I've compiled the discussed logic and game mechanics into a simple, multi-step set of starting instructions. This provides a roadmap for implementing the **Code Nexus** game. 💡

Phase 1: Core System Implementation (The Foundation)

This phase focuses on building the minimal viable environment required to run the game and validate the core mechanics.

Step 1: Design and Implement the Virtual Computer (VC) Kernel and NexusShell

1. **VC State Machine:** Define the core player data object (the VC) to track status: Player Level, \mathbb{C} (Credits), C-XP, VC Hardware Specs (RAM/CPU/NIC), and the core **Knowledge-Map (K-Map)** state.
2. **NexusShell Parser:** Implement the command parser logic. It must correctly tokenize input using **spaces** as delimiters and recognize variable usage with the **$** prefix.
3. **Core Command Execution:** Hard-code the immediate execution logic for the starting commands: ls, cat, ping, set, and run.

Step 2: Implement NexusScript Syntax and Module Editor

1. **Scripting Engine:** Create the interpreter for **NexusScript** functions (func, if, for). Use simple bracket syntax ({}) for blocks and space-separated arguments for functions (e.g., $obj.method(arg1 arg2)).
2. **Module Editor:** Implement the basic edit command interface with saving/loading functionality, allowing players to write and store their first simple scripts (modules).

Step 3: Implement Core Object Architecture and Boolean Logic

1. **Object Classes:** Define the initial classes for network objects: **IP_Object** and **Service_Object**.
2. **Server Logic:** Implement the core functions for scan and connect/login. When a player uses scan, the system must use Boolean logic to determine if the target's virtual firewall allows the action.
3. **Progression System:** Implement the C-XP system that awards points based on successful command execution and mission completion.

Phase 2: Freemium, Gating, and Acquisition

This phase integrates the monetization and knowledge-gating strategies.

Step 4: Implement Freemium Tiers and Resource Constraints

1. **F2P Constraint Logic:** Set initial low caps on F2P VC resources (CPU/RAM). Implement a time-delay on resource-intensive actions (e.g., hashcrack) to create the time-sink

constraint.
2. **Architect VIP Tier Logic:** Introduce the VIP subscription check. If active, lift the resource caps, remove time delays, and grant access to the **Trace Mode** for the run command.

Step 5: Design and Gate the Knowledge-Map (K-Map)

1. **Initial Lockout:** Ensure all advanced commands (pivot, thread spawn, raw) are initially set to **LOCKED** in the K-Map. The NexusShell must return the "Command not found" error if used.
2. **Fragment Discovery Logic:** Design the game files to contain easily parsable **Knowledge Fragments** (e.g., text containing "CMD_DECLARE: pivot"). Implement the VC kernel logic to scan copied files for these fragments.
3. **Synthesis and Validation:** Implement the Boolean Server Logic gates for advanced command unlocking. Require players to collect \ge 3 fragments and successfully run a **Synthesis Module** before a command's Man Page is available.

Phase 3: Dynamic Missions and PvP

This phase brings the system to life with dynamic content and competitive multiplayer.

Step 6: Create the Mission Generation and Reward System

1. **Target Generation:** Implement a script that procedurally generates PvE targets (IP, open ports, service versions, firewall rules) based on the target difficulty level.
2. **Reward Logic:** Integrate the reward system: Successful mission completion grants \mathbb{C} and a guaranteed, unrepeatable **Knowledge Fragment**.

Step 7: Implement PvP and Team Combat Logic

1. **Dynamic LAN Creation:** Design a system to spin up a temporary, isolated network instance for each PvP match, populating it with the VCs of all participating players.
2. **Acquisition Goal Logic:** Implement the PvP objective: Define a small set of files on defending players' VCs. Track the **Boolean Goal** for round victory: \text{WINNER} = (\text{ACQUIRED_FILES} \ge 3).
3. **Team Collaboration:** Implement the share command, which bypasses the Fragment Discovery logic and immediately updates a teammate's K-Map.

Step 8: Finalize Defense and Counter-Attack Mechanics

1. **Defense Integration:** Ensure players can code and deploy their custom defense modules and utilize the **vc.auto_defend** VIP command.
2. **Counter-Attack:** Define the logic for recovering stolen files, making the defense an active part of the winning goal.