

```

import random

class PlayerProfile:
    def __init__(self, name, position, year=2026):
        self.name = name
        self.position = position
        self.year = year
        self.ovr = 75
        self.xp = 0
        self.stats = {"SPD": 70, "STR": 70, "AWR": 70}
        self.is_draft_eligible = year >= 2026

    def process_offseason(self):
        """Logic for the 2026 Transfer Portal and Draft"""
        print(f"\n--- OFFSEASON {self.year} REPORT ---")

        # Skill upgrades based on XP
        growth = self.xp // 500
        self.ovr += growth
        for stat in self.stats:
            self.stats[stat] += random.randint(1, 3) + growth

        # Portal Logic
        if self.ovr > 85:
            print("ALERT: Blue-chip programs are scouting you in the Transfer Portal.")

        # NFL Draft Projection for 2026
        if self.is_draft_eligible:
            draft_grade = "First Round" if self.ovr > 90 else "Late Round"
            print(f"Draft Projection: {draft_grade}")

        self.year += 1
        return self.stats

# Example Simulation
user = PlayerProfile("Elite Prospect", "QB")
user.xp = 1500 # Player earned this in the JS engine
new_stats = user.process_offseason()
print(f"Upgraded Attributes: {new_stats} | New OVR: {user.ovr}")
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>GRIDIRON 26: UNIFIED ELITE ENGINE</title>

```

```

<script src="https://cdnjs.cloudflare.com"></script>
<style>
  :root { --p: #00ff88; --b: #050505; }
  body { margin: 0; background: var(--b); color: #fff; font-family: 'Inter', sans-serif; overflow: hidden; }
  #hud { position: absolute; top: 0; width: 100%; display: flex; justify-content: space-between; padding: 20px; pointer-events: none; z-index: 10; }
  .card { background: rgba(0,0,0,0.85); padding: 15px; border-left: 4px solid var(--p); }
  #overlay { position: absolute; inset: 0; background: rgba(0,0,0,0.9); display: flex; align-items: center; justify-content: center; z-index: 100; }
  .panel { background: #111; padding: 30px; border: 1px solid #333; width: 500px; text-align: center; }
  .btn { background: var(--p); color: #000; border: none; padding: 12px; font-weight: 800; cursor: pointer; width: 100%; margin: 5px 0; }
  #weather-overlay { position: absolute; inset: 0; pointer-events: none; }
</style>
</head>
<body>

<div id="hud">
  <div class="card">
    <div id="ui-game">2026 | WEEK 1</div>
    <div id="ui-down" style="color:var(--p); font-weight:900;">1ST & 10</div>
  </div>
  <div class="card" style="text-align:right;">
    <div id="ui-name">PROSPECT</div>
    <div id="ui-xp">XP: 0</div>
  </div>
</div>

<div id="overlay">
  <div class="panel" id="ui-content">
    <h1 style="color:var(--p);">GRIDIRON 26</h1>
    <select id="p-pos" class="btn"><option value="OFF">OFFENSE (QB/HB)</option><option value="DEF">DEFENSE (LB/DB)</option></select>
    <select id="p-diff" class="btn">
      <option value="1">ROOKIE</option><option value="2">PRO</option><option value="3">ALL-TIME</option>
    </select>
    <select id="p-weather" class="btn"><option value="Clear">CLEAR</option><option value="Rain">RAIN</option><option value="Snow">SNOW</option></select>
    <button class="btn" style="background:white;" onclick="Engine.launch()">START CAREER</button>
  </div>
</div>

```

```

</div>

<div id="weather-overlay"></div>

<script>
const Engine = {
  player: { pos: 'OFF', ovr: 75, xp: 0, mesh: null },
  settings: { difficulty: 1, weather: 'Clear' },
  scene: null, camera: null, renderer: null, entities: [],
  keys: {}, clock: new THREE.Clock(),

  launch() {
    this.player.pos = document.getElementById('p-pos').value;
    this.settings.difficulty = parseInt(document.getElementById('p-diff').value);
    this.settings.weather = document.getElementById('p-weather').value;
    document.getElementById('overlay').style.display = 'none';
    this.init3D();
    this.startPhase('KICKOFF');
  },
  init3D() {
    this.scene = new THREE.Scene();
    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);

    // Environment
    this.scene.add(new THREE.AmbientLight(0xffffff, 0.6));
    const field = new THREE.Mesh(new THREE.PlaneGeometry(120, 53.3), new
    THREE.MeshPhongMaterial({color: 0x13300d}));
    field.rotation.x = -Math.PI/2;
    this.scene.add(field);

    if(this.settings.weather === 'Snow') this.scene.background = new
    THREE.Color(0xdddddd);

    this.camera.position.set(-20, 15, 0);
    window.addEventListener('keydown', e => this.keys[e.code] = true);
    window.addEventListener('keyup', e => this.keys[e.code] = false);
    this.animate();
  },
}

```

```

startPhase(phase) {
  if (phase === 'KICKOFF') {
    console.log("Simulating Kickoff... Spectate Mode.");
    setTimeout(() => this.startPhase('GAMEPLAY'), 2000);
  } else {
    this.spawn11on11();
  }
},
};

spawn11on11() {
  const geo = new THREE.CapsuleGeometry(0.5, 1, 4, 8);
  // Player
  this.player.mesh = new THREE.Mesh(geo, new THREE.MeshPhongMaterial({color: 0x00ff88}));
  this.player.mesh.position.set(-30, 1, 0);
  this.scene.add(this.player.mesh);

  // AI 11-man Defense
  for(let i=0; i<11; i++) {
    const opp = new THREE.Mesh(geo, new THREE.MeshPhongMaterial({color: 0xff4400}));
    opp.position.set(-20 + Math.random()*5, 1, (i-5)*4);
    this.scene.add(opp);
    this.entities.push(opp);
  }
},
};

update() {
  if (!this.player.mesh) return;
  const speed = (0.1 * this.player.ovr/75) / this.settings.difficulty;

  // Handle Input
  if(this.keys['KeyW']) this.player.mesh.position.x += speed;
  if(this.keys['KeyS']) this.player.mesh.position.x -= speed;
  if(this.keys['KeyA']) this.player.mesh.position.z -= speed;
  if(this.keys['KeyD']) this.player.mesh.position.z += speed;

  // AI Pursuit Logic
  this.entities.forEach(opp => {
    const dist = opp.position.distanceTo(this.player.mesh.position);
    if(dist < 20) {
      opp.lookAt(this.player.mesh.position);
      opp.translateZ(0.05 * this.settings.difficulty);
    }
  })
}

```

```

        if(dist < 1.2) this.onTackle();
    });

this.camera.position.x = this.player.mesh.position.x - 15;
this.camera.lookAt(this.player.mesh.position);
},

onTackle() {
    this.player.xp += 50;
    document.getElementById('ui-xp').innerText = `XP: ${this.player.xp}`;
    alert("TACKLED! Skip Defense phase...");
    location.reload(); // Simple reset for demo
},

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

</script>
</body>
</html>

import random
import json
from datetime import datetime

class CareerEngine:
    def __init__(self, name, position, start_year=2026):
        # 1. Core Profile
        self.profile = {
            "name": name,
            "position": position,
            "year": start_year,
            "team": "Prospect University",
            "ovr": 75,
            "xp_bank": 1000,
            "status": "Active",
            "league": "NCAA"
        }

        # 2. Advanced Skill Tree (OVR is average of these)
        self.skills = {
            "SPD": 70, "STR": 70, "AWR": 75, # Physicals

```

```

        "THP": 65, "THA": 60,      # Passing (if QB)
        "TAK": 50, "ZCV": 40      # Defensive
    }

# 3. Career Performance Log
self.stats = {"tds": 0, "yards": 0, "ints": 0, "awards": []}
self.draft_stock = "Undrafted"

def record_game(self, yds, tds, mistakes):
    """Simulates result from JS 3D Engine and awards XP"""
    self.stats["yards"] += yds
    self.stats["tds"] += tds
    xp_earned = (yds * 2) + (tds * 100) - (mistakes * 50)
    self.profile["xp_bank"] += max(0, xp_earned)
    self.update_ovr()
    return f"Game Processed: Earned {xp_earned} XP."

def update_ovr(self):
    """Dynamic OVR calculation based on role-specific stats"""
    relevant_stats = list(self.skills.values())
    self.profile["ovr"] = int(sum(relevant_stats) / len(relevant_stats))

def draft_projection(self):
    """Calculates 2026 NFL Draft Stock based on OVR and Stats"""
    ovr = self.profile["ovr"]
    if ovr > 90 and self.stats["tds"] > 30:
        self.draft_stock = "1st Round (Top 10)"
    elif ovr > 85:
        self.draft_stock = "1st Round"
    elif ovr > 80:
        self.draft_stock = "Late Round"
    else:
        self.draft_stock = "Undrafted"
    return self.draft_stock

def transfer_portal(self):
    """Simulates 2026 NCAA Transfer Portal opportunities"""
    offers = []
    if self.profile["ovr"] > 80:
        offers = ["Texas", "Georgia", "Ohio State"]
    elif self.profile["ovr"] > 70:
        offers = ["Colorado", "Oregon"]
    return random.sample(offers, min(len(offers), 3))

```

```

def save_career(self):
    """Persists the 2026 career state to JSON"""
    filename = f"{self.profile['name']}_career.json"
    with open(filename, 'w') as f:
        json.dump({**self.profile, **self.skills, **self.stats}, f)
    return f"Career saved to {filename}"

# --- 2026 SEASON EXECUTION EXAMPLE ---
player = CareerEngine("Devin Wade", "QB", 2026)

# Simulate 3 weeks of the 2026 Season
for week in range(1, 4):
    print(f"Week {week}: {player.record_game(250, 3, 1)}")

# Trigger Offseason Logic
print(f"Transfer Portal Offers: {player.transfer_portal()}")
print(f"NFL Draft Grade: {player.draft_projection()}")
print(player.save_career())
import random
import json
from datetime import datetime

class CareerEngine:
    def __init__(self, name, position, start_year=2026):
        self.profile = {
            "name": name,
            "position": position,
            "year": start_year,
            "team": "Prospect University",
            "ovr": 75,
            "xp_bank": 1000,
            "status": "Active",
            "discipline": 50 # New stat: higher discipline = better route runner, higher XP potential
        }
        self.skills = {"SPD": 70, "STR": 70, "AWR": 75, "THP": 65, "THA": 60, "TAK": 50, "ZCV": 40}
        self.stats = {"tds": 0, "yards": 0, "ints": 0, "awards": []}
        self.draft_stock = "Undrafted"

    def record_game(self, yds, tds, mistakes):
        """Simulates result from JS 3D Engine and awards XP"""
        # XP calculation uses discipline stat as a multiplier
        discipline_bonus = self.profile['discipline'] / 100
        xp_earned = (yds * 2) + (tds * 100) - (mistakes * 50)
        total_xp = max(0, xp_earned * (1 + discipline_bonus)) # Disciplined players get more value

```

```

        self.stats["yards"] += yds
        self.stats["tds"] += tds
        self.profile["xp_bank"] += total_xp
        self.update_ovr()
        return f"Game Processed: Earned {total_xp:.0f} XP."
    }

    def update_ovr(self):
        relevant_stats = list(self.skills.values())
        self.profile["ovr"] = int(sum(relevant_stats) / len(relevant_stats))

    # ... (rest of Python code remains the same as previous response) ...
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>GRIDIRON 26: Pro-Framework Engine v2</title>
    <script src="https://cdnjs.cloudflare.com"></script>
    <style>
        :root { --p: #00ff88; --b: #050505; }
        body { margin: 0; background: var(--b); color: #fff; font-family: 'Inter', sans-serif; overflow: hidden; }
        #hud { position: absolute; top: 0; width: 100%; display: flex; justify-content: space-between; padding: 15px; pointer-events: none; z-index: 10; }
        .card { background: rgba(0,0,0,0.85); padding: 10px 15px; border-left: 3px solid var(--p); }
        #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: 500px; text-align: center; border-radius: 8px; }
        .btn { background: var(--p); color: #000; border: none; padding: 12px; font-weight: 800; cursor: pointer; width: 100%; margin: 5px 0; text-transform: uppercase; }
        #game-log { position: absolute; bottom: 15px; left: 15px; color: var(--p); font-size: 12px; }
    </style>
</head>
<body>
<div id="hud">
    <div class="card">
        <div id="ui-game">2026 | WEEK 1</div>
        <div id="ui-down" style="color:var(--p); font-weight:900;">1ST & 10</div>
    </div>
    <div class="card" style="text-align:right;">
        <div id="ui-name">PROSPECT</div>
        <div id="ui-xp">XP: 0</div>
    </div>

```

```

</div>
<div id="overlay">
  <div class="panel" id="ui-content">
    <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: 0, health: 100, mesh: null, role: 'OFF'
  },
  gameState: {
    active: false, avgRouteDeviation: 0, playCalled: false
  },
  // --- 3D Engine Properties & Playbook Logic ---
  scene: null, camera: null, renderer: null, entities: [], opponents: [], teammates: [],
  keys: {}, clock: new THREE.Clock(), playArrows: [], currentRoutePath: [],

  initEngine() {
    this.scene = new THREE.Scene();
    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();
    // ... (Input Listeners same as previous code) ...
    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();
  },
  setup3DEnvironment() {
    // ... (3D environment setup same as previous code) ...
  }
}

```

```

        this.scene.add(new THREE.AmbientLight(0xffffff, 0.8), new
THREE.DirectionalLight(0xffffff, 1.5).position.set(10, 50, 10));
        const fieldMat = new THREE.MeshPhongMaterial({ color: 0x13300d });
        const field = new THREE.Mesh(new THREE.PlaneGeometry(120, 53.3), fieldMat);
        field.rotation.x = -Math.PI/2; field.position.y = -0.1;
        this.scene.add(field);
    },

// --- UI & PLAYBOOK ---
showCustomizationUI() { /* ... (same as previous code) ... */
    document.getElementById('menu-overlay').style.display = 'flex';
    document.getElementById('ui-content').innerHTML =
        `

# GRIDIRON 26: CAREER HUB


        <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>
        </select>
        <button class="btn" onclick="Game.startGameDay()">START SEASON / NEXT
GAME</button>
    `;
},
startGameDay() { /* ... (same as previous code) ... */
    this.player.name = document.getElementById('p-name').value || "Prospect";
    this.player.pos = document.getElementById('p-pos').value;
    this.player.role = ['QB', 'HB', 'WR'].includes(this.player.pos) ? 'OFF' : 'DEF';
    document.getElementById('menu-overlay').style.display = 'none';
    this.startPlay('KICKOFF');
},
showPlaybookUI() {
    if (this.player.role === 'DEF') { this.simulateOtherPhase("Offense"); return; }
    document.getElementById('menu-overlay').style.display = 'flex';
    document.getElementById('ui-content').innerHTML =
        `

# PLAYBOOK: Select Your Route


        <button class="btn" onclick="Game.callPlay('GO_ROUTE')">GO ROUTE (High
XP)</button>
        <button class="btn" onclick="Game.callPlay('SLANT_ROUTE')">SLANT ROUTE (Med
XP)</button>

```

```

        <button class="btn" onclick="Game.callPlay('DRAG_ROUTE')">DRAG ROUTE (Safe
XP)</button>
        <button class="btn" onclick="Game.callPlay('FREESTYLE_RUN')"
style="background:#555;">FREESTYLE RUN (Low XP)</button>
    ;
    this.spawnOffense(true);
},

callPlay(routeType) {
    document.getElementById('menu-overlay').style.display = 'none';
    this.gameState.playCalled = routeType;
    this.drawRoutePath(this.player.mesh.position.clone(), routeType);
    this.gameState.active = true;
},
drawRoutePath(startPos, routeType) {
    this.clearArrows();
    this.currentRoutePath = [] // Array of points the user should follow

    if (routeType === 'FREESTYLE_RUN') return; // No path to follow for freestyle

    let points = [startPos];
    if (routeType === 'GO_ROUTE') { points.push(new THREE.Vector3(startPos.x + 40, 1,
startPos.z)); }
    else if (routeType === 'SLANT_ROUTE') { points.push(new THREE.Vector3(startPos.x +
10, 1, startPos.z + 10)); }
    else if (routeType === 'DRAG_ROUTE') { points.push(new THREE.Vector3(startPos.x +
20, 1, startPos.z + 20)); }

    // Generate visible arrows along the path
    for (let i = 0; i < points.length - 1; i++) {
        const dir = points[i+1].clone().sub(points[i]).normalize();
        const length = points[i].distanceTo(points[i+1]);
        const arrow = new THREE.ArrowHelper(dir, points[i], length, 0xffff00, 2, 1);
        this.scene.add(arrow);
        this.playArrows.push(arrow);
    }
    this.currentRoutePath = points;
},
clearArrows() {
    this.playArrows.forEach(arrow => this.scene.remove(arrow));
    this.playArrows = [];
},

```

```

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

    // Player movement uses standard WASD input
    const speed = (this.player.ovr / 75) * 0.1;
    if(this.keys['KeyW']) this.player.mesh.position.x += speed;
    if(this.keys['KeyS']) this.player.mesh.position.x -= speed;
    if(this.keys['KeyA']) this.player.mesh.position.z -= speed;
    if(this.keys['KeyD']) this.player.mesh.position.z += speed;

    this.trackRouteDiscipline();
    // ... (AI pursuit logic remains same as previous code) ...

    this.camera.position.x = this.player.mesh.position.x - 15;
    this.camera.lookAt(this.player.mesh.position);
},
trackRouteDiscipline() {
    if (this.currentRoutePath.length === 0) return;

    // Simple measure: distance to the next point on path
    const nextPoint = this.currentRoutePath[1] || this.currentRoutePath[0];
    const distanceToPath = this.player.mesh.position.distanceTo(nextPoint);
    // Accumulate deviation over the play duration
    this.gameState.avgRouteDeviation += distanceToPath * this.clock.getDelta();
},
// --- REWARDS & UI LOGIC ---
handleTackle() {
    if (!this.gameState.active) return;
    this.gameState.active = false;

    // Calculate XP based on performance and discipline
    const baseXP = 25;
    let bonusXP = 0;
    if (this.gameState.playCalled !== 'FREESTYLE_RUN') {
        // Higher deviation means less XP
        const disciplineScore = Math.max(0, 100 - (this.gameState.avgRouteDeviation / 0.5));
        bonusXP = baseXP * (disciplineScore / 100);
        this.log(`Tackle! Route Discipline Score: ${disciplineScore.toFixed(0)}%.`);
    } else {
}

```

```

        this.log("Tackle! (Freestyle run, low bonus potential)");
    }

    this.player.xp += Math.floor(baseXP + bonusXP);
    this.updateUI();
    setTimeout(() => this.startPlay('GAMEPLAY'), 3000);
},
// ... (handleScore remains similar) ...

updateUI() {
    document.getElementById('ui-name').innerText = this.player.name;
    document.getElementById('ui-xp').innerText = `XP: ${this.player.xp}`;
    // ... (score and down UI updates) ...
},
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>
import random
import json
from datetime import datetime

class CareerEngine:
    # ... (init and existing methods like record_game, update_ovr, save_career) ...
    # ... (These remain unchanged from the previous Python code) ...
    def __init__(self, name, position, start_year=2026):
        # 1. Core Profile
        self.profile = {
            "name": name,
            "position": position,
            "year": start_year,
            "team": "Prospect University",
            "ovr": 75,

```

```

        "xp_bank": 1000,
        "status": "Active",
        "discipline": 50
    }
    self.skills = {"SPD": 70, "STR": 70, "AWR": 75, "THP": 65, "THA": 60, "TAK": 50, "ZCV": 40}
    self.stats = {"tds": 0, "yards": 0, "ints": 0, "awards": []}
    self.draft_stock = "Undrafted"

def record_game(self, yds, tds, mistakes):
    discipline_bonus = self.profile['discipline'] / 100
    xp_earned = (yds * 2) + (tds * 100) - (mistakes * 50)
    total_xp = max(0, xp_earned * (1 + discipline_bonus))

    self.stats["yards"] += yds
    self.stats["tds"] += tds
    self.profile["xp_bank"] += total_xp
    self.update_ovr()
    return f"Game Processed: Earned {total_xp:.0f} XP."

def update_ovr(self):
    relevant_stats = list(self.skills.values())
    self.profile["ovr"] = int(sum(relevant_stats) / len(relevant_stats))

# --- NEW: Scouting and Player Generation Logic ---
def generate_prospects(self, count=5, min_ovr=60, max_ovr=90):
    """Generates a list of new players for recruiting/draft classes."""
    first_names = ["Jackson", "Liam", "Noah", "Aiden", "Caleb", "Mason", "Ethan"]
    last_names = ["Smith", "Johnson", "Williams", "Brown", "Jones", "Garcia", "Miller"]
    positions = ["QB", "HB", "WR", "LB", "DE", "CB"]
    prospects = []

    for _ in range(count):
        ovr = random.randint(min_ovr, max_ovr)
        potential = random.randint(ovr + 5, 99)
        stars = 5 if ovr >= 85 else (4 if ovr >= 75 else 3)

        prospect = {
            "name": f"{random.choice(first_names)} {random.choice(last_names)}",
            "pos": random.choice(positions),
            "OVR": ovr,
            "POT": potential,
            "Stars": stars,
            "Summary": f"{stars}-star prospect with OVR of {ovr}. High potential ceiling."
        }
        prospects.append(prospect)

    return prospects

```

```

        prospects.append(prospect)
    return prospects

# ... (draft_projection and transfer_portal methods remain unchanged) ...
def draft_projection(self):
    ovr = self.profile["ovr"]
    if ovr > 90 and self.stats["tds"] > 30:
        self.draft_stock = "1st Round (Top 10)"
    elif ovr > 85:
        self.draft_stock = "1st Round"
    elif ovr > 80:
        self.draft_stock = "Late Round"
    else:
        self.draft_stock = "Undrafted"
    return self.draft_stock

def transfer_portal(self):
    offers = []
    if self.profile["ovr"] > 80:
        offers = ["Texas", "Georgia", "Ohio State"]
    elif self.profile["ovr"] > 70:
        offers = ["Colorado", "Oregon"]
    return random.sample(offers, min(len(offers), 3))

def save_career(self):
    filename = f"{self.profile['name']}_career.json"
    with open(filename, 'w') as f:
        json.dump({**self.profile, **self.skills, **self.stats}, f)
    return f"Career saved to {filename}"

```

```

# --- 2026 SEASON EXECUTION EXAMPLE ---
player = CareerEngine("Devin Wade", "QB", 2026)
print(f"Generated a new QB prospect class for 2026:")
recruits = player.generate_prospects(count=3, min_ovr=70, max_ovr=88)
for r in recruits:
    print(r["Summary"])

print("\nSimulating Season (XP gained via JavaScript Engine)...")
# Assume XP is updated here
player.profile['xp_bank'] += 2500

print(f"\nOffseason Results for {player.profile['name']}:")
print(f"Transfer Portal Offers: {player.transfer_portal()}")

```

```

print(f"NFL Draft Grade: {player.draft_projection()}")
import random
import json
import uuid
from datetime import datetime

class GridironEliteCore:
    def __init__(self, player_name, position):
        self.session_id = str(uuid.uuid4())[:8]
        self.timestamp = datetime.now().strftime("%Y-%m-%d %H:%M")

        # 1. Persistent Career Stats
        self.career = {
            "player": player_name,
            "position": position,
            "ovr": 78,
            "xp": 1200,
            "stamina_cap": 100,
            "wear_and_tear": 0.0, # 0.0 (Fresh) to 1.0 (Critical)
            "discipline_rating": 85.0
        }

        # 2. League State 2026
        self.league_context = {
            "national_rank": 25,
            "bowl_projection": "None",
            "conference_standing": "Mid-Table",
            "heisman_odds": "+5000"
        }

    def simulate_sideline_plays(self, game_time_seconds):
        """
        Simulates the logic when the player is NOT on the field.
        Calculates field position and score changes for the 3D Engine to resume.
        """

        plays_simulated = game_time_seconds // 30
        points_scored = 0
        yardage_change = 0

        for _ in range(plays_simulated):
            # Weighted probability based on OVR
            if random.random() > 0.6: # Team success rate
                points_scored += random.choice([0, 3, 7])
                yardage_change += random.randint(20, 50)

```

```

        else:
            yardage_change -= random.randint(5, 15)

    return {
        "sim_score": points_scored,
        "new_field_pos": yardage_change,
        "msg": f"Simulated {plays_simulated} plays while you were on the sideline."
    }
}

def calculate_wear_and_tear(self, snaps_played, big_hits_taken):
    """
    Advanced 2026 Injury Logic.
    Wear and tear reduces SPD and STR stats in the JavaScript engine.
    """

    damage = (snaps_played * 0.001) + (big_hits_taken * 0.05)
    self.career["wear_and_tear"] = min(1.0, self.career["wear_and_tear"] + damage)

    # Recovery logic (off-day simulation)
    if self.career["wear_and_tear"] > 0.1:
        self.career["wear_and_tear"] -= 0.02

    status = "HEALTHY" if self.career["wear_and_tear"] < 0.3 else "QUESTIONABLE"
    if self.career["wear_and_tear"] > 0.7: status = "INJURED"

    return {"value": round(self.career["wear_and_tear"], 2), "status": status}

def evaluate_draft_premium(self):
    """
    Determines the 'Signing Bonus' and Round based on 2026 performance.
    """

    ovr = self.career["ovr"]
    disc = self.career["discipline_rating"]

    # Combined Score for NFL Scouts
    scout_score = (ovr * 0.7) + (disc * 0.3)

    if scout_score > 90: return {"Round": 1, "Bonus": "$25,000,000"}
    if scout_score > 80: return {"Round": 2, "Bonus": "$12,000,000"}
    return {"Round": "Undrafted Free Agent", "Bonus": "$50,000"}

# --- SYSTEM EXECUTION ---
if __name__ == "__main__":
    # Create the 2026 Prospect
    engine = GridironEliteCore("Devin Wade", "QB")

```

```

# 1. Player is on the sidelines (Offense chose, Defense is playing)
sideline_update = engine.simulate_sideline_plays(300) # 5 minutes of game time
print(f"[{engine.timestamp}] {sideline_update['msg']}")
print(f"Points Scored by Team: {sideline_update['sim_score']}")

# 2. Player takes a 'Big Hit' in the JS Engine
health_report = engine.calculate_wear_and_tear(snaps_played=45, big_hits_taken=3)
print(f"Medical Report: {health_report['status']} (Wear: {health_report['value']}"))

# 3. Final Draft Grade for the 2026 Season
draft_eval = engine.evaluate_draft_premium()
print(f"NFL Scouting Bureau: Projected Round {draft_eval['Round']} | Bonus: {draft_eval['Bonus']}")

CREATE TABLE player_career_2026 (
    player_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    player_name VARCHAR(50) NOT NULL,
    position CHAR(2) CHECK (position IN ('QB', 'HB', 'WR', 'LB', 'DB')),
    ovr INT DEFAULT 75,
    xp_balance BIGINT DEFAULT 0,
    discipline_score DECIMAL(5,2) DEFAULT 100.00,
    wear_and_tear DECIMAL(3,2) DEFAULT 0.00,
    draft_stock_rank INT DEFAULT 256,
    current_team VARCHAR(50) DEFAULT 'Prospect University',
    is_injured BOOLEAN DEFAULT FALSE,
    last_updated TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
#include <emsCripten.h>
#include <cmath>

extern "C" {
    // Calculates the "Hit Stick" momentum and resulting injury risk
    EMSCRIPTEN_KEEPALIVE
    float calculate_impact_force(float mass, float velocity, float defender_ovr) {
        float base_momentum = mass * velocity;
        // High OVR defenders deliver harder hits in 2026
        float force = base_momentum * (defender_ovr / 75.0);
        return force;
    }

    EMSCRIPTEN_KEEPALIVE
    int check_injury_probability(float impact_force, float wear_tear) {
        // If force is over 800 and wear is high, 45% injury chance
        if (impact_force > 800.0f && wear_tear > 0.5f) return 1;
    }
}

```

```

        return 0;
    }
}

from fastapi import FastAPI
import random

app = FastAPI()

@app.get("/scout/draft_projection/{ovr}/{discipline}")
async def get_draft_projection(ovr: int, discipline: float):
    # 2026 logic: Discipline acts as a multiplier for Draft Stock
    stock_score = (ovr * 0.6) + (discipline * 0.4)

    if stock_score > 92: return {"round": "Top 5 Pick", "bonus": "$35M"}
    if stock_score > 85: return {"round": "1st Round", "bonus": "$20M"}
    return {"round": "Day 3", "bonus": "$2M"}

@app.post("/game/simulate_defense")
async def sim_defense(team_ovr: int):
    # Logic for skipping defensive plays while user is an offensive character
    points_allowed = random.randint(0, 14) if team_ovr > 85 else random.randint(7, 28)
    return {"points_allowed": points_allowed, "turnovers": random.randint(0, 2)}

#grid-hud {
    border-top: 2px solid #00ff88;
    background: linear-gradient(180deg, rgba(0,255,136,0.1) 0%, rgba(0,0,0,0.9) 100%);
    box-shadow: 0px 0px 20px rgba(0, 255, 136, 0.5);
    clip-path: polygon(0% 0%, 100% 0%, 95% 100%, 5% 100%);
    font-family: 'Orbitron', sans-serif;
    animation: pulse 2s infinite;
}

@keyframes pulse {
    0% { opacity: 0.9; }
    50% { opacity: 1; }
    100% { opacity: 0.9; }
}

const Engine2026 = {
    // 3D Scene Properties
    activeRoute: [],

    drawPlayArt(routeType) {
        // Logic to draw yellow arrows on the field
        const start = this.player.mesh.position;
        const directions = {

```

```

'SLANT': new THREE.Vector3(15, 0, 10),
'GO': new THREE.Vector3(40, 0, 0)
};

const arrow = new THREE.ArrowHelper(
  directions[routeType].clone().normalize(),
  start,
  directions[routeType].length(),
  0xffff00, 2, 1
);
this.scene.add(arrow);
this.activeRoute = directions[routeType];
console.log(`2026 PlayArt Rendered: ${routeType}`);
},

processDisciplineXP(playerPos) {
  // Compare player distance to activeRoute
  const deviation = playerPos.distanceTo(this.activeRoute);
  let xpGain = 0;

  if (deviation < 2.0) {
    xpGain = 500; // Reward for perfect 2026 route running
    this.log("PERFECT ROUTE: +500 XP (Discipline Bonus)");
  } else {
    xpGain = 50; // Minimal reward for freestyle
    this.log("FREESTYLE RUN: +50 XP (Scouts Disapprove)");
  }
  return xpGain;
},

skipToNextPossession(role) {
  // Connects to Python Sim API
  this.log(`Simulating ${role === 'OFF' ? 'Defense' : 'Offense'}...`);
  // Visual "Continue" screen would show here
  document.getElementById('ui-continue').style.display = 'block';
}

};

import random
import uuid
from dataclasses import dataclass
from typing import List, Dict

@dataclass
class ProspectClass:

```

```

"""Generates the 2026-2027 Scouting Class to compete with the player."""
rank: int
name: str
stars: int
commitment: str = "Uncommitted"

class CareerIntelligence:
    def __init__(self, player_name: str, position: str):
        self.player_id = f"GR26-{uuid.uuid4().hex[:6].upper()}"
        self.player_name = player_name
        self.position = position

    # 2026 Performance Metrics
    self.metrics = {
        "hype_rating": 1500,      # Influences NIL and Portal offers
        "coach_trust": 50,       # 0-100: Higher opens more playbook routes
        "nil_valuation": 5000.00, # Actual currency for upgrades
        "fan_base": 10000        # Social media following
    }

    # League Landscape
    self.transfer_portal_active = False

def calculate_nil_payout(self, game_performance_score: float):
    """
    Calculates 2026 NIL earnings.
    Good route running increases Hype, which multiplies payout.
    """

    performance_modifier = game_performance_score / 100
    market_multiplier = self.metrics["hype_rating"] / 1000

    payout = (1000 * market_multiplier) * performance_modifier
    self.metrics["nil_valuation"] += payout
    self.metrics["fan_base"] += int(payout * 0.5)

    return {
        "payout": f"${payout:.2f}",
        "total_value": f"${self.metrics['nil_valuation']:.2f}",
        "hype": self.metrics["hype_rating"]
    }

def simulate_transfer_portal(self, current_ovr: int):
    """
    Logic for the 2026 Transfer Portal.
    """

```

```

If your OVR is high, 'Power 4' schools will attempt to poach you.
"""

offers = []
tier_1 = ["Georgia Bulldogs", "Ohio State Buckeyes", "Texas Longhorns"]
tier_2 = ["Colorado Buffaloes", "Miami Hurricanes", "Oregon Ducks"]

if current_ovr >= 90:
    offers = random.sample(tier_1, 2) + random.sample(tier_2, 1)
elif current_ovr >= 80:
    offers = random.sample(tier_2, 2)

self.transfer_portal_active = len(offers) > 0
return {
    "portal_status": "OPEN" if self.transfer_portal_active else "CLOSED",
    "interested_programs": offers,
    "nil_incentive": f"${random.randint(50000, 250000)}" if offers else "$0"
}

def update_coach_trust(self, route_discipline_score: float):
    """
    Directly impacts the JS Playbook.
    High discipline allows the user to call 'Audibles' in 2026.
    """

    if route_discipline_score > 90:
        self.metrics["coach_trust"] += 5
    elif route_discipline_score < 50:
        self.metrics["coach_trust"] -= 10

    self.metrics["coach_trust"] = max(0, min(100, self.metrics["coach_trust"]))

    audible_unlocked = self.metrics["coach_trust"] >= 85
    return {"trust_level": self.metrics["coach_trust"], "audibles": audible_unlocked}

# --- 2026 SEASON LIVE SIMULATION ---
if __name__ == "__main__":
    # Initialize Player for the 2026 Campaign
    prospect = CareerIntelligence("Devin Wade", "WR")

    # 1. Post-Game Route Discipline Report (From JS Engine)
    # Player followed the yellow arrows 95% of the time.
    trust_report = prospect.update_coach_trust(95.0)
    print(f"Coach Trust: {trust_report['trust_level']}% | Audibles: {trust_report['audibles']}")

    # 2. Financial & Hype Update

```

```

# Based on a 150-yard, 2-TD performance
nil_report = prospect.calculate_nil_payout(150.0)
print(f"NIL Payout: {nil_report['payout']} | Valuation: {nil_report['total_value']}")

# 3. Mid-Season Portal Check
# If player is OVR 85 by mid-2026
portal_intel = prospect.simulate_transfer_portal(85)
if portal_intel['interested_programs']:
    print(f"PORTAL ALERT: {portal_intel['interested_programs']} are offering NIL deals up to
{portal_intel['nil_incentive']}")

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>GRIDIRON 26: ELITE UNIFIED ENGINE</title>
    <script src="https://cdnjs.cloudflare.com"></script>
    <style>
        :root { --neon: #00ff88; --warn: #ff4400; --bg: #050505; }
        body { margin: 0; background: var(--bg); color: #fff; font-family: 'Orbitron', sans-serif;
        overflow: hidden; }

        /* 2026 NEON HUD */
        #hud { position: absolute; top: 0; width: 100%; z-index: 10; pointer-events: none; }
        .neon-bar { background: rgba(0,0,0,0.85); border-bottom: 2px solid var(--neon); padding:
        15px; display: flex; justify-content: space-between; box-shadow: 0 0 20px rgba(0,255,136,0.3); }
        .stat-group { display: flex; gap: 20px; align-items: center; }
        .chem-meter { width: 100px; height: 8px; background: #222; border: 1px solid var(--neon); }
        .chem-fill { height: 100%; background: var(--neon); width: 50%; transition: 0.5s; }

        /* OVERLAYS */
        #overlay { position: absolute; inset: 0; background: rgba(0,0,0,0.95); display: flex;
        align-items: center; justify-content: center; z-index: 100; }
        .panel { background: #111; padding: 30px; border: 1px solid #333; width: 500px; text-align:
        center; border-radius: 10px; }
        .btn { background: var(--neon); color: #000; border: none; padding: 12px; font-weight: 900;
        cursor: pointer; width: 100%; margin: 8px 0; text-transform: uppercase; }
        .news-ticker { background: #000; color: gold; padding: 5px; font-size: 11px; text-align:
        center; border-bottom: 1px solid #333; }
        #game-log { position: absolute; bottom: 20px; left: 20px; color: var(--neon); font-size: 12px;
        text-shadow: 0 0 5px #000; }

    </style>
</head>
<body>

```

```

<div id="hud">
  <div class="news-ticker" id="ui-news">SCOUTING REPORT: JANUARY 2026 - DRAFT
  STOCK STABLE</div>
  <div class="neon-bar">
    <div class="stat-group">
      <div><small>CHEMISTRY</small><div class="chem-meter"><div id="chem-fill"
      class="chem-finish"></div></div></div>
      <div><small>HYPE</small><div id="ui-hype" style="color:var(--neon)">1500</div></div>
    </div>
    <div class="stat-group" style="text-align: right;">
      <div id="ui-score">COL 0 - 0 TEX</div>
      <div id="ui-clock" style="color:var(--warn)">15:00</div>
    </div>
  </div>
</div>

<div id="overlay">
  <div class="panel" id="ui-content">
    <h1 style="color:var(--neon); margin:0;">GRIDIRON 26</h1>
    <p style="font-size: 10px; color: #888;">POLYGLOT CAREER ENGINE v1.0</p>
    <input type="text" id="p-name" placeholder="ENTER NAME" class="btn"
    style="background:#222; color:#fff; text-align:center;">
    <select id="p-pos" class="btn">
      <option value="OFF">OFFENSE (QB/WR)</option>
      <option value="DEF">DEFENSE (LB/CB)</option>
    </select>
    <select id="p-weather" class="btn">
      <option value="Clear">CLEAR SKIES</option>
      <option value="Rain">HEAVY RAIN (Slippery)</option>
      <option value="Snow">SNOW (Low Traction)</option>
    </select>
    <button class="btn" onclick="Engine.initGame()">BREAK HUDDLE</button>
  </div>
</div>

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

<script>
/**
 * GRIDIRON 26: UNIFIED SYSTEM ARCHITECTURE
 * Languages Emulated: JS (Graphics), C++ (Physics), Python (Career Logic)
 */
const Engine = {
  // Career State (Python Logic Emulation)

```

```

career: { name: "", pos: "OFF", ovr: 78, chemistry: 50, hype: 1500, xp: 0, draftStock: "Late Round" },
game: { score: [0, 0], clock: 900, active: false, weather: "Clear", currentDown: 1 },

// 3D Engine (Three.js)
scene: null, camera: null, renderer: null, player: null, opponents: [],
keys: {}, clock: new THREE.Clock(),
routeArrow: null,

initGame() {
    // Capture UI data
    this.career.name = document.getElementById('p-name').value || "Prospect";
    this.career.pos = document.getElementById('p-pos').value;
    this.game.weather = document.getElementById('p-weather').value;

    document.getElementById('overlay').style.display = 'none';
    this.init3D();
    this.log(`2026 Career Started: ${this.career.name} as ${this.career.pos}`);
    this.showPlaybook();
},
init3D() {
    this.scene = new THREE.Scene();
    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);

    // Environment
    const amb = new THREE.AmbientLight(0xffffff, 0.6);
    const sun = new THREE.DirectionalLight(0xffffff, 1.2);
    sun.position.set(5, 20, 10);
    this.scene.add(amb, sun);

    const grassColor = this.game.weather === 'Snow' ? 0xdddddd : 0x13300d;
    const field = new THREE.Mesh(new THREE.PlaneGeometry(150, 60), new
    THREE.MeshPhongMaterial({color: grassColor}));
    field.rotation.x = -Math.PI/2;
    this.scene.add(field);

    // Add Yard Lines
    for(let i=-60; i<=60; i+=10) {

```

```

        const line = new THREE.Mesh(new THREE.PlaneGeometry(0.3, 60), new
THREE.MeshBasicMaterial({color: 0xffffffff, transparent: true, opacity: 0.3}));
        line.rotation.x = -Math.PI/2; line.position.set(i, 0.01, 0);
        this.scene.add(line);
    }

    this.spawnEntities();

    window.addEventListener('keydown', e => this.keys[e.code] = true);
    window.addEventListener('keyup', e => this.keys[e.code] = false);
    this.animate();
},
spawnEntities() {
    const geo = new THREE.CapsuleGeometry(0.5, 1, 4, 8);
    // Player
    this.player = new THREE.Mesh(geo, new THREE.MeshPhongMaterial({color: 0x00ff88}));
    this.player.position.set(-40, 1, 0);
    this.scene.add(this.player);

    // 11 Defenders
    for(let i=0; i<11; i++) {
        const opp = new THREE.Mesh(geo, new THREE.MeshPhongMaterial({color:
0xff4400}));
        opp.position.set(-30 + Math.random()*5, 1, (i-5)*5);
        this.scene.add(opp);
        this.opponents.push(opp);
    }
},
showPlaybook() {
    if(this.career.pos === 'DEF') { this.simulatePhase("Offense"); return; }

    document.getElementById('overlay').style.display = 'flex';
    document.getElementById('ui-content').innerHTML =
      <h2 style="color:var(--neon)">SELECT ROUTE</h2>
      <button class="btn" onclick="Engine.callPlay('SLANT')">SLANT (Quick XP)</button>
      <button class="btn" onclick="Engine.callPlay('STREAK')">STREAK (High
Hype)</button>
      <p style="font-size:10px">Discipline Bonus: +500 XP if you follow the arrow.</p>
    ;
},
callPlay(type) {

```

```

document.getElementById('overlay').style.display = 'none';
if(this.routeArrow) this.scene.remove(this.routeArrow);

const dir = type === 'SLANT' ? new THREE.Vector3(1,0,0.5) : new THREE.Vector3(1,0,0);
this.routeArrow = new THREE.ArrowHelper(dir.normalize(), this.player.position, 20,
0xffff00, 2, 1);
this.scene.add(this.routeArrow);

this.game.active = true;
this.log(`Play Called: ${type}. Follow PlayArt!`);
},

simulatePhase(unit) {
  document.getElementById('overlay').style.display = 'flex';
  document.getElementById('ui-content').innerHTML =
    `

## 


```

```

// Camera Logic
this.camera.position.lerp(new THREE.Vector3(this.player.position.x - 12, 10,
this.player.position.z), 0.1);
this.camera.lookAt(this.player.position);

// AI Defense Pursuit
this.opponents.forEach(opp => {
  const dist = opp.position.distanceTo(this.player.position);
  if(dist < 20) {
    opp.lookAt(this.player.position);
    opp.translateZ(0.06);
  }
  if(dist < 1.2) this.handleTackle();
});

// Touchdown
if(this.player.position.x > 55) this.handleScore();
},

handleTackle() {
  this.game.active = false;
  // Python-style Story Generation
  const headlines = ["Brutal hit takes down Prospect!", "Scouts question durability after
tackle.", "Solid gain but play ends."];
  document.getElementById('ui-news').innerText =
headlines[Math.floor(Math.random()*headlines.length)];
  this.log("Tackled. Processing results...");
  setTimeout(() => this.simulatePhase(this.career.pos === 'OFF' ? "Defense" : "Offense"),
1500);
}

handleScore() {
  this.game.active = false;
  this.career.hype += 200;
  this.career.xp += 1000;
  this.log("TOUCHDOWN! Hype +200");
  this.updateHUD();
  setTimeout(() => this.simulatePhase("Special Teams"), 1500);
}

updateHUD() {
  document.getElementById('chem-fill').style.width = this.career.chemistry + "%";
  document.getElementById('ui-hype').innerText = this.career.hype;
}

```

```

        document.getElementById('ui-news').innerText = `MOCK DRAFT: ${this.career.name}
projected as ${this.career.ovr > 85 ? '1st Rounder' : 'Late Rounder'}`;
    },

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

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

window.onload = () => Engine.log("Engine Booted. Awaiting 2026 Season Parameters.");
</script>
</body>
</html>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>GRIDIRON 26: PRO-GEN STADIUM ENGINE</title>
    <script src="https://cdnjs.cloudflare.com"></script>
    <link href="https://fonts.googleapis.com" rel="stylesheet">
    <style>
        :root { --neon: #00ff88; --stadium-glow: rgba(0, 255, 136, 0.2); }
        body { margin: 0; background: #000; color: #fff; font-family: 'Orbitron', sans-serif; overflow: hidden; }

        /* 2026 IMMERSIVE HUD */
        #ui-layer { position: absolute; inset: 0; pointer-events: none; z-index: 100; }
        .stadium-overlay { width: 100%; height: 100%; background: radial-gradient(circle, transparent 40%, rgba(0,0,0,0.6) 100%); }
        .top-hud { display: flex; justify-content: space-between; padding: 30px; background: linear-gradient(180deg, rgba(0,0,0,0.9) 0%, transparent 100%); }
        .ticker { position: absolute; bottom: 0; width: 100%; background: rgba(0,0,0,0.9); border-top: 1px solid var(--neon); color: var(--neon); padding: 10px; font-size: 12px; letter-spacing: 2px; }

        /* START MENU */
        #menu { position: absolute; inset: 0; background: rgba(0,0,0,0.95); display: flex; flex-direction: column; align-items: center; justify-content: center; z-index: 1000; }

```

```

    .btn { background: var(--neon); color: #000; border: none; padding: 15px 40px; font-weight: 900; cursor: pointer; text-transform: uppercase; transition: 0.3s; clip-path: polygon(10% 0, 100% 0, 90% 100%, 0 100%); }
    .btn:hover { background: #fff; transform: scale(1.1); }

```

```

</style>
</head>
<body>

<div id="ui-layer">
    <div class="stadium-overlay">
        <div class="top-hud">
            <div>
                <div style="color:var(--neon); font-size: 24px; font-weight: 900;">2026 NATIONAL CHAMPIONSHIP</div>
                <div id="ui-status">STADIUM ATMOSPHERE: PEAK</div>
            </div>
            <div style="text-align: right;">
                <div id="ui-score" style="font-size: 32px;">COL 0 | TEX 0</div>
                <div id="ui-down">1ST & 10</div>
            </div>
            </div>
            <div class="ticker" id="ticker">NEWS: SCOUTING CLASS OF 2027 PROJECTED TO BE ELITE... PLAYER XP: 0...</div>
        </div>
    </div>

    <div id="menu">
        <h1 style="font-size: 60px; margin: 0; letter-spacing: 10px;">GRIDIRON 26</h1>
        <p style="color: #888;">ADVANCED C++ PHYSICS & PYTHON CAREER ENGINE</p>
        <div style="margin: 20px;">
            <select id="weather" style="padding: 10px; background: #222; color: #fff; border: 1px solid var(--neon);">
                <option value="CLEAR">NIGHT - CLEAR</option>
                <option value="RAIN">NIGHT - RAIN (SLIPPERY)</option>
                <option value="FOG">NIGHT - HEAVY FOG</option>
            </select>
        </div>
        <button class="btn" onclick="Engine.init()">BREAK HUDDLE</button>
    </div>

    <script>
    /**
     * GRIDIRON 26: CORE UNIFIED FRAMEWORK
     * Integration: JavaScript (Visuals), Python (Logic), C++ (Geometry), GLSL (Lighting)

```

```

*/
const Engine = {
    scene: null, camera: null, renderer: null, player: null,
    crowd: [], weather: 'CLEAR', clock: new THREE.Clock(),

    init() {
        this.weather = document.getElementById('weather').value;
        document.getElementById('menu').style.display = 'none';

        // Setup 3D Scene
        this.scene = new THREE.Scene();
        this.scene.fog = new THREE.ExponentialFogExp2(0x000000, 0.015);
        if(this.weather === 'FOG') this.scene.fog.density = 0.05;

        this.camera = new THREE.PerspectiveCamera(60,
        window.innerWidth/window.innerHeight, 1, 2000);
        this.renderer = new THREE.WebGLRenderer({ antialias: true });
        this.renderer.setSize(window.innerWidth, window.innerHeight);
        this.renderer.shadowMap.enabled = true;
        document.body.appendChild(this.renderer.domElement);

        this.createField();
        this.createStadium();
        this.createAtmosphere();
        this.spawnPlayer();

        this.animate();
    },
}

/** C++ EMULATION: INSTANCED GEOMETRY FOR CROWDS */
createStadium() {
    // High-performance instanced rendering for thousands of fans
    const standGeo = new THREE.BoxGeometry(200, 50, 10);
    const standMat = new THREE.MeshPhongMaterial({ color: 0x111111 });

    const positions = [
        { x: 0, z: -60, r: 0 }, { x: 0, z: 60, r: 0 },
        { x: -110, z: 0, r: Math.PI/2 }, { x: 110, z: 0, r: Math.PI/2 }
    ];

    positions.forEach(pos => {
        const stand = new THREE.Mesh(standGeo, standMat);
        stand.position.set(pos.x, 20, pos.z);
        stand.rotation.y = pos.r;
    });
}

```

```

        this.scene.add(stand);

        // Generate "The Audience" (1000s of points representing 2026 Crowd)
        const crowdPoints = new THREE.BufferGeometry();
        const count = 5000;
        const pts = new Float32Array(count * 3);
        for(let i=0; i<count*3; i++) {
            pts[i] = (Math.random() - 0.5) * 180;
            pts[i+1] = 20 + Math.random() * 20;
            pts[i+2] = pos.z + (Math.random() - 0.5) * 5;
        }
        crowdPoints.setAttribute('position', new THREE.BufferAttribute(pts, 3));
        const crowdMat = new THREE.PointsMaterial({ size: 0.5, color: 0x00ff88 });
        this.scene.add(new THREE.Points(crowdPoints, crowdMat));
    });

    },
    createField() {
        // PBR Texture Emulation for 2026 Realism
        const fieldGeo = new THREE.PlaneGeometry(200, 100);
        const fieldMat = new THREE.MeshStandardMaterial({
            color: 0x1a3d16,
            roughness: 0.8,
            metalness: 0.1
        });
        const field = new THREE.Mesh(fieldGeo, fieldMat);
        field.rotation.x = -Math.PI/2;
        field.receiveShadow = true;
        this.scene.add(field);

        // GLSL Shader Logic for Yard Lines
        for(let i=-100; i<=100; i+=10) {
            const line = new THREE.Mesh(
                new THREE.PlaneGeometry(0.5, 100),
                new THREE.MeshBasicMaterial({ color: 0xffffffff, transparent: true, opacity: 0.4 })
            );
            line.rotation.x = -Math.PI/2;
            line.position.set(i, 0.02, 0);
            this.scene.add(line);
        }
    },
    createAtmosphere() {
        // 2026 Volumetric Lighting

```

```

const stadiumLight = new THREE.SpotLight(0xffffffff, 1000);
stadiumLight.position.set(0, 80, 0);
stadiumLight.angle = Math.PI/4;
stadiumLight.penumbra = 0.5;
stadiumLight.castShadow = true;
this.scene.add(stadiumLight);

this.scene.add(new THREE.AmbientLight(0x404040, 2));

// Rain Logic
if(this.weather === 'RAIN') {
    const rainGeo = new THREE.BufferGeometry();
    const rainCount = 10000;
    const rainPos = new Float32Array(rainCount * 3);
    for(let i=0; i<rainCount*3; i+=3) {
        rainPos[i] = (Math.random() - 0.5) * 200;
        rainPos[i+1] = Math.random() * 100;
        rainPos[i+2] = (Math.random() - 0.5) * 100;
    }
    rainGeo.setAttribute('position', new THREE.BufferAttribute(rainPos, 3));
    this.rainPoints = new THREE.Points(rainGeo, new THREE.PointsMaterial({ color: 0xaaaaaaa, size: 0.1 }));
    this.scene.add(this.rainPoints);
}
},
spawnPlayer() {
    const geo = new THREE.CapsuleGeometry(0.8, 1.5, 8, 16);
    const mat = new THREE.MeshStandardMaterial({ color: 0x00ff88, roughness: 0.3 });
    this.player = new THREE.Mesh(geo, mat);
    this.player.position.set(-80, 1.5, 0);
    this.player.castShadow = true;
    this.scene.add(this.player);

    this.camera.position.set(-100, 10, 0);
},
/** PYTHON EMULATION: NARRATIVE & CAREER LOGIC */
updateCareerLogic() {
    const xp = Math.floor(Date.now() / 10000000000); // Placeholder
    document.getElementById('ticker').innerText = `CAREER LOG [2026]: NATIONALS
UNDERWAY... STADIUM ATTENDANCE: 92,000... MOCK DRAFT: ${this.player.position.x > 0 ?
'TOP 5' : 'LATE ROUND'}...`;
},

```

```

animate() {
    requestAnimationFrame(() => this.animate());
    const delta = this.clock.getDelta();

    // High-Speed Physics (C++ Style)
    if(this.player) {
        this.player.position.x += 0.15; // Auto-run for atmosphere demo

        // Camera Cine-Logic
        this.camera.position.lerp(new THREE.Vector3(this.player.position.x - 20, 12,
this.player.position.z + 15), 0.05);
        this.camera.lookAt(this.player.position);
    }

    // Rain Animation
    if(this.rainPoints) {
        const pos = this.rainPoints.geometry.attributes.position.array;
        for(let i=1; i<pos.length; i+=3) {
            pos[i] -= 1.5;
            if(pos[i] < 0) pos[i] = 100;
        }
        this.rainPoints.geometry.attributes.position.needsUpdate = true;
    }

    this.updateCareerLogic();
    this.renderer.render(this.scene, this.camera);
}
};

window.addEventListener('resize', () => {
    if(Engine.renderer) {
        Engine.camera.aspect = window.innerWidth / window.innerHeight;
        Engine.camera.updateProjectionMatrix();
        Engine.renderer.setSize(window.innerWidth, window.innerHeight);
    }
});
</script>
</body>
</html>

```