That's the heart of the system. Let's expand on the core **Logic and Main Functions** of the **Code Nexus** game, focusing on how the **Virtual Computer (VC)** processes commands, manages objects, and executes the player's **NexusScript** modules.

6. Logic and Main Function Design

The system runs on a simplified, layered logic model that simulates a real operating system and network stack, allowing for complex emergent gameplay while maintaining the easy-to-use NexusScript interface.

A. The NexusShell Command Parser

The first layer of logic is how the VC interprets what the player types.

1. **Input Parsing:** The shell uses spaces to divide the player's input into tokens: \text{Input} \rightarrow \text{Token 1 (Command/Object)} | \text{Token 2 (Target/Method)} | \text{Token 3+ (Parameters)}
    - **Example Input:** set $ip = new IP 10 0 0 1
    - **Parsed:** set (Command) | $ip (Variable) | = (Assignment) | new IP (Object Constructor) | 10 0 0 1 (Parameters)
2. **Function Dispatch:** The parser determines if the first token is a **Core Command** (ls, scan, set) or a **Variable/Object Method Call** ($router.connect).
    - **Core Command:** The VC's built-in kernel executes the command logic directly.
    - **Method Call:** The system looks up the object's class (IP, Router, Exploit) and executes the specific method (connect, scan, deploy) defined within that object.
3. **Error Handling (Logic Feedback):** The parser provides immediate, simple feedback for invalid syntax, teaching the player the correct format:
    - **Syntax Error:** "Error: Invalid parameter count for scan. Usage: scan [IP]"
    - **Object Error:** "Error: Method connect is not available on Object IP. Try a Service_Object."

B. Core Function: Object-Oriented Network Stack

The most critical function is how the VC simulates network interaction through the custom **Object Architecture**. Every interaction is a transaction between objects, which maps directly to real-world network layers.

NexusScript Object	Real-World Analog Analogy	Main Function Logic
IP / Network	Network Layer (Layer 3)	**Address Resolution:** The core function is to resolve the address and manage the virtual **latency** (simulated ping time) based on the player's NIC hardware.
Port / Service	Transport/Application Layer (Layer 4/7)	**State Management:** Tracks if the port is **open/closed/filtered** and manages the running **service version** (which determines which

NexusScript Object	Real-World Analog Analogy	Main Function Logic
		Exploit_Objects will work).
Exploit	Payload Execution	**Vulnerability Check & Code Injection:** When $exploit.deploy() is called, the system checks if the target Service_Object version matches the vulnerability defined in the $exploit module. If matched, the logic flips the target.access_level to root or user.
Firewall	Security Appliance	**Rule-Set Evaluation:** When an attack is initiated, the target's Firewall_Object checks the attacker's IP and **method** against its internal rules. If a rule is hit, the attack is aborted, or an **Intrusion Detection System (IDS)** alert is generated.

C. Main Function: The Module Execution Loop

When a player runs a script using run $module_name, the VC initiates the following loop, managing resources and checking for failure states:

1. **Resource Check:** The system verifies the script's complexity (size, length of loops) against the VC's current **RAM** limit. If exceeded, the script is immediately halted with an Out of Memory error.
2. **Execution:** The **NexusScript** commands are executed sequentially.
3. **CPU/Time Accounting:** Each command consumes a small amount of virtual time and a portion of the VC's **CPU Load**.
    - *Free players:* Have lower CPU thresholds, meaning a complex script will run slower and risk the VC overheating/detection.
    - *VIP players:* Have higher thresholds, allowing for faster and more complex execution.
4. **Error Handling & Learning:** If a logical error occurs during a mission (e.g., an exploit fails), the system returns a simple **ERROR_OBJECT** to the player's session.
    - The player must then use a specific command like target_log read error_object to get the simulated, educational reason for the failure (e.g., "Authentication failed: Invalid credentials provided to FTP service").
5. **Completion & Reward:** If the script successfully meets the mission's objective (e.g., flipping a remote server's access_level to root and exfiltrating the data), the loop ends, and the player is awarded **Credits (\mathbb{C})** and **Experience Points (XP)**.

This logical structure ensures that players learn the **Penetration Testing Methodology** (Recon \rightarrow Vulnerability \rightarrow Exploit \rightarrow Pivot) by requiring them to chain the correct sequence of object interactions.