That's an excellent direction! Focusing on a complex yet simple, feature-rich design grounded in real-world cybersecurity processes will make **Code Nexus** a highly effective learning tool. Here is the expanded design, integrating real-world hacking methodology and a scalable mission structure.

1. The NexusShell Language & Tool Simulation

The core of the game is the **NexusShell** interface and its object-oriented custom language, **NexusScript**. This system is designed to simulate the core *function* of real-world tools without requiring players to master the full syntax complexity of languages like C++ or Python.

A. Core NexusScript Concepts

Real Concept	NexusScript Implementation	In-Game Command / Function	Learning Focus
Data Types	String, Integer, Float, Boolean, **IP_Object**, **Port_Object**	set target_ip = new IP_Object("10.0.0.1")	Variables and Object Instantiation
Control Flow	Loops and Conditionals	if (target_ip.is_open_port (80)) { run web_scan }	Logic and Automation
Functions/Modules	Custom scripts created by the player	func my_scan_module(IP) { ... }	Modular Programming/Code Reusability
Exploit Framework	The **Exploit_Object** Class	exploit_v1.inject(target_service)	Object Inheritance and Parameters

B. Simulated Hacking Tools

NexusScript will have built-in **system objects** that simulate the function of real-world tools. Players must learn the correct *sequence* and *parameters* (like in the real world of **penetration testing**).

Real-World Tool Category	NexusShell Function/Object	Real-World Analogues	Game Functionality
Reconnaissance	scan (for ports/OS), ping	**Nmap, Zenmap, OSINT tools**	Finds open ports, identifies services, determines target OS (which dictates usable exploits).
Traffic Sniffing	sniff_traffic	**Wireshark, Tcpdump**	Captures virtual network packets. Low-level players see hashes; high-level players get partial passwords/keys that require further decryption.
Exploitation	exploit (class), inject	**Metasploit Framework**	Loads a purchased or

Real-World Tool Category	NexusShell Function/Object	Real-World Analogues	Game Functionality
			custom **Exploit Module** and executes it against a vulnerable Port_Object or Service_Object.
Web App Testing	web_request, fuzz	**Burp Suite, OWASP ZAP**	Intercepts, modifies, and replays web traffic to test for vulnerabilities like **SQL Injection** or **Cross-Site Scripting (XSS)** (simulated through string manipulation).
Password Cracking	hashcrack	**John the Ripper, Hashcat**	Takes a captured hash (from a mission) and attempts to crack it based on a dictionary/ruleset (or custom player-written rule script).

2. Scalable Mission Design and Progression

Missions are structured around the phases of a real-world penetration test: **Reconnaissance, Scanning/Vulnerability Analysis, Exploitation, and Post-Exploitation (Pivoting/Maintaining Access).**

A. The New Player Tutorial (The LAN Confinement)

This phase serves as a mandatory, single-player tutorial teaching the absolute basics of the **NexusShell**.

1. **Phase 1: Minimal Resource (The Terminal)**
    - **Goal:** Access a file on the local machine.
    - **Action:** Player is limited to ls (list files) and cat (read file) commands. Teaches file system basics.
    - **Reward:** Unlocks the **IP_Object** and the **ping** command.
2. **Phase 2: Network Discovery (The LAN)**
    - **Goal:** Pwn/control 3 simple devices on the local network (e.g., a printer, a thermostat, a file server).
    - **Action:** Player must use **ping** and the basic **scan** command to find the devices. They use a single, hardcoded simple exploit function (e.g., exploit_default_pass) to gain root access.
    - **Reward:** Unlocks the **Module Editor** (scripting), the **for loop** function, and the ability to purchase **CPU upgrades** (improving module execution speed).
3. **Phase 3: The Gateway (The Firewall)**
    - **Goal:** Bypass a simple **Network Firewall Object** to gain access to the external

network (PvP and main PvE world).
    - **Action:** The player must write their first **NexusScript** module using the newly unlocked for loop to automate the attack or circumvent a simple rule-set, thus learning *why* automation is critical.
    - **Completion:** The player is now Level 1 and enters the main game world.

B. Mid-Game and End-Game Scaling

Missions are sourced through a dynamic **Contract Board**, where difficulty scales based on the player's highest completed tier and the complexity of their acquired/developed modules.

Mission Tier	Target Complexity	Vulnerability Focus	Scaling Mechanism
Tier 1 (Trainee)	Small business server, simple web page, IoT device.	**Default Credentials**, basic **Buffer Overflows** (simulated by oversized string inputs).	Requires simple **for loops** and the first **Exploit_Object** type.
Tier 2 (Operator)	Financial Kiosk, Mid-level Corporate Network (multiple subnets).	**Known CVEs** (simulated: patching systems leaves old versions with flaws), **SQL Injection**.	Requires multi-stage **NexusScripts** (Recon \rightarrow Exploit \rightarrow Pivot). Requires RAM upgrades for large scripts.
Tier 3 (Master)	Major Corporate Server, Government Systems, Rival Faction VC.	**Complex Privilege Escalation**, **Zero-Day Simulation**, **Custom Firewall/IDS modules**.	Requires **Object Inheritance** in modules, multiple concurrent processes (CPU), and high-bandwidth NIC to exfiltrate massive data volumes.

C. Integrating Real-World Exploits (CVE/Zero-Day)

To enhance the learning aspect, vulnerabilities should mirror real concepts:

- **CVE Simulation (Known Vulnerability):** A mission target is running a specific virtual service version (e.g., FTP_Server_v1.0). The player must find or purchase a corresponding **Exploit_Object** for a publicly known vulnerability (like a simulated **CVE**). This teaches the importance of **version control** and **patching**.
- **Zero-Day Simulation (Dynamic/Unpatched):** In high-tier missions, the target service is a **custom version** with a unique, hidden flaw (the "Zero-Day").
    - The player cannot buy a module. They must use the **fuzz** or **debug** function on the target to analyze its behavior.
    - This analysis generates a specific **Error Log Object** in the NexusShell.
    - The player must then analyze the error log and **write a completely new, custom NexusScript module** designed specifically to exploit that single flaw, simulating the creation of a Zero-Day exploit code.

This design ensures the game is simple to start (CLI basics), but quickly scales in complexity,

requiring mastery of programming concepts and the **simulated ethical hacking process** to succeed.