That sounds like an ambitious and engaging concept! A multiplayer hacking learning game with a custom object-oriented shell could be a fantastic way to teach programming and cybersecurity concepts.
Here's an overview, initial feature set, and development plan to get you started.

Game Overview: *Code Nexus*

The game, let's call it **Code Nexus**, is an **MMORPG (Massively Multiplayer Online Role-Playing Game)** focused on the competitive and cooperative development of software and hardware hacking modules.

- **Core Loop:** Players start with a minimal, low-power **Virtual Computer (VC)** and a basic shell interface. They complete missions to earn **Credits** (\mathbb{C}), which are spent on **upgrading VC hardware** (CPU, RAM, Network Bandwidth) and **purchasing/writing new software/scripting modules**. Progress unlocks more complex missions and deeper access to the game world's virtual network.
- **Learning Focus:** The core mechanism of the game is the custom scripting interface. Players must learn real programming concepts (object orientation, data types, control flow) to progress.
- **Genre:** Educational/Hacking Simulation/MMORPG.
- **Setting:** A near-future digital landscape where rival factions (corps, collectives, governments) vie for data and influence.

Core Features and Design

1. The Virtual Computer (VC) and Custom Interface

The **Virtual Computer (VC)** is the player's core asset and interface.

- **The Shell: The *Nexus CLI***
  - **Interface:** A custom **Command Line Interface (CLI)**, similar to **Bash** or **PowerShell** in look and feel.
  - **Core Concept (Object-Oriented Function):** Every action in the game—from running a scan to attacking a server—is a **function call** that operates on an **object**.
    - **Syntax Example:** [function] [target_object] [parameters]
    - scan network 192.168.1.0/24 (The scan function operates on the network object.)
    - attack target_server --module="Bruteforce_v1.2" (The attack function targets target_server using a module as a parameter.)
  - **Custom Scripting:** Players don't just use pre-built commands; they **write their own modules (scripts)** in a simplified, in-game, object-oriented language (e.g., *NexusScript*). These modules are essentially custom functions that chain built-in or acquired commands together to automate complex tasks, like an attack sequence or a defensive firewall check.
- **Upgradeable Hardware:** | Hardware Component | Function/Impact | Start Level | Max Level | | :--- | :--- | :--- | :--- | | **CPU Cores** | Determines **Script Execution Speed** and maximum **concurrent processes**. | 1 Core | 16+ Cores | | **RAM (Memory)** | Determines the complexity/size of modules you can load and run. | 1GB | 64GB+ | | **Network Card (NIC)** | Determines **Scan Speed**, **Attack Data Rate**, and **Ping Latency** to targets. | 10

Mbps | 1 Gbps+ |

2. Player Progression and Learning

- **Leveling System:** Players level up by successfully completing missions, writing complex, efficient code modules, and engaging in player-vs-player (PvP) activities.
  - **Low Level:** Access to basic I/O, variable declaration, simple control flow (if/else). Missions focus on information gathering and simple network traversal.
  - **Mid Level:** Unlocks **object creation/instantiation**, **function definition**, and access to advanced network protocols/objects (e.g., **TCP/IP stack objects**). Missions involve bypassing firewalls and early encryption.
  - **High Level:** Unlocks **multi-threading/parallel execution** (tied to CPU upgrade), **polymorphism/inheritance** for module design, and complex **cryptography objects**. Missions involve large-scale data breaches or defense of critical infrastructure.
- **The Learning Atmosphere:**
  - **Code Documentation Object:** A built-in object like help [command/module] that provides clear, simple explanations of the underlying programming or networking concept (e.g., explaining **polymorphism** when a player tries to upgrade a module).
  - **Tutorial Missions:** Carefully designed initial missions that force players to use core programming concepts like **loops** and **conditional statements** to progress.

3. Multiplayer Gameplay (PvP & PvE)

- **PvE (Player vs. Environment): Missions**
  - **Contract System:** Players accept contracts (missions) from various in-game organizations for Credits (\mathbb{C}).
  - **Mission Types: Data Retrieval**, **Server Sabotage**, **Security Patching** (for defense-minded players), and **Network Mapping**.
  - **Dynamic Targets:** Targets should be procedurally generated, virtual server networks with different OS "flavors," security settings, and digital assets.
- **PvP (Player vs. Player): Hacking Duels**
  - **Module Battles:** Players can "hack" each other's VCs. The battle is a duel of modules—one player's **Attack Module** vs. the other's **Defense Module (Firewall/IDS)**.
  - **Real-Time Scripting:** Players can monitor the battle output in the Nexus CLI and make on-the-fly adjustments or deploy counter-scripts. Victory is awarded to the player whose module is more efficient, less resource-intensive, or better coded to exploit a security hole (bug) in the opponent's defense module.
  - **Staking:** Players can stake Credits (\mathbb{C}) on duels.

Development Plan: Phase 1 (Minimum Viable Product - MVP)

Phase 1 focuses on building the core engine and the minimum necessary features to prove the concept and the learning loop.

Stage 1: Core Engine & Single-Player Shell

1. **Develop the *Nexus CLI* Engine:** Create the custom interpreter for the in-game scripting language (*NexusScript*).
   - Must support: Variables, Basic Math, Control Flow (if/else, for/while loops).
   - Define core **Object Types**: Network, Server, File, Module.
2. **Build the VC Interface:** A basic terminal-style interface for the player. Implement the three starting hardware components (CPU, RAM, NIC).
3. **Implement Basic Core Functions:** scan, connect, ls (list files), cat (view file contents), and run (execute a module).

Stage 2: Initial Progression and PvE

1. **Implement Hardware Upgrades:** Allow the player to spend a placeholder currency to upgrade the starting hardware.
2. **Design the Module Editor:** A simple in-game text editor where players write, save, and debug *NexusScript* modules.
3. **Create 5 Tutorial Missions:** Missions must guide the player to:
   - Use the scan and connect functions.
   - Write a module that uses a **loop** to try a list of passwords (Basic Brute-force).
   - Use an if/else statement to handle a successful vs. failed connection.
4. **Credit System:** Implement the Credits (\mathbb{C}) currency earned from missions.

Stage 3: Multiplayer Foundation

1. **Basic Server Infrastructure:** Set up a central server to manage player profiles and game state.
2. **Implement Player-to-Player PING:** The only allowed initial PvP interaction is the ability to run a latency check (ping [player_IP]).

MVP Goal:

A player can boot up their VC, write a basic module using loops and conditionals, successfully run it to complete a simple network-hacking mission, and upgrade their starting CPU using the earned credits.
Would you like to dive deeper into the design of the **NexusScript** language itself, or focus on expanding the mission and progression ideas? 💻