I can certainly optimize those instructions. To effectively use an AI code platform like Jules, the instructions need to be **modular**, focus on **interface requirements**, and explicitly define the **custom language elements**.
Here are the optimized, structured instructions for an AI platform to build the starting framework for the **Code Nexus** game.

Optimized AI Framework Instructions (Code Nexus)

The following instructions are segmented for rapid, iterative development, focusing on the core functional components first.

Module 1: Core System and NexusShell (Foundation)

GOAL: Create the basic execution environment and CLI interface.

Component	Instruction	Output Requirement
Project Setup	Initialize a cross-platform (Python/JS preferred) project structure for a simulated CLI environment.	Base file structure, main execution loop (while True: get_input()), and command handler function (execute_command).
VC State	Define a Player class and a VC_State object to store: level, credits, ram_max, cpu_max, and a dictionary for **kmap_state** (all commands initialized to LOCKED).	Player class methods for update_xp and check_kmap.
NexusShell Parser	Implement the input parser. Use **space** as the primary token delimiter. Variable recognition must be strictly enforced via the **$** prefix.	The parse_input(text) function must return a list of tokens, identifying the command and its arguments.
Core Commands	Implement the simplest commands: ls, cat, and print. Use local file system I/O only.	Executable functions for cmd_ls(path) and cmd_cat(file).

Module 2: NexusScript Language & Objects

GOAL: Implement the custom scripting engine and object interaction.

Component	Instruction	Output Requirement
NexusScript Interpreter	Create a basic runtime environment capable of interpreting the custom **NexusScript** syntax: func, set $var = new Object(), if () {}, and for in {}.	A run_script(module_name) function that can execute a loaded script line by line, handling control flow logic.
IP_Object Class	Define the IP_Object class. Its constructor must take 4 integer parameters (e.g., 10 0 0 1).	The IP_Object class with properties and a ping() method.

Component	Instruction	Output Requirement
	Include a simple ping() method that returns TRUE or FALSE randomly for simulation.	
set Command	Implement the set command for variable assignment and object instantiation. It must correctly route "new IP..." to the IP_Object constructor.	The cmd_set(tokens) function handling both primitive values and new Object() calls.
Module Editor	Implement the edit $module_name function, allowing players to write and save multi-line **NexusScript** code to the VC's virtual file system.	A simple text input loop for the editor and a file-saving function.

Module 3: Gating and Freemium Constraints

GOAL: Integrate the core progression and monetization logic.

Component	Instruction	Output Requirement
Command Gating	Implement the **K-Map Check** inside the execute_command function. If a command is not found in the K-Map or is set to LOCKED, return the error "Command not found."	A central check_unlock(cmd_name) function that gates all advanced commands.
Fragment Logic	Create a scan_file_for_fragment(content) function. This function must search the input string for a hardcoded declaration tag (e.g., "CMD_DECLARE: PIVOT") and, if found, update the player's K-Map state for that command to DISCOVERED.	The scan_file_for_fragment function linked to the cat and copy commands.
F2P Constraints	Define two modes: F2P and VIP. In F2P mode, introduce a 5-second time.sleep() delay into a placeholder hashcrack function to simulate a time-sink.	A global boolean flag is_vip and conditional logic inside time-sensitive functions.
man Page Integration	Implement the man [command] help system. It should only display content if the command's state in the K-Map is at least **DISCOVERED**.	The cmd_man(command) function utilizing the pre-written command database content.