary); color: #000; border: none; padding: 12px; font-weight:
900; cursor: pointer; width: 100%; margin: 5px 0; text-transform: uppercase; }
        #game-log { position: absolute; bottom: 15px; left: 15px; color: var(--primary); font-size:
12px; }
    </style>
</head>
<body>

<div id="hud">
    <div class="stat-box">
        <div style="font-size: 10px; color: #888;">QTR 1 | 15:00</div>
        <div id="ui-score">COL 0 - 0 TEX</div>
        <div id="ui-down" style="color: var(--primary); font-weight: bold;">1st & 10</div>
    </div>
    <div class="stat-box" style="text-align: right;">
        <div id="ui-player-name">PROSPECT</div>
        <div id="ui-ovr">OVR: 75</div>
        <div id="ui-xp" style="color: gold;">XP: 0</div>
    </div>
</div>

<div id="menu-overlay">
    <div class="panel" id="ui-content">
        <!-- Content injected by JavaScript -->
        <h1>Loading Engine...</h1>
    </div>
```

```
</div>

<div id="game-log">System: Engine Initialized. Ready for 2026 Season.</div>

<script>
const Game = {
    // --- Data Store & State ---
    player: {
        name: "Prospect", pos: "QB", ovr: 75, xp: 500, health: 100,
        stats: { spd: 70, str: 70, awr: 70 }, mesh: null, velocity: new THREE.Vector3()
    },
    gameState: {
        year: 2026, down: 1, ytg: 10, pos: -40, score: { home: 0, away: 0 },
        clock: 900, active: false, team: "Colorado", mode: "NCAA", playPhase: 'KICKOFF'
    },
    config: { difficulty: 'COLLEGE', baseSpeed: 0.08, weather: 'Clear' },

    // --- 3D Engine Properties ---
    scene: null, camera: null, renderer: null, entities: [], opponents: [], teammates: [], keys: {},
clock: new THREE.Clock(),

    initEngine() {
        this.scene = new THREE.Scene();
        this.scene.background = new THREE.Color(0x050505);
        this.camera = new THREE.PerspectiveCamera(75,
window.innerWidth/window.innerHeight, 0.1, 1000);
        this.renderer = new THREE.WebGLRenderer({ antialias: true });
        this.renderer.setSize(window.innerWidth, window.innerHeight);
        document.body.appendChild(this.renderer.domElement);

        this.setup3DEnvironment();
        window.addEventListener('keydown', e => this.keys[e.code] = true);
        window.addEventListener('keyup', e => this.keys[e.code] = false);
        window.addEventListener('resize', () => {
            this.camera.aspect = window.innerWidth / window.innerHeight;
            this.camera.updateProjectionMatrix();
            this.renderer.setSize(window.innerWidth, window.innerHeight);
        });

        this.showCustomizationUI();
        this.animate();
        this.log("System: 3D Engine Initialized. Customization ready.");
    },
```

```
    setup3DEnvironment() {
        this.scene.add(new THREE.AmbientLight(0xffffff, 0.8), new THREE.DirectionalLight(0xffffff,
1.5).position.set(10, 50, 10));
        const fieldGeo = new THREE.PlaneGeometry(120, 53.3);
        const fieldMat = new THREE.MeshPhongMaterial({ color: 0x1a3d16 });
        const field = new THREE.Mesh(fieldGeo, fieldMat);
        field.rotation.x = -Math.PI/2; field.position.y = -0.1;
        this.scene.add(field);
        // Add yard lines here for polish
    },

    // --- UI & CAREER MANAGEMENT ---
    showCustomizationUI() {
        document.getElementById('menu-overlay').style.display = 'flex';
        document.getElementById('ui-content').innerHTML = `
            <h1>GRIDIRON 26: CAREER HUB</h1>
            <h3>1. CUSTOMIZE CHARACTER</h3>
            <input type="text" id="p-name" placeholder="Player Name" class="btn"
style="background:#222; color:#fff; text-align:left;">
            <select id="p-pos" class="btn">
                <option value="QB">QB (Offense)</option><option value="HB">HB
(Offense)</option><option value="WR">WR (Offense)</option>
                <option value="LB">LB (Defense)</option><option value="DB">DB
(Defense)</option>
                <option value="K">K/P (Special Teams)</option>
            </select>
            <select id="p-diff" class="btn">
                <option value="ROOKIE">Rookie (Easiest)</option><option
value="COLLEGE">College (Normal)</option>
                <option value="PRO">Pro (Hard)</option><option value="ALL_TIME">All Time
(Legendary)</option>
            </select>
            <button class="btn" onclick="Game.startGameDay()">START SEASON / NEXT
GAME</button>
        `;
    },

    startGameDay() {
        this.player.name = document.getElementById('p-name').value || "Prospect";
        this.player.pos = document.getElementById('p-pos').value;
        this.config.difficulty = document.getElementById('p-diff').value;
        document.getElementById('menu-overlay').style.display = 'none';
        this.log("Starting game day. Weather is " + this.config.weather);
        this.updateUI();
```

```
    this.startPlay('KICKOFF');
},

// --- GAMEPLAY PHASE MANAGEMENT ---
startPlay(phase) {
    this.gameState.playPhase = phase;
    this.clearField();

    if (phase === 'KICKOFF') {
        this.simulateKickoff();
    } else if (this.isUserOffense()) {
        this.spawnOffense();
        this.gameState.active = true;
    } else if (this.isUserDefense()) {
        this.spawnDefense();
        this.gameState.active = true;
    } else { // Special Teams Spectate Mode
        this.simulateGenericPlay();
    }
},

isUserOffense() { return ['QB', 'HB', 'WR'].includes(this.player.pos); },
isUserDefense() { return ['LB', 'DB'].includes(this.player.pos); },

// --- SIMULATION LOGIC (Spectator Mode) ---
simulateKickoff() {
    this.log("KICKOFF: Spectator Mode Active. Waiting for return...");
    // In a full game, you'd run AI simulation loop here
    setTimeout(() => {
        // Assume the user's team gets the ball on the 25-yard line
        this.gameState.pos = -25;
        this.log("Return to the 25 yd line. Your Offense is up.");
        this.startPlay('OFFENSE');
    }, 3000);
},

simulateGenericPlay() {
    this.log("Spectating non-user play... Play in progress.");
    // Simulate a few seconds of AI play
    setTimeout(() => {
        // Assume random result
        this.log("Play finished. Next play loading.");
        this.startPlay(this.isUserOffense() ? 'OFFENSE' : 'DEFENSE');
    }, 2000);
```

```
},

// --- 3D SPAWNING & AI TACTICS ---
clearField() {
    this.entities.forEach(e => this.scene.remove(e));
    this.entities = []; this.opponents = []; this.teammates = [];
},

spawnOffense() {
    // User Player Spawn
    const pMat = new THREE.MeshPhongMaterial({ color: 0x00ff88 });
    this.player.mesh = new THREE.Mesh(new THREE.CapsuleGeometry(0.5, 1, 4, 8), pMat);
    this.player.mesh.position.set(this.gameState.pos, 1, 0);
    this.scene.add(this.player.mesh);
    this.entities.push(this.player.mesh);

    // AI Defense (Coverage Tactics)
    for(let i=0; i<11; i++) {
        const oppMat = new THREE.MeshPhongMaterial({ color: 0xff4400 });
        const opp = new THREE.Mesh(new THREE.CapsuleGeometry(0.5, 1, 4, 8), oppMat);
        // Simple Zone coverage AI spawn logic
        opp.position.set(this.gameState.pos + 10 + Math.random()*5, 1, (i-5)*4);
        opp.AI_ROLE = 'ZONE_DEEP'; // Assign AI Role
        this.scene.add(opp);
        this.opponents.push(opp);
    }
},

spawnDefense() {
    // User Player Spawn (Defense)
    const pMat = new THREE.MeshPhongMaterial({ color: 0xff4400 });
    this.player.mesh = new THREE.Mesh(new THREE.CapsuleGeometry(0.5, 1, 4, 8), pMat);
    this.player.mesh.position.set(this.gameState.pos + 5, 1, 0); // Start 5 yards off ball
    this.scene.add(this.player.mesh);
    this.entities.push(this.player.mesh);

    // AI Offense (Route Running Tactics)
    for(let i=0; i<11; i++) {
        const tm8Mat = new THREE.MeshPhongMaterial({ color: 0x00ff88 });
        const tm8 = new THREE.Mesh(new THREE.CapsuleGeometry(0.5, 1, 4, 8), tm8Mat);
        tm8.position.set(this.gameState.pos, 1, (i-5)*4);
        tm8.AI_ROLE = i % 2 === 0 ? 'GO_ROUTE' : 'FLAT_ROUTE'; // Assign AI Routes
        this.scene.add(tm8);
        this.teammates.push(tm8);
```

```
        }
    },


    // --- GAME LOOP AND PHYSICS ---
    updatePhysics() {
        if(!this.gameState.active) return;
        const delta = this.clock.getDelta();

        // Player Movement (User Input & Collision)
        const speedMultiplier = { ROOKIE: 0.8, COLLEGE: 1.0, PRO: 1.2, ALL_TIME: 1.5 };
        const baseSpeed = (this.player.stats.spd / 600) * speedMultiplier[this.config.difficulty];

        // ... (Player movement W/A/S/D logic omitted for brevity, identical to previous script) ...

        // AI Tactics Update
        this.opponents.forEach(opp => this.updateAITactic(opp, delta));
        this.teammates.forEach(team => this.updateAITactic(team, delta));
    },

    updateAITactic(aiPlayer, delta) {
        const difficultyFactor = { ROOKIE: 0.5, COLLEGE: 0.8, PRO: 1.0, ALL_TIME: 1.2
}[this.config.difficulty];
        const aiSpeed = Game.config.baseSpeed * difficultyFactor;

        // Offensive Routes
        if (aiPlayer.AI_ROLE === 'GO_ROUTE') {
            aiPlayer.position.x += aiSpeed * delta * 60;
        } else if (aiPlayer.AI_ROLE === 'FLAT_ROUTE') {
            if (aiPlayer.position.x < this.gameState.pos + 5) aiPlayer.position.x += aiSpeed * delta *
60;
            else aiPlayer.position.z += (aiPlayer.position.z > 0 ? -aiSpeed : aiSpeed) * delta * 60;
        }

        // Defensive Coverage (Simple Pursuit)
        if (aiPlayer.AI_ROLE === 'ZONE_DEEP') {
            const dist = aiPlayer.position.distanceTo(this.player.mesh.position);
            if(dist < 40) {
                aiPlayer.lookAt(this.player.mesh.position);
                aiPlayer.translateZ(aiSpeed * delta * 60 * 0.5); // Slower pursuit speed for zone
            }
        }
    },
```

```
    // --- REWARDS & UI LOGIC ---
    handleTackle() {
        // ... (existing tackle logic) ...
        this.player.xp += 25; // Reward XP for good defense
        this.updateUI();
        this.log("Tackle Made! +25 XP.");
    },

    handleScore() {
        // ... (existing score logic) ...
        this.player.xp += 150; // Reward XP for TD
        this.updateUI();
        this.log("TOUCHDOWN! +150 XP.");
    },

    updateUI() {
        // Update all UI elements from the game state
        document.getElementById('ui-player-name').innerText = this.player.name;
        document.getElementById('ui-ovr').innerText = `OVR: ${this.player.ovr} | Health:
${this.player.health}%`;
        document.getElementById('ui-xp').innerText = `XP: ${this.player.xp}`;
        document.getElementById('ui-score').innerText = `${this.gameState.team}
${this.gameState.score.home} - ${this.gameState.score.away} OPP`;
        document.getElementById('ui-down').innerText = `${this.gameState.down} &
${this.gameState.ytg}`;
    },

    log(msg) {
        document.getElementById('game-log').innerText = `System: ${msg}`;
    },

    animate() {
        requestAnimationFrame(() => this.animate());
        this.updatePhysics();
        this.renderer.render(this.scene, this.camera);
    }
};

window.onload = () => Game.initEngine();
</script>
</body>
</html>
```