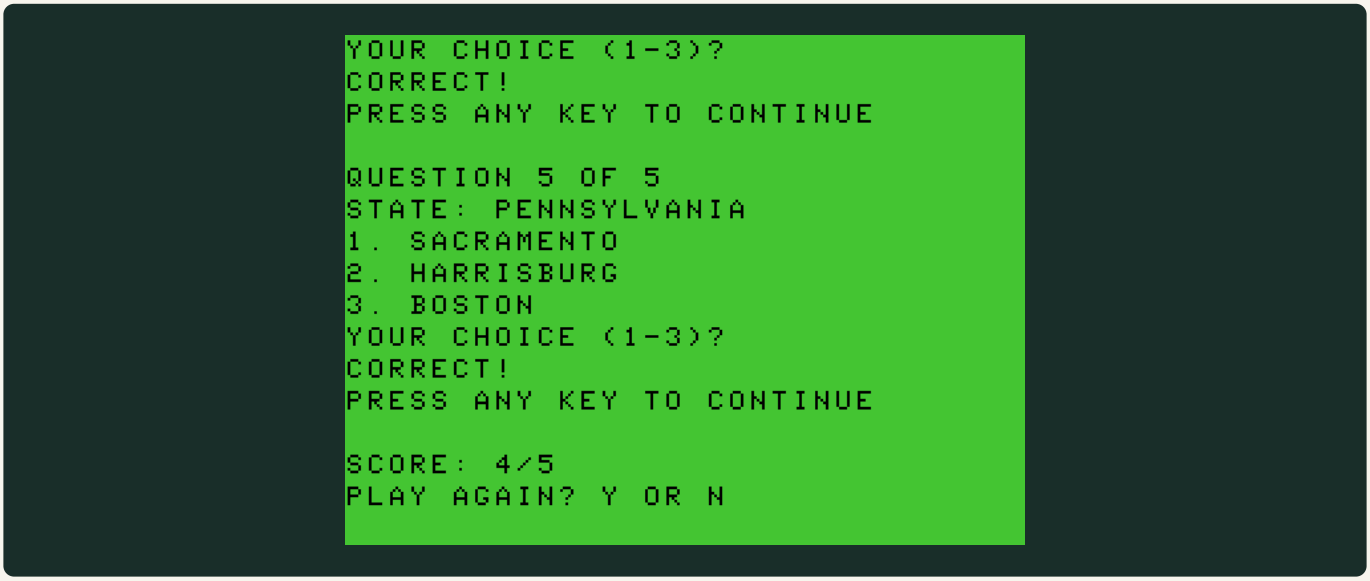
Vibe Coding with ChatGPT

A little Color Computer program can ask a surprisingly good question. Which planet would you like to visit? What is the capital of Kentucky? You can describe the game in ordinary words, ask ChatGPT to write a first draft, and watch a CoCo try it. The most interesting moment often comes when the CoCo does something neither of you expected.



Assembly quiz showing the correct answer to the fifth question, Pennsylvania, followed by SCORE 4/5 and PLAY AGAIN? Y OR N.

Brian K. Holman

Book 12 · VCC Plus 3.7.1

© Brian K. Holman 2026

Contents

Vibe Coding with ChatGPT 3
Make the first connection 4
From an idea to a working program 7
Give Mars a different greeting 10
Let the C compiler find the typo 12
Move a block one cell to the right 23
Build a quiz and give ChatGPT the keyboard 25
Find the first wrong result 33
Keep a field guide at hand 36
Start a project you can hand off 41
Publication and artwork notices 46

Vibe Coding with ChatGPT

In this book, *vibe coding* means making a program through conversation. You decide what the program should do. ChatGPT can read a draft, suggest a change and help investigate the result. VCC Plus gives you a place to see exactly what would change, decide whether to keep it and run it on the emulated machine. By the end you'll have a way to answer a very practical question. Did the program you kept actually do what you asked?

Set a place at the desk

You'll need VCC Plus for Mac with the build 30 interactive controls, the ChatGPT Mac desktop app signed in with access to **Work** and local MCP servers, and a folder for practice files. The lessons use a CoCo 3 running Disk BASIC. The C and assembly workspaces use the tools supplied with VCC Plus. Start with a disposable program, a few minutes at the keyboard and permission to share its contents with ChatGPT.

If you've never written a program, start with the first three chapters. One sentence changes a greeting, and you can try both answers. If you already program on a CoCo, make the first connection, then pick the chapter for your language. The capitals workshop brings the three paths back together.

The capitals workshop includes the complete example sources and a CoCo disk image. You can keep a copy to compare with your own first draft. In Chrome on Mac, Option-click a source link to save the file if a regular click opens it as text. Outcomes identified as **recorded** come from running the examples in VCC Plus. Prompts are invitations to try a task yourself. The words and code you get on your Mac can differ.

Other books on the shelf

Choose **Help → Manual Library...** in VCC Plus. Programming in CoCo BASIC on the Color Computer 3 is a good companion for numbered listings. Programming in C on the CoCo 3 — CMOC and Its APIs covers the C workspace and the bundled libraries. Your First 6809 Programs starts with registers and memory. Inside the CoCo 3 — Hardware and Debugging follows a running program more closely. You can also ask ChatGPT to find a section of an installed book during a lesson.

For historical magazines and manuals, the Archive Library has its own search.

Starting a larger project? Start a project you can hand off supplies an editable setup prompt and a downloadable folder of project standards, connection checks, test records and decision templates. The original CMOC and LWTools references cover the bundled compiler, assembler and linker alongside this library's teaching books.

Make the first connection

Chapter 1 of 9

On your Mac, two windows will share the work. The VCC Plus editor holds your program. The ChatGPT window is where you describe what you'd like to make. A small helper passes requests between them using the Model Context Protocol, usually called **MCP**. Let's give the helper one page to read and see whether it can find a message you left there.

Put a message in a practice draft

1. Copy **VCC Plus** from its disk image into **Applications**. Eject the image and open the installed app. Open the signed-in **ChatGPT** desktop app on the same Mac.
2. In VCC Plus, choose **Develop → Connect to ChatGPT...**. It opens the main BASIC draft and its connection panel. Look at which draft the panel names. If it contains work you care about, leave it alone. Open a separate practice draft in **BASIC Development** using **New**. A new editor can begin with the Earth or Mars starter.
3. In the practice editor, add a numbered remark near the top. For example, type `5 REM BLUE LANTERN` on a line by itself. **Make up your own two words** so the answer is something only this editor can supply. Keep the remark as the lowest numbered line. Save this practice listing as `Lantern.bas` in your own folder.
4. Open **Local Assistant Access...** from that editor's **...** menu. Verify the document named in the panel is the one you just prepared. Each BASIC connection shares a particular draft.

A BASIC remark is a note to a reader. You can leave one in the listing without changing how the game plays. You won't need to start the emulated CoCo for this first check.

Give ChatGPT a page to read

Choose **Save Read-Only Connection...**. Save the connection file in a private folder you'll recognize. It grants access to the selected document. Keep it out of the conversation and out of any folder you plan to share with your project.

Choose **Add to ChatGPT** in VCC Plus. Open a **new chat** in the ChatGPT Mac app and choose **Work** to use the local computer. Type `/mcp` if you'd like to see the configured servers. Seeing VCC Plus there is a good first sign. Now try a request that needs the actual editor.

> Use the VCC Plus BASIC connection. Read the current shared draft. Tell me its title and quote the lowest numbered line exactly.

Compare the reply with the line you typed. Did it find both of your words? If you chose `BLUE LANTERN`, for example, the answer should include `5 REM BLUE LANTERN`. You can also use **Copy First Prompt** in VCC Plus for its built-in connection check. Leave the shared editor and VCC Plus open while you work.

For a new project, **Copy Project Setup Prompt** provides a longer, editable starter with connection-setup permission, reference priorities and working rules. Choose **Edit...** to review it first. Copying changes only the clipboard; review the text before sending it to your assistant. See the project-starting chapter for the starter documents and the separate project, manual and Archive checks.

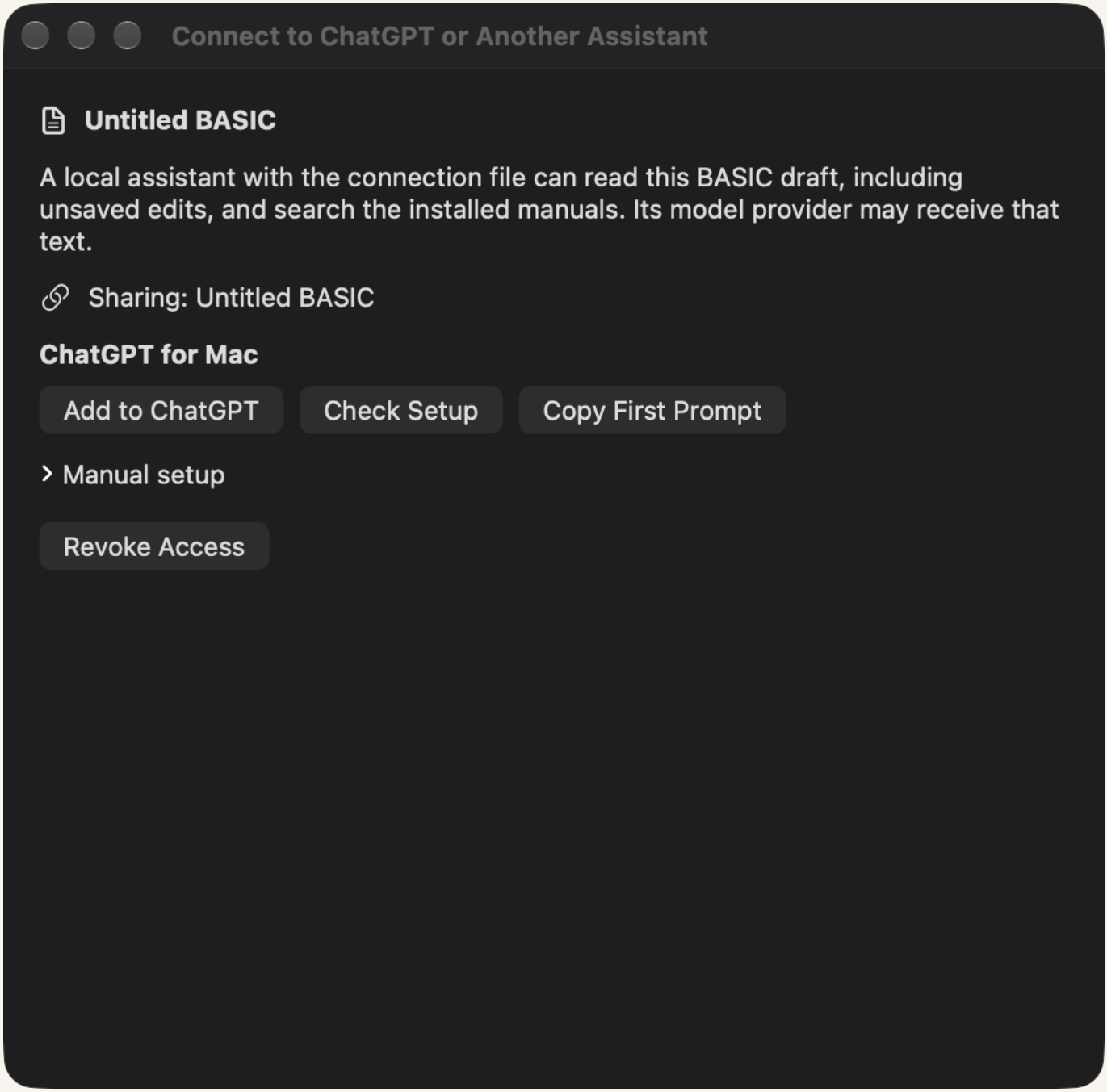


Figure 1: The BASIC connection panel identifies the shared draft above Add to ChatGPT, Check Setup and Copy First Prompt. Here the practice draft is named Untitled BASIC.

The connection to VCC Plus runs on your Mac. When ChatGPT reads a listing, diagnostic or screen through it, that text can go to the model provider and appear in the chat. A small practice project is a good first choice.

If the message doesn't arrive

- **ChatGPT guessed or gave you instructions for opening the file.** Make sure you're in **Work** in a new chat on the Mac. Ask it to use the VCC Plus tool to read the open draft and report the tool's response if it fails. Leave your secret words out of the prompt.

- **It quoted a different program.** Compare the named draft with the one in the connection panel. Open the right editor's **Local Assistant Access…**, save its connection, and add it. Name the intended workspace when more than one is available.
- **VCC Plus is missing from the list.** Use **Check Setup** in VCC Plus. If necessary, expand **Manual setup**. In a ChatGPT version offering **Settings → MCP servers**, choose **Add server → STDIO**. Copy the displayed name, command and two arguments into their respective fields. Each argument gets its own entry. Save, restart that server and repeat the reading test. This walkthrough uses a ChatGPT account and workspace that offer local servers on the Mac. OpenAI's desktop MCP instructions

Let the assistant propose an edit

Return to the practice draft's connection panel. Select **Allow proposed edits and controls for runs I start** and choose **Save Review and Run Connection…**. Use the updated connection in ChatGPT. If you replace the file used by an existing server, restart that server to make it read the new grant. Repeat the lowest-line check. For interactive lessons, save a fresh connection from the installed app and check its advertised controls.

You still have two decisions to make in VCC Plus. First you can accept or cancel the source change. Then you can start a shared run. Once a supported run starts, ChatGPT can read its text screen and send supported keys to that run. The C workspace also has a separate, explicit permission for disposable playtests; that grant permits repeated artifact-bound test launches without another launch prompt.

Use a separate connection file for each workspace you want to keep available. The equivalent panel is **Assistant…** in C Development and **Local Assistant Access…** in Assembly Development. All three workspaces use the same helper. A BASIC connection points to one draft. A C or assembly connection points to one project.

When you finish, choose **Revoke Access** in the workspace's panel to disable that grant and remove its file. Previously shared results can still appear in the chat. To work together again, save a new connection, reconnect and repeat the reading check.

An existing local Codex chat

The installed helper also works with a local Codex chat. Add its STDIO command and the two separate grant arguments in **Settings → MCP servers**, restart the server, then use `/mcp` in the existing project chat to check live discovery. Keep VCC Plus and the shared C project open. Use the private connection exported for that project, and keep it outside Git. Official OpenAI MCP setup

For a custom Make project, select **Use Project Makefile** in the C project's Assistant panel before exporting. The C workflow explains the build-support folders and configuration choices.

Check discovery separately from access

A saved server, tools visible in this chat, and a successful call are three different checks. Ask the assistant to discover its tools, call `get_capabilities`, and read the actual shared title and revision. For an updated C connection, also ask which capabilities it advertises: project refresh, diagnostics, disposable playtests, audio and processor selection. Installing a new app does not silently expand an older private grant.

If the server is registered but its tools are missing, use the client's supported **Restart** action and check its catalog again. A delayed catalog is not a reason to grant more files or keep exporting new connections. If the reply instead says `host_unavailable`, leave VCC Plus and the intended project open and check that the saved grant is still valid. That failure is different from a missing catalog.

For a technical connection check, the same installed helper and approved private grant support ordinary STDIO MCP: `initialize`, `notifications/initialized`, `tools/list`, then an advertised `tools/call`. Ask the assistant to report that result separately from tools visible in the chat. It must keep grant contents private and use the public helper; reading app caches or calling a private emulator bridge is not part of this workflow.

From an idea to a working program

Chapter 2 of 9

Suppose you'd like a scorekeeper for a game at your kitchen table. “Make me a scorekeeper” leaves ChatGPT to invent the rules. Pick two keys and write down what you expect them to do. Now you have both a request and a way to judge the first answer.

Describe a turn you could play

In a separate practice BASIC draft with review access, try the request below. Starting a practice draft and connecting it are covered in Chapter 1. You can also use this as a pattern when you design the capitals quiz.

> Make a small CoCo 3 Disk BASIC scorekeeper in this draft. Start at zero. Show which key to press. The 1 key adds one point. The 2 key prints FINAL SCORE followed by the total, then ends. Other keys leave the total alone. Read one key at a time. Propose the numbered listing in VCC Plus and wait for my review.

Prediction. If you send `1`, `1`, `0`, then `2`, the final score should be 2. Jot down that prediction before you read the proposed solution. It gives you a test that stays the same even if the program changes.

If you want to understand the keyboard routine, ask for a reference while you work.

> Search the installed VCC Plus manuals for INKEY$. Read a matching BASIC section. Give me the book and section title and explain what the program gets when nobody is pressing a key.

All three workspaces can search the installed manuals. ChatGPT can use `search_help` to find a section and `read_topic` to read it. You can follow the answer in **Help → Manual Library...** with **Ask the Manuals**. For this program, an empty string means no key is waiting. Polling again keeps the scorekeeper ready for your next turn.

VCC Plus also has a separate read-only MCP connection for historical publications. Open **Help → Archive Library...**, select the link button, choose **Save Archive Search Connection...**, then **Add to ChatGPT**. In a new Work chat, try “Search the Color Computer Archive for a BASIC manual page about INKEY$. Give me the manual title, the physical PDF page and the original Archive link so I can check it.” You can open that page in the Archive Library and compare its explanation with the behavior of your scorekeeper.

If the reader offers **Copy Snippet** or **Open in BASIC Editor**, it has a reviewed listing associated with that page. Keep its citation when discussing or adapting the example. Opening a BASIC listing creates a separate editable copy; inspect it before choosing Run. A photographed program or an OCR search hit does not by itself promise corrected, runnable source. The handbook's Archive reader walkthrough explains the page, citation and listing controls. The Archive connection remains read-only; use your separate development connection to propose project edits.

When you are researching software already in Software Library, look for **Guides & References** in its details. A reviewed walkthrough, map or tutorial opens directly at its Archive document/page. These links supplement the main manual; they do not assert that every title has been researched exhaustively. Keep the historical citation when it informs your project. Both libraries resume their browsing/reading contexts, and the Manual Library's **Bookmarks** list can keep useful native-book and toolchain-reference sections together. Those personal reading choices do not add MCP permissions or expose your private library.

Read the change before you play

Look at the proposal in **VCC Plus**. For a whole BASIC listing, check the first and last lines and then follow the short path that matters. Where does the score start? Which line adds one? What takes the player back to the key check after an invalid choice? Ask ChatGPT to point out the numbered lines if you'd like help reading them.
Choose **Apply Edit** when the proposal matches the game. A code block posted in the chat still needs to be proposed through the connection before you can review it in the app.
For a C or assembly project, you can make the same request in smaller pieces. Tell ChatGPT which existing files to change. Read the proposed changes together, apply them, then ask it to build the new revision. The compiler or assembler can catch a misspelling before you even start the CoCo. A successful build is your cue to try the game.

Start the CoCo and watch one turn

Ask ChatGPT, "Check the machine status for this draft. Tell me whether I can run it and name any blocker you find." Use a disposable Disk BASIC session. Save anything you want to keep before replacing a guest program. If the app reports writable media, protect or eject the disk after saving your work. Eject the starter cassette, finish any recording or Mac shell session and turn off DriveWire networking if the readiness check calls for it. The CoCo should be running at its supported BASIC prompt. Chapter 8 explains how to read a refusal.
Choose **Run** for the applied BASIC proposal in VCC Plus. For C and assembly, start the current build from the shared project as described in Chapters 4 and 5. Back in ChatGPT, ask for this test.

> Use the run I just started. Read its CoCo text screen. Send 1, 1, 0, then 2 through the VCC Plus game controls, one key per call. After each key wait for the program to accept it before continuing. Quote the final score from the CoCo. Tell me whether it agrees with my prediction of 2.

A choice game is easiest to test when it shows that it heard each key. Perhaps you see the updated total after a point and a short reminder after an invalid choice. If a screen stays the same, ask what the assistant actually observed. Avoid repeating a key until you know what happened. You can request visible feedback as a refinement.

Change one rule and keep the evidence

Suppose the score is right but your first-time player doesn't know how to quit. Try, "Add a line at startup that says PRESS 2 TO FINISH. Keep the rules and the scoring the same. Propose the smallest change that does that." Read the change in VCC Plus and run the same four keys again. Test an immediate 2 in a separate run as well. The total should be zero.
Save the source you want to keep from the native editor or project. Keep a short text note beside it. This pattern works for a ten-line game as well as a larger project.

Write down	What to put there
First request	The words you gave the model, the language and the target CoCo.
First attempt	Whether the first proposal was accepted, whether it built if needed, and what appeared on the first run.
A failed check	The exact input and the observed screen line or compiler diagnostic.
Refinement	What you asked for next, what changed, and which version you tested.
Working copy	The saved filename, target processor where relevant, and the last test someone else can repeat.

If the first run failed, keep that entry even after a repair succeeds. It answers a different question from whether the final game works. For a worked example with three first drafts and three recorded playtests, turn to the capitals quiz.

Give Mars a different greeting

Chapter 3 of 9

The CoCo asks you to choose Earth or Mars. Earth gets a blue greeting. Mars gets a red one. This time you'd like a visitor to Mars to see **Bring a warm coat!** You can describe that change to ChatGPT in a sentence. The interesting question is how you'll know the sentence made it all the way to the emulated CoCo.

Find the greeting

1. Open **Develop → BASIC Development**. If you have a draft you want to keep, save it. Choose **New** to open the Earth or Mars starter. In VCC Plus, the BASIC editor holds the listing on your Mac. The CoCo has its own program memory.
2. In this editor's **…** menu, choose **Local Assistant Access…**. Select **Allow proposed edits and controls for runs I start**, then **Save Review and Run Connection…**. Use **Add to ChatGPT** and open a new **Work** chat in ChatGPT for Mac. If you already connected this exact draft with review access, you can keep using that chat.
3. Ask ChatGPT to read the listing through VCC Plus. Give it a question whose answer you can check in the editor.

> Find the numbered line that sets the Mars greeting. Quote that line from my shared BASIC draft. What does each side of the colon do?

You should find the following line in both places.

```
90 PALETTE 6,37:M$="Hello Cold Red Dot World!"
```

BASIC executes two statements on line 90. The first assigns color 37 to palette entry 6. The second puts a string into the variable `M$`. A later `PRINT` displays the string. Changing the words inside the quotes lets you leave the palette and the keyboard routine alone.

Review one line

> Propose a one-line edit through VCC Plus. On line 90 change the Mars greeting to exactly "Bring a warm coat!" Keep the rest of the listing as it is.

Look in the VCC Plus review window. If ChatGPT only displays code in the conversation, ask it to submit the edit to the shared draft for review. Check the line number, the palette values, the quotation marks and the exclamation mark. Here is the line to look for.

```
90 PALETTE 6,37:M$="Bring a warm coat!"
```

Choose **Apply Edit**. The new line now appears in the Mac editor. You can still choose **Undo Edit** at this revision. If the preview changes anything else, choose **Cancel Edit** and ask for the single line again. If you typed in the editor while the review was open, have ChatGPT reread the current draft and submit a fresh proposal.

Try the two planets

Use a running Disk BASIC system at its **OK** prompt. Save any guest program you want to keep before replacing it. In the review window, choose **Run Both Choices**. This button recognizes the starter with the exact revised line above. For a different listing, use **Run** and test the choices yourself or ask the connected assistant to press the keys.

Watch the results for each planet. VCC Plus tries **3** first and checks that the menu stays up. It then tries **2** and reads **Bring a warm coat!** from the guest. On a fresh run it tries **1** and reads **Hello Nice Blue Dot World!** The check samples the background color of each greeting. It also types `PRINT 7` after each run and looks for `7` followed by `OK`. That last exchange tells you BASIC is accepting commands again.

> Read the verification results from VCC Plus. What happened after key 3? Give the greeting and the observed background for each valid key.

Use **RGB** output for the starter's blue and red. On composite output, the Mars palette value can appear amber. If a run is blocked, ask which condition VCC Plus reports. A mounted cassette, a writable disk or active DriveWire can prevent a shared run. Use the native media controls to resolve the reported condition after saving any changes you want to keep.

Start with an empty listing

Save your planet program. Choose **Revoke Access** for that connection. Choose **New**, click in the source editor, select all its text and delete it. Share this empty draft with review access and give ChatGPT the next job.

> Propose a complete numbered BASIC listing for this empty draft. Print the squares of the whole numbers 1 through 4, one result per line, then end. Use VCC Plus to submit it for review.

Read the proposed loop before choosing **Apply Edit**. Which line repeats, and what ends the loop? Choose **Run**, then ask ChatGPT to read back the screen. Look for `1`, `4`, `9` and `16` in order. Change the request to stop at 5, predict the last result, and check it against a second run. Save the listing you want to keep.

Let the C compiler find the typo

Chapter 4 of 9

One extra letter can stop a whole C project from building. You'll put that letter in a practice copy of **Hello Disk**, ask ChatGPT to get the compiler's diagnosis and review its repair. When the build succeeds, the CoCo will read five words from a floppy. That gives you something to check beyond the message that says the compiler finished.

Give the compiler a small problem

1. Choose **Develop → C Development...**. Click **New**, choose **Hello Disk**, and create a practice project in a new folder. Open `main.c` and look near the end of `main` for the line that prints the generated word count.
2. Add a lowercase **s** to the end of the variable name. In the supplied project that's line 14. Leave the printed message unchanged. Check your altered statement against the one below.

```
    printf("GENERATED WORD COUNT: %u\n", generatedWordCounts);
```

3. Click **Assistant...**. Expand **Files shared**. Alongside `main.c` you'll see the header `include/hello.h`, the C files in `src`, and `assets/words.txt`. Choose **Save Private Connection...**, give the C connection its own filename, then use **Add to ChatGPT**. Open a new Work chat and use **Copy First Prompt** to confirm the project. The compiler can work with your open draft, including the typo.

> Build the current revision of my shared Hello Disk project. Report the first compiler diagnostic with its file, line and exact message. Read the relevant project header, then propose the smallest repair through VCC Plus.

CMOC reports `undeclared identifier 'generatedWordCounts'` in `main.c`. An *identifier* is a name in the source. Compare the diagnostic with the declaration in the header.

```
extern const unsigned char generatedWordCount;
```

C treats the singular and plural spellings as different names. The declaration tells the compiler that the project supplies an unsigned byte named `generatedWordCount`. Removing the final **s** makes the statement refer to that byte. If the initial build succeeds, compare the variable in your editor with the altered line above and ask ChatGPT which project and revision it built.

Apply, rebuild and read the disk

Read the current and proposed lines in the VCC Plus review window. Choose **Apply Repair** when the proposed change removes the extra letter. Look for the **applied revision**. A revision identifies the particular draft the assistant built or edited. The failed build belongs to the earlier revision. Click **Rebuild** to compile the applied revision and watch for the successful result.

Before starting the program, leave a running Disk BASIC system at an empty **OK** prompt. Hello Disk needs **FD0** empty so VCC Plus can temporarily mount its generated disk there. Save any disk changes before ejecting it. If a BASIC listing is still in guest memory from the previous lesson, save any open BASIC editor draft, use **Read from CoCo** and **Save As...** to keep the guest listing. When you are ready to clear its program and variables, type `NEW` at the CoCo prompt.

Choose **Run and Verify**. VCC Plus matches an instruction in the running binary to the source, continues execution, checks for a floppy read and shows the program's output. Look for **HELLO**, **WORLD**, **CMOC**, **VCC** and

PLUS, followed by **DISK RESULT: 0** and **GENERATED WORD COUNT: 5**. Compare the words with `assets/words.txt`. Save your repaired project if you want the corrected sources on disk.

Play an ordinary C program

For a program that takes keyboard input, create a separate project using **New → Hello World**. This starter offers Earth or Mars in a single C file. Share the new project from **Assistant…** and ask ChatGPT to build its current revision. An ordinary shared build compiles the project's C files with bundled CMOC. For a project with its own Make rules, use the connection described below. An ordinary shared build can run with a write-protected disk in FD0.

Choose **Run C Build…**. **Play in CoCo** continues from the first verified statement in `main` and lets the connected assistant read the screen and send game keys. **Inspect at main** pauses at that statement so you can ask which source line and machine address were reached. Choose **Play in CoCo** for this exercise and give ChatGPT the keyboard task below.

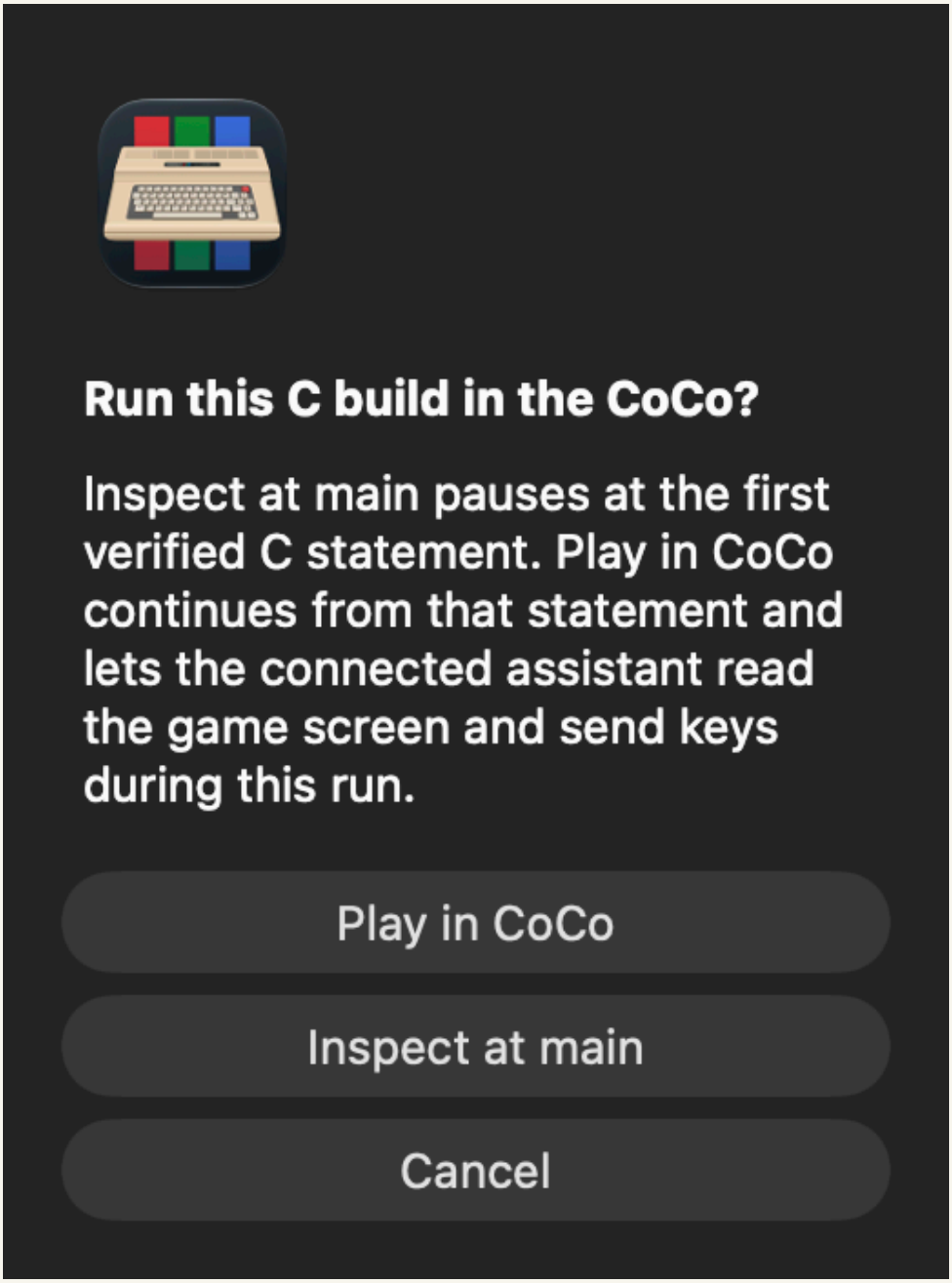


Figure 2: The Run C Build dialog offers Play in CoCo for the interactive game and Inspect at main for a debugger stop.

> Read the menu from this C run. Send the key for Mars, wait for the program to return to BASIC, and report the greeting from the final text screen.

Look for **Hello Cold Red Dot World!** The previous chapter changed the BASIC draft. This greeting comes from the C program you just built. A finished run can report the text captured at return. To choose Earth, start a fresh run.

Keep a project's Make rules

A larger game may combine C, inline assembly, generated graphics and a custom linker layout. Open its project folder in C Development. In **Assistant...**, select **Use Project Makefile**, check **Files shared**, and save a fresh

private connection. This permits the local host commands in that Make recipe and its compiler inputs. The native **Build** button and the assistant then use the project's declared recipe with the installed bundled toolchain. Its headers and inline assembly can stay in place.

Leave the checkbox off for the ordinary restricted C connection used earlier in this chapter. That builds ordinary sources with an app-generated recipe. Hello Disk also has its own installed recipe and verification flow. Choose the mode that matches the project you intend to share.

Suppose your repository contains `game/Makefile` and a sibling `tools` folder of packers. Open `game` as the project. In its existing `vccplus-project.json`, add `"buildSupportDirectories": ["../tools"]`, or enter `../tools` in Project Settings' **Build-support folders** field. The build snapshot copies that explicitly named folder beside the project so the recipe's `../tools` paths still work. Inspect the listed support folders before saving the connection.

> Identify this project's current revision, approved Make recipe, build actions and configurations. Build the current revision using its declared disk action and Release configuration, if those choices are available. Report the build ID, result and first diagnostic. Do not start the CoCo yet.

The assistant reads these choices through `get_capabilities` and requests the declared build or disk action through `build_project`. It can choose a listed configuration; it cannot invent a Make target or send extra command arguments. Source edits advance the revision. Assistant source-edit proposals do not change Makefiles, scripts or project configuration. Make those changes locally, then use the scoped refresh below and review the changed build controls. An older connection without refresh needs a fresh export instead.

A successful disk build does not establish that a packed game has a verified C run or debugger stop. A custom project can generate a banked debug manifest that binds its packed binary and disk to original source, linker, object and listing evidence. Its source set includes the complete retained compiler output, so large type records can be recovered from hash-verified assembly when the assembler listing truncates a long comment. VCC Plus checks the unpacked module bytes and actual physical RAM banks before showing source locations or values. Expected bank switches retain that connection. Optimized locals and unknown frames remain unavailable. A successful recipe is not proof of a debugger stop or of the game's behavior.

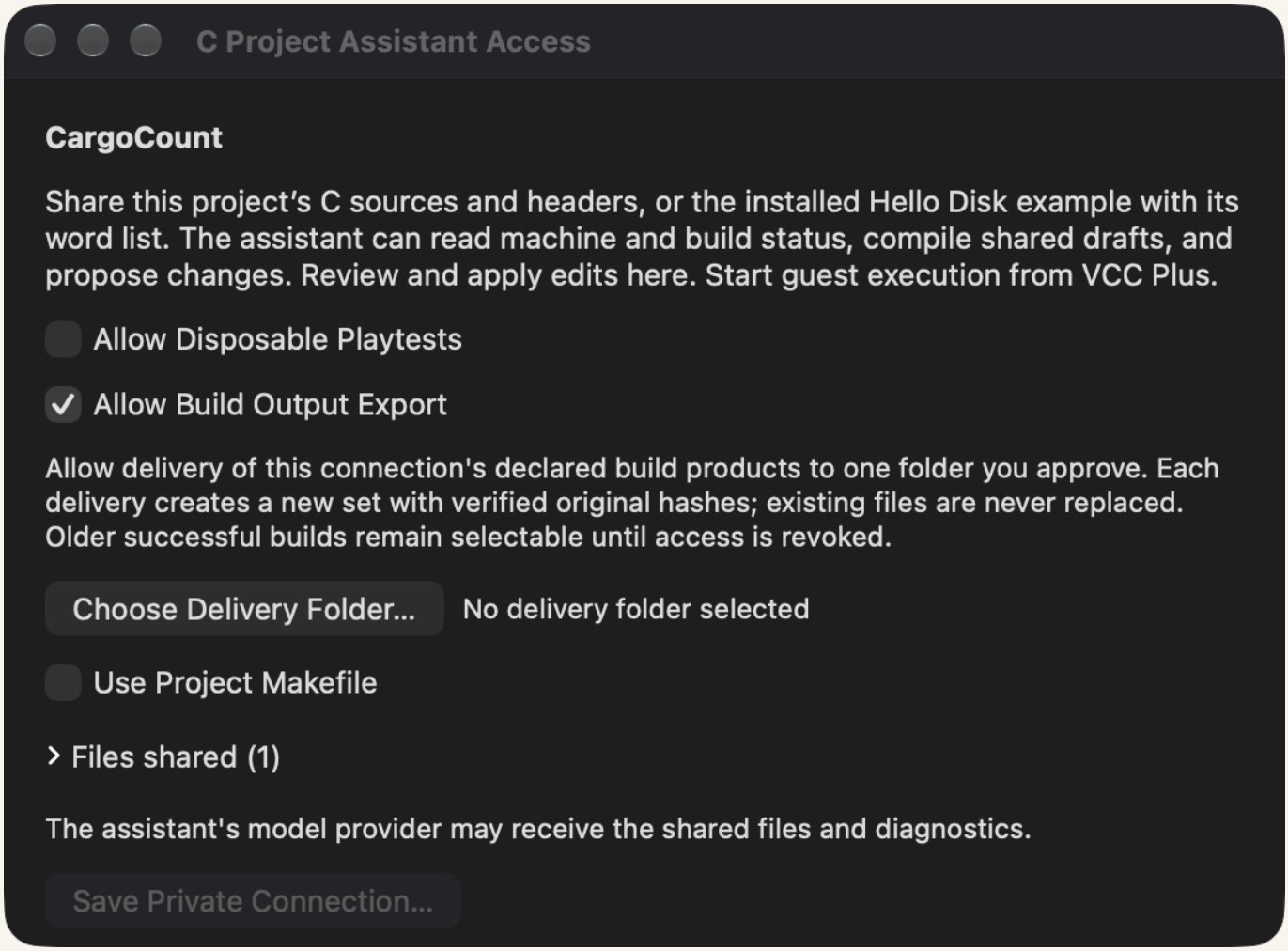


Figure 3: C Project Assistant Access offers separate choices for disposable playtests, build-output export and the project Makefile. Export is selected here, but no delivery folder has been approved, so Save Private Connection remains unavailable. Review the files and permissions before saving.

Deliver the build you already tested

A successful build has its own identity and frozen products. To let an assistant deliver those files, open **Assistant...**, select **Allow Build Output Export**, and use **Choose Delivery Folder...** before saving a fresh private connection. This permission is separate from permission to run disposable playtests. The assistant receives an opaque destination ID rather than access to browse Mac folders. Older connections keep their existing permissions.

> Build the normal Release and Debug configurations. Keep both build receipts. Export each configuration's declared binary, Home disk, Data disk and Home alias, if this project declares them, to the approved delivery folder. Do not build again to retrieve files. Independently compare every delivered length and SHA-256 with the original build receipt, then give me the two delivery sets.

`get_build` lists `exportable_outputs`. The assistant selects those output IDs and the exact original project/build/artifact/revision with `export_build_outputs`, then polls `get_build_export` until completion. Building Debug does not erase the previous Release choice. Each delivery gets its own verified folder, so identical filenames do not silently replace another configuration's files. An identical request retry verifies the prior delivery; a changed request or changed delivered file is reported as a conflict.

The binary, primary disk, existing debug manifest and additional playtest disks are declared products. Use `exportOutputs` in project settings for other concrete relative deliverables or aliases; an output directory is not a permission to export everything found there. A delivery preserves frozen build bytes. It does not include a playtest's later saves or publish software to the website. **Build → Show Build Products** remains available when you want the native route.

Refresh the inputs you meant to share

Connections that advertise `project_refresh` can inspect the approved project inputs and request a scoped refresh after an external edit. Ordinary edits to shared files update the revision. New shared files or changed build controls open a native review showing the additions and exact before/after contents. You can approve or reject; neither choice runs the project. Your unsaved drafts stay intact, and conflicting disk edits still need local resolution. Saving screenshots or receipts in a declared output folder does not change the source revision.

> Use read_project_inputs to list this project's included files and excluded output folders. I added a header locally. Request refresh_project at the current revision and report the review ID. Poll get_project_refresh for my decision, then identify the new revision before building. Keep unsaved drafts.

The manifest explains each input's role, inclusion and hash; it is paged, so ask for later pages when needed. New files, changed recipes and support scope need local review. Pending or rejected changes block builds and runs. Approval adopts the reviewed input set; it does not execute Make or start a game. If a file on disk conflicts with an unsaved editor draft, resolve that conflict locally.

Keep source resources separate from generated results in `vccplus-project.json`. `resourceInputs` names intentional data files, `generatedInputs` names retained generated inputs, and `outputDirectories` names result folders to exclude. These are literal project-relative paths. An approved sibling support folder remains an explicit choice; a dependency file cannot grant access to another folder. Touching an unchanged file alone does not change its content-based revision.

The build cache is a disposable working copy. VCC Plus can repopulate it when its receipts or dependencies no longer match approved inputs; it does not run an arbitrary project's `clean` recipe to repair the cache. A stale-state reply can name changed paths and expected/current identities. Read that evidence and refresh the actual project rather than deleting source or changing a request until a refusal disappears.

Let an assistant play a test copy

Select **Allow Disposable Playtests** in the C Assistant panel before saving a connection. This permits the assistant to launch the current built disk in a separate 512 KB CoCo, with MC6809 as the default and HD6309 available when the connection advertises processor selection, without changing your open machine or mounted media. It can hold combinations of physical keys and standard joystick axes and buttons for bounded video fields, capture the game's graphics, and cold restart its own writable test disk. You need not approve each test launch after granting this option. Rebuilding, changing the shared revision or revoking access ends that run.

> Build this revision's disk. Start a disposable playtest of that exact artifact. Capture its title screen, describe the controls, and test a short movement and jump. Report the run ID, field counts and what the images actually show.

The assistant uses `start_playtest`, `playtest_input` and `capture_playtest`. `restart_playtest` keeps the same writable disk, so save and load tests use the game's real controls. `stop_playtest` retains the test disk in VCC Plus

development results. Inputs have request IDs: an identical retry returns the prior receipt without repeating movement. The clock measures completed emulated video fields, not wall time or the game's own update count. A breakpoint can end an input plan early; read the returned clock and remaining field/restart allowance before continuing.

Starting a disposable playtest can return **starting** while VCC Plus verifies the build evidence and boots the machine. The assistant should poll `get_run` for progress and readiness before sending keys or debugging. Retrying the same request recovers that startup; a new request must not create a duplicate machine. The reserved run can be stopped while preparation is pending.

Keep the original project, revision, artifact and request IDs if a connection times out. `get_run` reports the preparation phase, `busy` and a terminal failure when one occurs. A ready startup has a worker `receipt`; check its actual state and stop evidence before claiming the requested function was reached. To cancel a pending start, use `stop_playtest`, then poll until cancellation has finished. An interrupted transport is not proof that no guest was created.

Typed answers can combine letters, `space`, `period`, `backspace` and `enter`. For example, the assistant can type **ST. PAUL**, correcting a letter with Backspace before submitting. These are physical CoCo keys; an empty key array releases them, and Shift works as a normal held chord. This applies to disposable playtests. The ordinary shared-run `send_keys` tool retains its smaller key set.

Test disk changes without restarting the program

A game can keep its program in memory while asking for another disk, or use Home and Data together in different drives. In **Project Settings → Build**, declare additional project-relative **playtest disk outputs**. The approved Make recipe must produce them with the primary disk; each run gets separate writable copies. Up to eight additional floppy products are supported.

Fresh multi-drive connections can select FD0–FD3. Earlier media connections keep FD0-only exchange, so inspect live capabilities after reviewing a new connection and restarting its helper. Omitted `drive` still means FD0.

> Build Home, Data and the deliberately wrong test disk. Start the owned Home run, pause at an appropriate prompt and insert the listed Data copy in FD1. Use the game's normal controls to load an uncached level, save to Home0 and reload. Compare per-drive activity and exact hashes: Data1 should remain unchanged when the game saves only to Home. Repeat the game's empty, wrong, damaged and write-protected media recovery paths without changing its memory.

`list_playtest_media` lists artifact products and their IDs/hashes. Include `run_id` to inspect that run's current private copies, because guest writes can change their hashes. `exchange_playtest_media` takes a listed media ID, its exact current hash and the target `drive`. On insert, optional `write_protected: true` provides a real protected private image; omission preserves its existing setting. The assistant supplies no Mac path. A writable copy cannot occupy two drives at once: eject it before moving it. Distinct private copies can have identical bytes.

Pause explicitly before inserting or ejecting. The shared floppy controller must also be idle: a keyboard prompt or stopped motor alone does not prove a pending command is complete, even if it belongs to a different drive. Replacement durably retains the outgoing disk first and affects only the selected drive. It does not reset the guest, change BASIC's default drive, alter RAM/registers or advance the field clock. Resume with normal input and inspect the game's response; a mount alone does not prove it accepted the volume, edition or checksum.

Fresh receipts show all four drives, their current copy identities, original and current hashes, protection, generation and actual transferred-byte activity. Read/write bytes are not counts of successful Save commands. Combine them with exact disk contents and the game's own save/reload result. Older top-level disk fields continue to describe FD0.

`restart_playtest` preserves all current drive assignments and writable copies; ensure FD0 holds the intended DOS boot disk first. It does not move Home from a nonzero drive automatically. `stop_playtest` freezes and

independently retains every disk, including ejected copies, before ending the worker. Later starts can select saved copies through `retained_media`; the primary retained-disk fields still identify the primary product wherever it was mounted. Saved originals stay unchanged. This resumes disk contents, not CPU/RAM state. Startup drive maps are not added: use explicit paused exchanges for the other drives.

Reuse a request ID only for the identical drive, selection and protection plan. After an exchange, obtain a fresh debugger snapshot even though guest time did not advance. A lost reply is uncertain; retry the identical request under current authority rather than issuing a new mutation that may repeat a completed swap.

Test a joystick without changing your controller setup

A fresh disposable-playtest connection supports both standard CoCo joystick ports, independent of your personal gamepad, mouse or keyboard assignments. For each `playtest_input` step, keep `keys` and `fields` and optionally add `joysticks`. Each port snapshot names `right` or `left`, integer `x` and `y` positions from 0 to 16383, and `buttons` containing `fire1`, `fire2`, both or neither. Center is 8191; X increases rightward and Y downward.

```
{"run_id":"<owned run>","request_id":"walk-left-fire","steps":[{"keys":
[],"fields":12,"joysticks":[{"port":"right","x":0,"y":8191,"buttons":["fire1"]}]},{"keys":
[],"fields":6}]}
```

Each step replaces the entire keyboard and both-port state. An omitted port centers and releases its buttons. Omitting `joysticks` centers both ports. Every completed or interrupted input plan releases keys and centers the sticks. The receipt's `input_trace` shows the input applied during each step and its actual completed fields; final `input` shows the released state. These observations confirm delivery, not that the game responded correctly. Read the game's controls and inspect screenshots of actual movement, jumps and menus.

`get_playtest_input` reads the owned machine's input and clock without advancing it. For the personal shared machine, `get_machine_status.input` instead separates the native controller preview, the last submission and the actual core input. It reports source, interface, connection and focus availability without revealing controller identities or unrelated key contents. Preview movement alone does not prove delivery; a paused machine releases its input. These tools do not alter personal input settings. Standard analog playtest input does not simulate the app's optional high-resolution joystick interfaces.

> Discover joystick support, then test right-port movement, Fire 1 and release through the guest's normal controls. Capture before and after frames. Compare the input trace with what the game actually did. Do not substitute keyboard arrows for a joystick-only acceptance test.

Compare two graphics pages without advancing

For graphics projects whose verified manifest declares extra buffers, pause and use `capture_playtest_graphics`. The assistant can compare the register-selected picture with one declared buffer by ID at the same stopped state. The returned images and physical identities help diagnose page flipping without writing RAM or advancing the guest. Normal `capture_playtest` still shows the last published video frame, which may precede a register change.

For a custom scrolling GIME game, the project can declare bounded `current_gime` canvases in its verified manifest. Captures preserve the live viewport, row stride, pan, palette and relative video offset when comparing the two pages. Fixed HSCREEN previews keep their BASIC layout. An escaping viewport is refused; the assistant must not mislabel a custom canvas as HSCREEN 2.

> Pause this owned run. Capture the normal framebuffer, then use capture_playtest_graphics for the declared back-buffer ID. Report which image is the last published frame and which is the paused register preview. Compare the canvas, viewport offset and physical identities without changing video registers or supplying an unapproved address.

A paused preview shows the registers and memory at one frozen state. It does not reconstruct an earlier scanout or guess an undeclared page. The project declares canvas origins, bounds and layout in verified build evidence; this capture tool accepts a buffer ID, not arbitrary memory addresses.

Keep a save between test sessions

After an app update or a later connection, `list_playtest_disks` finds the project's recorded test disks. The assistant can select one by its retained ID and exact hash when starting a new test. That cold-boots a separate writable copy; use the game's normal Load or Continue command to restore its saved data. The retained original stays unchanged; stopping the resumed run creates its own retained ID with a parent link. A recorded disk requires the same base artifact bytes. `artifact_disk_sha256` identifies the current produced disk, `disk_original_sha256` identifies the saved copy initially loaded into this run, and `disk_sha256` tracks its current writable contents. These need not be equal after the game saves. Older retained disks require their previous run ID and exact hash and report `legacy_receipt` origin: the earlier artifact provenance was not recorded and is not invented by rebuilding today.

To examine files in a saved disk yourself, keep a separate copy and use **Media → Floppy Disks… → Open Disk Image…** for the native Disk Image Editor. You can browse supported DECB or OS-9 directories and export the game's saved files. Keep the app-managed retained disk unchanged: its exact hash belongs to the resume receipt. Editing a separate copy does not turn it into a new retained MCP selection. Applying edits to a mounted disk is a separate local action that locks the machine and requires a confirmed restart.

Choose a stop the build can prove

For a project with verified debug evidence, the assistant can start at a named C function, pause or instruction-step its owned run, inspect locally configured named watches, and read only the advertised stopped-program memory ranges.

The build receipt's `source_debugging.functions` lists names with a mapped C statement. `source_debugging.function_entries` lists verified executable symbols, including assembly routines. Choose a name from the matching list. An assembly input sampler can use `stop_kind: "function_entry"`: the snapshot reports its real symbol, current address, RAM bank, registers and declaration location. It leaves `source_stop` empty where no C statement is mapped, and instruction steps there are labeled `assembly_instruction`. A declaration line is not the executing C line. Verified shared globals remain available; frame locals are not guessed.

An optimized Release build may have verified executable symbols but no typed C frame metadata. Its receipt reports `source_only`, with `executable_symbols` and unavailable variables. A verified symbol can still stop execution and expose registers and permitted code bytes. Such snapshots report `symbol_stop` and `executable_stop`; they do not invent a C function range, watch value or frame. Use the selected build's own evidence, never addresses borrowed from Debug.

A fresh C playtest connection can also advertise `get_call_context`. After `get_debug_snapshot`, use that same run and snapshot ID to read its bounded 64-byte window at the guest's current stack pointer. Raw words remain uninterpreted. Caller and integer argument values appear only where the compiled metadata and call instructions prove them; optimized or unknown layouts stay unavailable. A step, restart or rebuild requires a new snapshot. This reads only the owned disposable guest and never edits its registers or memory.

Read the error that stopped the build

When a build fails, read the build receipt's log as well as source diagnostics. It preserves bounded combined compiler, linker and Make output, the exit status and whether output was truncated. A warning list alone may omit the fatal reason. On a diagnostics-enabled connection, ask for the first actual error and any remaining build-log pages. The receipt distinguishes a compiler, assembler, linker or Make failure when the retained evidence identifies it. Successful builds can still carry warnings. Debugging receipts list verified C-line stops, function entries and available global declarations; unavailable values need an explanation rather than a guessed result.

> Read get_build for this failed build. Report first_error, failure_stage and exit_code. Use get_build_log and its next_offset to read the remaining retained output. Distinguish the actual error from warnings, and say whether any process output was already dropped. Propose a repair only after locating the cause.

The log combines compiler, assembler, linker and Make output; it does not pretend to separate stdout and stderr. A zero process exit can accompany a later `validation` failure when the produced artifact is unacceptable. If Make hides a command, the receipt cannot invent its text. Pagination retrieves retained output, not bytes discarded at the process-output limit. Keep the build ID and configuration with the diagnostic so the next repair targets the right result.

Hear the guest's next interval

An audio-enabled disposable connection can also capture a short WAV directly from the guest. Ask it to press the sound-producing key normally, capture the following interval, and report its field range, boot and sample count. Audio capture advances the guest and leaves it paused; a screenshot does not advance it. Retrying the same audio request returns the original clip without playing the interval again. This records no microphone or personal emulator session.

> Discover whether capture_playtest_audio is available. Use ordinary guest keys to trigger a sustained tone, then capture its following 60 fields. Report the WAV's sample rate, channels, sample count, boot and actual field interval. Tell me whether the clip contains changing samples. Keep the original request ID if delivery is interrupted; do not play the interval twice.

Each request accepts 1–120 fields. The result is a standard MCP WAV audio block: 44,100 Hz stereo, signed 16-bit PCM, before the Mac's volume control. Audio capture releases keys and drains preceding audio before advancing, so a short beep that ended during the input plan may already be over. A clip can end early at a verified stop. Check `fields_advanced` and `completed`; do not call an unheard or clipped interval a successful sound test. A constant DAC level can have a DC offset without audible variation.

Each run has bounded field, restart, mutation and audio allowances. Read `remaining_mutation_requests`, `remaining_audio_captures` and `remaining_audio_bytes`, as well as field and boot allowance. Audio is limited to 16 clips and 4 MiB across the run's boots; a new clip needs room for a maximum clip. `stop_playtest` remains available for cleanup and disk retention even after execution requests are exhausted. If retention is pending, keep the app open and retry the original stop request until the receipt confirms the disk was saved.

Keep observation separate from a debugger edit

> Read this build's two function lists. Stop at the input sampler using a verified function entry, report the symbol and physical bank, and inspect my configured player-state watch. Execute one instruction and report what changed. Do not alter registers, inventory or position through the debugger.

Those controls do not write registers, move the character or alter inventory through the debugger. Report build, launch, source debugging and gameplay results separately; a compiled program alone proves none of the later steps.

Rename a function across files

Reopen and share your repaired Hello Disk practice copy. Ask ChatGPT to rename `showHeading` to `showDiskHeading` throughout the project in one proposal. Look for a declaration in `include/hello.h`, a definition in `src/display.c` and a call in `main.c`. Open each file in the review window's **File** menu. Check the return type, arguments and spelling at all three places. **Apply All Changes** updates the drafts together. **Cancel Changes** keeps the current drafts. If you add a new project file yourself, use the connection's project refresh and approve the added file locally; older connections need a fresh export. Rebuild the applied revision and run it to check that your rename preserved the behavior.

Move a block one cell to the right

Chapter 5 of 9

On a 32-column CoCo text screen, the upper-left cell has a memory address, `$0400`. The next cell is at `$0401`. In this lesson you'll ask ChatGPT to move a block between them. You can check its work one instruction at a time and tell exactly when the CoCo display changes.

Put a byte on the screen

Open **Develop → Assembly Development...**. For the exercise, use a spare Disk BASIC machine at its `OK` prompt in 32-column mode. If you need to restore that width, type `WIDTH 32` at the guest prompt and press Return. Save any program you want to keep before loading another.

Make a writable copy of the screen-cell source. Choose **File → Open Source...** in Assembly Development and open the copy. Set the assembler target to **MC6809**. This is the processor the program is written for. It runs on either an emulated MC6809 or an emulated HD6309, including when your Mac itself has an Apple M-series chip.

```
        org     $3f00
start   lda     #$8f
        sta     $0400
        rts
        end     start
```

A dollar sign introduces a hexadecimal number. The first line tells LWASM, the assembler, to place the machine instructions starting at address `$3F00`. The word `start` names the first instruction. The last line tells the assembler where execution should begin.

Read the three instructions in the middle as a set of directions. `LDA #$8F` puts the byte `$8F` into A, a small storage location inside the CPU called an *accumulator*. The `#` tells the CPU to take that value directly from the instruction. `STA $0400` copies A to the screen address. The display hardware interprets that byte as a solid block. `RTS` returns to the caller. The VCC Plus loader sets up a BASIC caller for this program.

You can find the original exercise under **Quick Reference → First Program — Change a Screen Cell**. That example branches to itself so you can inspect the result for as long as you like. The book's version ends with `RTS` so you can also practice returning to BASIC.

Follow the first two instructions

1. Click **Assemble**. Open **Build details** and find `start` in the listing at `$3F00`. Assembling creates the binary. The CoCo starts executing it when you launch it.
2. Choose **Debug from Entry** and confirm the load. The program counter, labeled **PC**, should read `$3F00`. It points at the instruction the CPU will execute next.
3. Click **Step** once. A contains `$8F` and PC reads `$3F02`. At this point the program has only loaded A. Before taking the second step, locate the upper-left cell on the guest display.
4. Click **Step** again. The store has executed and PC reads `$3F05`, the address of `RTS`. Look at the upper-left cell. In the debugger's mapped memory view, the byte at `$0400` is now `8F`. If the cell already showed a block, the byte still lets you verify the store.
5. Choose **Continue**. The CoCo returns to `OK`. BASIC can now write to the display again, which is why the time to inspect that byte was just before `RTS`.

Give ChatGPT one instruction to change

In the assembly editor choose **File → Local Assistant Access...**, then **Save Assembly Connection...**. Open the connection with **Add to ChatGPT** and start a Work chat for this project. Give ChatGPT a job with a visible result.

> Read my shared assembly source. Tell me which instruction chooses the screen cell. Propose a one-line edit through VCC Plus that moves the block one cell to the right. Keep the value and the MC6809 target the same.

In the review, look for `sta $0400` changing to `sta $0401`. The other two instructions still have work to do. If the proposal instead changes the number after `#`, it changes what gets displayed. Cancel that edit and ask for the destination address to be changed. Choose **Apply Edit** when the proposed instruction is right. Save the practice copy.
Ask ChatGPT to assemble the **current shared revision** and report its entry point and any diagnostics. In the listing, place the caret on the executable `rts` line and choose **Set Stop**. Bring the disposable machine back to the 32-column BASIC prompt. You can type `CLS` there first to make the new block easier to see. Choose **Run Shared Build...** in the access panel and confirm the revised build. Each run starts with your local approval. A successful run of the earlier binary cannot verify this edit.

> At the stop for the run I just started, read the source line, PC and register A. Read the three instruction bytes at $3F02 using the current debugger snapshot. What screen address does that instruction specify?

The stopped instruction should be `rts`, PC should be `$3F05`, and A should be `$8F`. The three bytes are `B7 04 01`. The assembler encoded the destination address in the last two bytes. Build 30 shares the mapped instructions and the first screen byte with the assistant. You can inspect the second screen byte yourself in the native debugger at `$0401`. It should read `8F`. When the quiz in the next chapter puts three full lines of answers on screen, the CPU will still be doing work a byte at a time.
Choose **Continue** to return to BASIC. If the debugger reports that a snapshot is stale after you continue or edit, start the current build again and take a new snapshot at the stop.

Try the second processor

1. Save the revised assembly source and anything you want to keep from guest RAM. On the spare machine, open **Machine** → **Machine Settings...** → **Hardware** and choose **Hitachi 6309**. Choose **Apply and Restart...** and confirm. The processor change cold-restarts the CoCo. Wait for Disk BASIC's `OK` prompt.
2. Back in Assembly Development, change **Target** to **HD6309**. Ask ChatGPT to check the actual guest processor with `get_machine_status`, then assemble the current revision and report the assembler target. Check **HD6309** in both places. If the project needs reconnecting, save its connection again before requesting the build.
3. Set the stop on `rts` for this build and choose **Run Shared Build...** again. The earlier approval and snapshot belonged to the previous run. Inspect the new stop. Do you still see PC `$3F05`, A `$8F`, instruction bytes `B7 04 01` at `$3F02`, and `8F` at `$0401` in native memory? Those are the expected results. Both processors implement these instructions with the same encodings.
4. Choose **Continue** and check for the BASIC prompt. Revoke the assembly connection when you finish. To resume a practice setup that expects an MC6809, save any guest work and restore the processor in Machine Settings and the target in the editor.

Build a quiz and give ChatGPT the keyboard

Chapter 6 of 9

"What is the capital of Kentucky?" makes a good test question. A quiz that offers Louisville as a choice should let you pick it, tell you the capital is Frankfort, and count the miss exactly once. You can check all three behaviors by playing. You'll use the same test for a BASIC listing, a C program and a 6809 assembly program.

Play a round from the floppy

Choose the capitals game disk and save a copy as `CAPITALS.DSK`. Start a spare **Disk BASIC** machine and mount the image in drive 0. You can write-protect it. Give the CoCo display keyboard focus. At the `OK` prompt, type one of these commands and press Return.

Game	Command at the CoCo prompt	After the fifth question
Extended Color BASIC	`RUN"CAPITALS"`	Press **1** for another round or **2** to quit.
CMOC C	`LOADM"CAPSC":EXEC`	Press a key for another round. To switch games, restart the spare guest and return to BASIC.
MC6809 assembly	`LOADM"CAPSASM":EXEC`	Press **Y** to replay or **N** to quit.

Press Space at the title and wait for question one. Try **0** while the quiz is showing its three choices. You should still be on the same question. Choose **1**, **2** or **3**, read the feedback, then press Space to advance. There is no need to press Return after each choice. These games watch for individual keys. Pressing Return again can carry you past the feedback you wanted to read.

The disk is a convenient way for *you* to play. To let ChatGPT take a turn, open one of the editable projects below and start its shared run. You can do the rest of the chapter without mounting the floppy.

Give the writer a playable brief

The three versions began in separate ChatGPT Work chats. Each chat had access to one VCC Plus workspace. If you want to write your own version, start with one language. Save any existing work and use a new practice draft or project folder. Here's a brief you can reuse.

> Make a US state-capitals quiz for a CoCo 3 with a 32-column text screen. Include all 50 states. Ask about five different states per round. Offer three distinct capitals numbered 1, 2 and 3 and vary the position of the right answer. Retry an invalid key on the same question. After a miss, show the right capital. Wait for a key before clearing feedback. Show a score out of five and offer another round. Submit the complete source through my VCC Plus connection for review.

Add one of these directions, depending on what you've opened.

- **BASIC.** Ask for a numbered Extended Color BASIC listing using `INKEY$` for single keys. Ask the writer to reserve enough string space for the names and keep variable names distinct in their first two characters. In the saved BASIC listing, line 20 reserves 5,000 bytes and line 40 reads the 50 pairs into arrays. A typo in the pair for Kentucky would affect every round that draws Kentucky.
- **C.** Create a project with an editable `main.c` in C Development. Ask for a CMOC program using the bundled CoCo library for keyboard and screen I/O. Ask ChatGPT to propose the complete replacement for `main.c` through the project connection. The saved C source keeps a state's name alongside its capital in a

`Place` structure. Download the complete CMOC project for the same source and the Makefile VCC Plus uses to build it.

- **Assembly.** Start an assembly project with a `main.asm` and select **MC6809**. Ask for LWASM source with an entry address, the Disk BASIC ROM's character-input and output routines, and an `RTS` path for quitting. The saved assembly source uses `POLCAT` to poll the keyboard and `CHROUT` to print a character. It uses the time you spend on the title screen to choose a starting state.

At review, read the Kentucky entry and follow one path from a wrong choice to the score. In particular, look for an opportunity to count the same question twice when a player tries an invalid key. Apply a proposal you want to try, then save the draft or project. A chat containing a pasted listing can be useful for discussion. The editable game you can reopen lives in the saved project.

Run the saved sources with ChatGPT

Start a fresh Disk BASIC session in VCC Plus. Use a disposable machine with a floppy controller, the cassette ejected, and mounted disks ejected or write-protected. If C reports that a BASIC program is already in memory, save any listing you want to keep and start with an empty BASIC session. Don't clear an unfamiliar program to get past the warning.

1. **BASIC.** Select CAPITALS.BAS in the Mac reader to open an editable copy in BASIC Development. In a browser, save the listing and open it in **Develop → BASIC Development...**. From that editor's **Local Assistant Access...**, enable **Allow proposed edits and controls for runs I start** and save a review-and-run connection. Use **Add to ChatGPT** and start a Work chat. Opening an existing listing is enough for manual play. For the shared run, VCC Plus also needs a proposal you've applied to the current revision. Ask ChatGPT to change only the comment on line 10 to `10 REM US STATE CAPITALS QUIZ`. Compare the two lines, choose **Apply Edit**, then **Run** on the applied proposal in the access panel. To play the exact original over MCP instead, begin with an empty practice draft and have ChatGPT submit the supplied numbered listing verbatim. Compare it with the saved file before applying and running it.
2. **C.** Save the CMOC project and expand the ZIP in Finder. Open the `CapitalsCMOC` folder containing `main.c` and `Makefile` in **Develop → C Development...**. Choose **Assistant...**, save its private project connection and use **Add to ChatGPT** in a fresh Work chat. Ask ChatGPT to build the current shared project and report any diagnostics. After a successful build, choose **Run C Build... → Play in CoCo**. The other choice, **Inspect at main**, is useful when you want to stop at the first source statement. For this round, wait for the title and the prompt to start.
3. **Assembly.** Save main.asm, then open the saved file in **Develop → Assembly Development...**. Confirm **MC6809**, then use **File → Local Assistant Access... → Save Assembly Connection...** and add this connection to its own Work chat. Ask ChatGPT to assemble the shared revision. Check the successful build and entry point `$3F00`. Clear any source stop you set while experimenting. With the guest at the BASIC prompt, choose **Run Shared Build...** and confirm the launch.

Each choice above is a complete route to a game. You can test one language today and come back to another later. If you do test all three, give each connection a distinct file and return the disposable guest to a clean BASIC prompt between games.

Watch someone else take the quiz

Leave the quiz on its title screen. In the Work chat connected to that game, give ChatGPT both the controls and a stopping point.

> Play one round using the active VCC Plus run. Read the guest screen and send the game keys yourself. Start with Space. Try 0 on the first question, then answer it correctly. Deliberately miss question two and read the capital the game gives you. Answer the remaining questions correctly. After each answer, read

> the feedback before sending Space. If the screen is slow to change, read it again. Stop at the replay prompt. Report the missed state, its capital and the score exactly as displayed.

The player obtains the active run ID with `get_run`, reads with `read_screen_text`, and sends a single letter, digit or space with `send_keys`. Each intended keypress gets its own request ID. A retry of the *same* key request reuses that ID so it won't count as another press. Give the player time to observe the screen between presses. If it stalls, ask what the guest **currently displays**. Seeing the replay menu yourself is a good way to check the final report.

What happened on the M2

On November 9, 2026, the first complete submissions for all three languages ran in VCC Plus build 29 on the M2 Mac. The operator reviewed the sources, authorized the runs and played the games. The original BASIC listing reached its title screen. Bundled CMOC and LWASM each built its first submission with zero diagnostics. The operator played each round to **4/5**. All three saved game sources are byte-for-byte copies of the first proposals. Independent emulator runs on both guest processors also brought every one of the 50 states to the screen in each language.

With VCC Plus 0.1.0 build 30, the operator set up three fresh connections on the M2, reviewed and started BASIC, and built and launched C and assembly. This time the stock **ChatGPT for Mac** app sent every game key in three separate Work chats. A separate observer read the guest screen. Each player tried **0**, deliberately missed the second question, and stopped at the replay prompt.

Saved game	Correction displayed after the miss	Score in the guest
BASIC	Rhode Island — Providence	`YOUR SCORE: 4 /5`
C	North Carolina — Raleigh	`FINAL SCORE: 4/5`
Assembly	Kentucky — Frankfort	`SCORE: 4/5`

```
YOUR CHOICE (1-3)?
CORRECT!
PRESS ANY KEY TO CONTINUE

QUESTION 5 OF 5
STATE: PENNSYLVANIA
1. SACRAMENTO
2. HARRISBURG
3. BOSTON
YOUR CHOICE (1-3)?
CORRECT!
PRESS ANY KEY TO CONTINUE

SCORE: 4/5
PLAY AGAIN? Y OR N
```

Figure 4: Assembly quiz showing the correct answer to the fifth question, Pennsylvania, followed by SCORE 4/5 and PLAY AGAIN? Y OR N.

On that last screen, the assembly game is still running and waiting for a decision. If you want to test its exit path, ask ChatGPT to send **N** once. The program prints `THANKS FOR PLAYING!` and returns to BASIC. A finished run can report its final screen. Its game-key controls close when it returns. Revoke the connection after your round.

Grow the idea into Capital Quest

The five-question games above make the connection easy to understand. A later project, **Capital Quest**, took the same subject much further: every state in one 50-question quiz, a color map, typed answers and a review of mistakes. Its public CoCoCapitalQuiz repository contains the editable C project, map inputs and credits, a ready-to-play `CAPITAL.DSK`, sample screens, tests and the development brief. This is a separate game from the book's `CAPITALS.DSK` and its three short examples.

The core prompt generated an almost feature-complete program with ChatGPT Codex using GPT-6.1 Sol High. The initial delivery already included both quiz modes, all 50 states, map highlighting, spelling correction and a review of missed answers. VCC Plus supplied the installed toolchain and public MCP interface for building and playing it. The target is a **512 KB CoCo 3 with an MC6809**, with a 320 × 192, 16-color map and Northeast/Hawaii detail views.

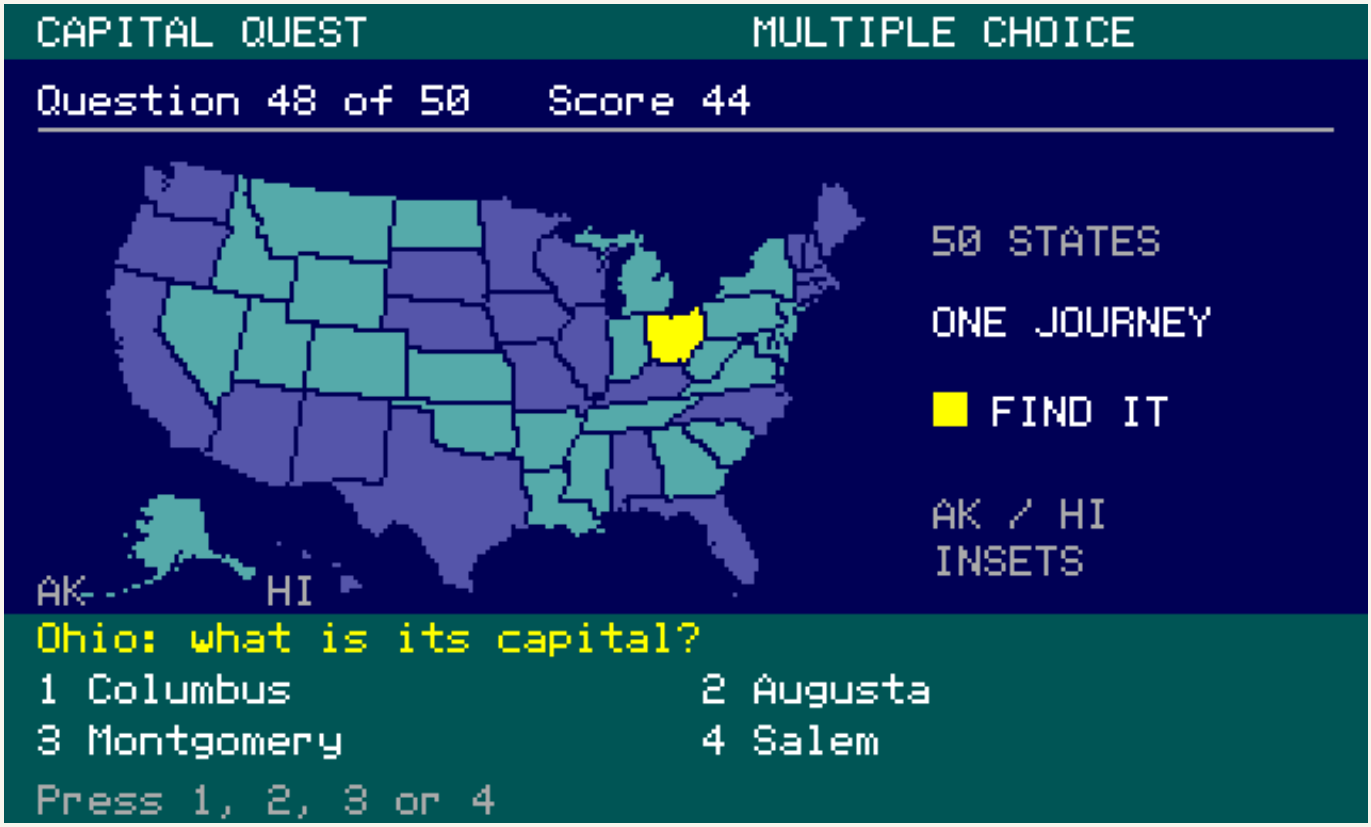


Figure 5: Earlier Capital Quest edition: Question 48 of 50, score 44, Ohio highlighted on the national map, and Columbus among four capital choices.

The map is adapted from © Fla-shop.com, under Creative Commons Attribution 4.0. The adaptation reduces resolution, simplifies boundaries, recolors and encodes the drawing data, and adds detail crops. The repository's `assets/CREDITS.md` retains the source, license and adaptation notices; **C** on the game's title opens its credits.

Give a larger project a richer brief

Open the project's complete CHATGPT_PROMPT.md to see what the writer was asked to deliver. Notice how much more it settles than "make a capitals quiz." It specifies behavior, constraints, evidence and what to preserve while working.

Part of the brief	What it lets you check
Ask all 50 states exactly once, numbered 1–50	A complete round can reveal duplicates, omissions and off-by-one numbering. Alaska and Hawaii are included.
Offer four distinct choices and a typed mode	Both the menu player and the line editor need real tests.
Accept case and spacing variants, but handle a near spelling carefully	A close answer gets a correction opportunity, without an early point or an immediate answer reveal. Another state's exact capital remains a wrong answer.
Highlight the question's state and enlarge small regions	The map must agree with the data and remain useful at CoCo resolution.
Preserve attribution and convert graphics before running	The guest uses compact drawing data; it does not try to render an SVG.

Part of the brief	What it lets you check
Use ordinary guest input and actual framebuffer captures	The test must observe the running program, rather than setting its score through a debugger.
Keep source, assets, build instructions and test limitations	Another reader can rebuild, inspect and improve the result.

The brief also asks the assistant to discover the installed tools and consult the manuals before choosing APIs. It gives the writer room to make routine design decisions while keeping the target machine, program behavior and testing method clear. You can use the same pattern for a word game, a music lesson or a small business application: describe a complete experience, then give it observations that can establish whether that experience works.

If you reuse the brief to create your own version, change its checkout path and read its assumptions first. It describes the original empty project. Today's download already contains a game. Use a new practice folder when you want to start over; preserve the supplied project when you want to study or extend it. The current prompt also includes a later **Addendum** for sound and automatic multiple-choice progression.

Follow the build and test loop

Open the project in C Development and use its Make recipe to combine C sources with the prepared map data. In **Assistant...**, enable **Use Project Makefile** and **Allow Disposable Playtests**, review the shared inputs and save a private connection outside Git. Begin with live tool discovery and `get_capabilities` so the assistant identifies the project, revision and available build choices.

Ask it to build the declared disk action in a named configuration and read `get_build`. When the build succeeds, use its artifact ID with `start_playtest` and poll `get_run` until ready. This creates a separate writable test disk and 512 KB guest, preserving the personal machine. Send ordinary physical keys with `playtest_input`, include release fields between presses, and inspect the actual framebuffer with `capture_playtest`.

Capital Quest's tests compare drawn text against the font in the matching build and selected-state pixels against the map masks. You can also inspect the pictures yourself: does the named state light up, are the four choices readable, and does the displayed score change when it should? For a logic question, use the build's advertised C-line or function-entry evidence and read-only snapshots. For sound, `capture_playtest_audio` supplies the guest's following bounded interval as a WAV. Chapter 4 explains its timing and key-release behavior.

Host logic tests and guest play complement each other. Host tests exercise thousands of shuffles and wrong-capital pairs quickly. Full guest rounds check input, screen layout, numbering, feedback, scores, review, replay and exit. The project's `Test` configuration uses seed **2709** for repeatable rounds; ordinary `Release` play mixes timing and start input into its seed.

Keep each result with its build ID, configuration and artifact. Choose Test when reproducing a particular sequence, and finish its run before building another configuration. Save evidence in a declared output folder or outside the watched checkout so captures remain separate from program inputs. Use scoped project refresh after changing shared files or build controls, as explained in Chapter 4. Stop the owned worker when you finish and keep its retained disk receipt with your observations.

See what the complete program delivers

A rich brief gave the project its shape. The core program shuffles all 50 states, offers four-choice or typed play, keeps the highlighted map associated with each question, and finishes with a score and paginated review. A close spelling lets the player edit or reveal; a successful correction earns one point and is tracked separately. This is enough to make a complete learning game from the original idea.

The next refinement added rising tones for correct answers and falling tones for misses using the CoCo's built-in sound. Multiple-choice feedback advances automatically after a readable pause, while typed mode keeps

Space-to-continue. The prompt's Addendum captures that request. It is a useful example of shaping an already working program through a focused follow-up.

Further refinements made the map show the player's answer history: light teal for unanswered states, purple for correct answers, red for misses, and yellow for the current question. The current state stays yellow through feedback and takes its answer color when the next question opens. Northeast and Hawaii details follow the same rule, and a new quiz clears the history. These colors have no regional meaning.

Another follow-up changed multiple-choice placement. The first correct answer may occupy any slot; each next question chooses randomly from the other three. It excludes the previous question's correct slot even when the player selected a wrong answer. Host tests checked 100,000 question sets, and complete guest rounds checked the choices actually displayed. This is a useful refinement to test separately from shuffling the state order.

The project's test records include these complete rounds:

Playtest	Recorded outcome
Multiple choice with deliberate misses	All 50 states once, numbering 1–50 and a final score of 42/50.
Typed answers with spelling exercises	All 50 states once, 47/50 and two successfully corrected spellings.
Ordinary Release play	A complete 50/50 round and the corresponding review.

The mixed-answer rounds exercise scoring and feedback as well as correct answers. Tests also cover casing, multiword capitals, aliases, Backspace, long input, replay, held-key handling and clean return to BASIC. Read `docs/TESTING.md` and `docs/FEEDBACK_TESTING.md` for the original scenarios, and `docs/MAP_PROGRESS.md` and `docs/ANSWER_SLOTS.md` for the later map and slot checks. Those are dated development records. See `docs/evidence/` for the retained receipts.

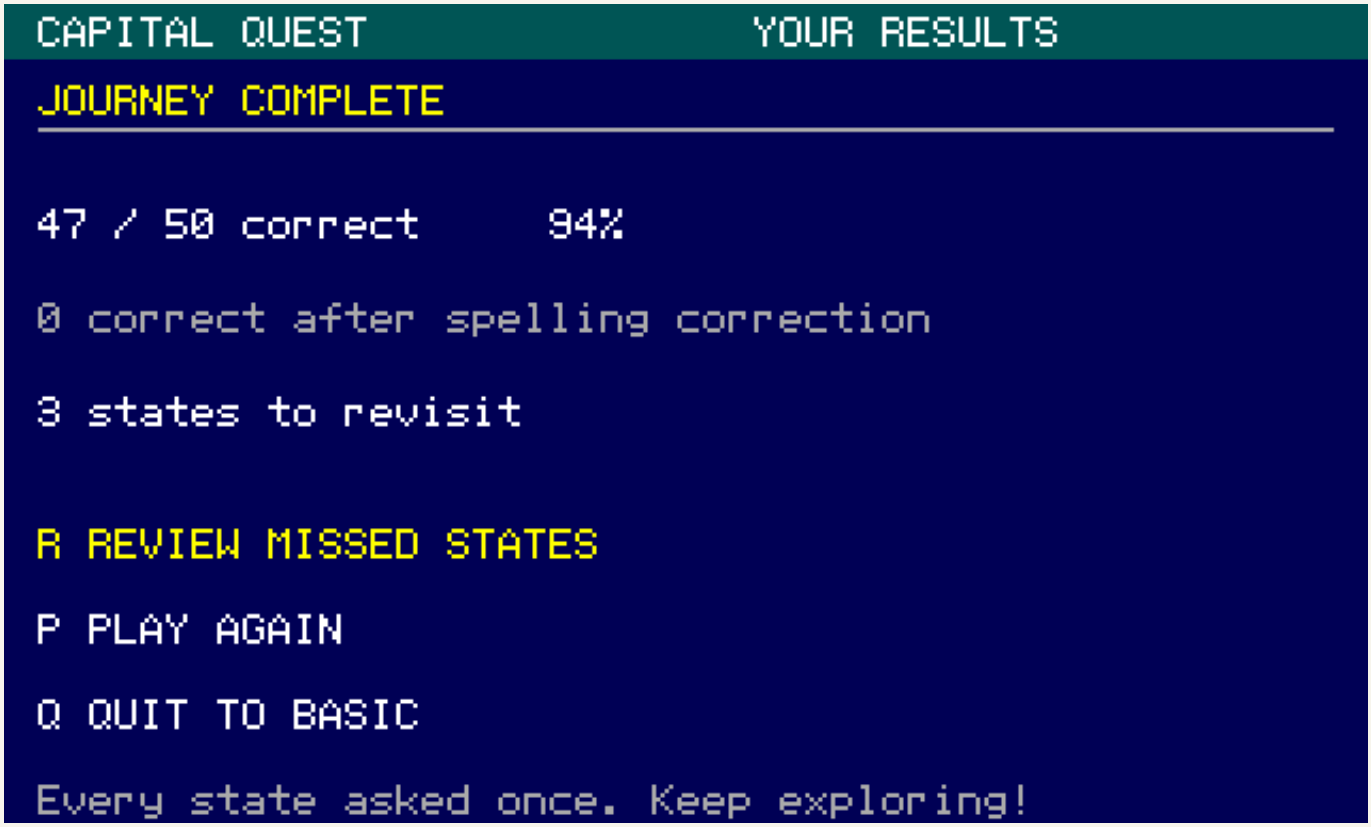


Figure 6: Earlier Capital Quest results: 47 of 50 correct, 94 percent, zero spelling corrections and three states to revisit, with review, replay and quit controls.

This results screen gives the player a useful next action: review the three missed states, play again or return to BASIC. The same idea belongs in your own program's completion screen. A score is more useful when the player can see what to practise next.

Try the delivered game, then change one rule

1. Read the project's README and player manual. To play the delivered disk, mount your own `CAPITAL.DSK` in **FD0** of a **CoCo 3 with 512 KB RAM and Disk BASIC**. Preserve existing work, choose **Cold Restart**, wait for BASIC's `OK` prompt, then type `DOS`, or `LOADM"CAPITAL.BIN":EXEC`. RGB is the tested color presentation; both emulated MC6809 and HD6309 have been checked. Keep personal media separate. This route needs no assistant connection.
2. To inspect or rebuild, obtain the repository files through your browser or your usual Git client. Open that folder in **Develop → C Development...**. Select **Release**, then **Build**. The declared target builds the disk; use **Show Build Products** to locate this build's `CAPITAL.dsk` and binary. Preserve the map's `assets/CREDITS.md` and source notices with your copy.
3. To repeat the assistant workflow, use **Assistant...**, review the shared inputs, select the Make and disposable-playtest options, and save a fresh private connection. Discover its tools and ask it to identify the project, current revision and advertised configurations before building.
4. Ask for a short observable check before a full round. The prompt below connects the current tools to a behavior from the original brief.

> Build this project's advertised Test disk and start a disposable 512 KB MC6809 run of that artifact. Poll startup until ready. In typed mode, read the state from the actual framebuffer, try a close spelling, and capture the correction prompt before fixing it with ordinary keys. Show that the score changes once. Report what passed, what you could not observe, and the retained disk receipt when you stop. Do not change guest memory or registers.

The **November 12, 2026 game edition** uses **1** for multiple choice, **2** for typed answers and **C** for credits. **Q** (or lowercase **q**), **Esc** and CoCo **Break** quit to BASIC from every screen, including questions, spelling help, feedback, review and the automatic reading pause. VCC Plus maps the Mac's **Esc** key to CoCo Break. Release each key before the next action.

Multiple-choice answers use **1–4**. Typed answers use letters, spaces, periods, Backspace and Enter. Press **Enter with an empty answer** to reveal the capital and count the question as a miss; clear any text with Backspace first. A near spelling offers **R** to edit or **V** to reveal. Q, Esc and Break quit rather than revealing an answer. Typed feedback waits for **Space**; multiple-choice feedback advances automatically. After a complete round, **R** opens the missed-answer review, **N** advances its page, **B** returns to results and **P** starts again.

Use the matching disk and player manual together. An existing Getting Started installation or personal source copy can contain an earlier edition. Reading the new online manual does not replace that disk. To use these quit controls, download the November 12 edition through the project's player guide and load a separate copy; preserve your existing disks and edits.

Now choose one small refinement of your own. Perhaps you want a longer reading pause or a different highlight color. Predict which observation should change and which should remain the same. Review the source edit, build the new revision and repeat that check on its actual artifact. The useful outcome is a program you can inspect and improve, together with evidence that explains its behavior.

Find the first wrong result

Chapter 7 of 9

Suppose a game awards one point for a correct answer. You start with seven points, answer once, and finish with nine. Which part needs fixing? The score might have gone up twice, or the game might have printed the wrong number. You can settle that question before changing a line.

Stop before the score is printed

Use a separate BASIC draft for this exercise. Save any work you want to keep in the editor and on the guest. Share the practice draft with review-and-run access. Ask ChatGPT to propose the following listing exactly, including the deliberate scoring mistake.

```
10 S=7
20 S=S+2
30 PRINT "SCORE";S
40 END
```

1. Review the four lines in VCC Plus and choose **Apply Edit**. In the BASIC editor, click the gutter beside line 30 to set a breakpoint. Open **Variables and Watches** and add the watch `S`.
2. Start the reviewed program with **Run** in its assistant panel. When line 30 is highlighted, lines 10 and 20 have executed. The `PRINT` at line 30 is still ahead of you.
3. Read `S` in the native workspace. Ask ChatGPT to make its own observation.

> Use the current BASIC run and get_debug_snapshot. Which numbered line has the guest reached? What value does my watch S show? Wait for me before proposing a change.

The expected stop is line 30 with `S = 9`. The guest already holds the wrong number before it draws the score. **Continue** in VCC Plus lets the pending statement print it. Wait for the guest to return to its BASIC prompt.

> The rule is one point per correct answer. Use the current revision to propose the smallest change that fixes the measured error. Keep the starting score and the display line.

The repair you want to see is `20 S=S+1`. Check the line in the native review. Apply it, confirm the breakpoint and watch are still set, then start a **new** reviewed run. The watch should now show eight at line 30. Continue and compare that value with the score printed in the guest. Save the corrected draft if you want to keep it. You have tested both places where the error could have appeared.

Choose a repair the evidence supports

A compiler diagnostic can identify a name that the compiler cannot resolve. It cannot tell you that a game should award one point. In the C version of this exercise, an expression such as `score += 2;` builds successfully. A request for a diagnostic-backed C line repair would have no diagnostic to attach.

For a scoring mistake in a shared C or assembly project, ask for the source statement responsible and a project edit. Even a project edit can touch just one file. Read the before and after text in VCC Plus. Does the change correct the arithmetic, or does it merely print the number you hoped to see?

> My rule is one point per correct answer. The watch shows the wrong score before printing. Read the current project and propose one change to the calculation. Tell me which input will distinguish that repair from printing a fixed number.

When you apply a project repair, ask for a build of its new revision. Start the resulting program locally and replay the same input. A second run with a different starting value will test whether the repair generalizes.

Read the stop you actually reached

For an ordinary C project, add a watch in the C workspace before starting the shared build. Choose **Run C Build… → Inspect at main** when you want to pause at the first verified statement in `main`. Choose **Play in CoCo** to run an interactive program. Play records the entry stop and then continues. A watch from that stop describes entry even if the game is now showing its fifth question.

Here is a recorded observation from a two-file C project in installed build 29. Build 30 retains these debugger tools. The project defined the unsigned integer `beaconWord`. A reader trying a different project will see different addresses and names.

Observation	What ChatGPT read through MCP
Verified source stop	`main.c`, line 8
Program counter (PC)	`$2E71`
Configured watch	`beaconWord = 19754`
Readable code in the snapshot	`$2E6D` through `$2E97`
Three bytes at the PC	`17 00 16` in hexadecimal

Try this with a watch that belongs to your project.

> Read the run I started. From get_debug_snapshot, report the source stop, PC and available watches. If three bytes starting at the PC fit inside one reported range, use this run and snapshot to read them. Say which values you observed.

`get_run` identifies the launch. `get_debug_snapshot` supplies the snapshot ID and the ranges captured at the stop. `read_debug_memory` uses both IDs. The dollar sign in `$2E71` marks a hexadecimal address. The three bytes in this example came from that address in the captured program. A watch marked unavailable is worth reporting as such. Select a mapped executable statement and an in-scope variable for your next native debugging session.

For assembly, a useful place to stop is the final `RTS` after a calculation and a store. Set the source breakpoint on the return instruction before choosing **Run Shared Build…**. At that stop the store has executed and the register used by the calculation still holds its value. In a tested MC6809 routine, A and the byte at `$0400` both contained `$29` (41). Check the readable ranges before asking for that byte in another program.

Compare the two processors

For a C project, **Allow Disposable Playtests** in its Assistant panel grants isolated runs of its built disk. A fresh connection can start a 512 KB MC6809 (the default) or HD6309. Ask the assistant to discover its current tools first. Changing the Mac's processor architecture does not change the emulated CoCo CPU.

> Build the current compatible project once. Start one disposable MC6809 run, stop at a verified function, and record the artifact, registers, available globals and framebuffer. Stop and retain its disk. Start a separate HD6309 run of that same artifact and repeat the observation. Report actual processor, execution mode, MD and any unavailable evidence. Use ordinary guest input.

The 6309 resets into emulation mode. Only a guest instruction changes it to native mode. Its debugger also exposes E, F, W, V and MD; a 6809 has no such registers. Selecting a 6309 does not rebuild your code or supply missing C metadata. A cold restart keeps the selected CPU and writable test disk; choose a new run to change processors. An early startup reply names the requested CPU; wait for a ready receipt before claiming the observed mode.

Both input plans and audio capture release physical keys when they finish. Audio capture also releases keys before recording its following bounded interval. A sound clip taken after a movement plan does not prove that the movement keys remained held throughout that clip.

Repeat one observation after a change

If you edit the draft while ChatGPT is preparing a repair, stop there. Have it read the current revision, compare it with the intended change, and submit a fresh proposal if the repair still belongs there. A new build can move a statement to another address. A reset, a different mounted medium or a new launch can invalidate an earlier run. If the app reports a stale run or snapshot, obtain the identifiers for the matching launch and stop again before reading memory.

For a last pass, choose one behavior the repair should preserve. With the four-line BASIC exercise, try a starting value of zero in your practice copy. Predict the score, rerun, and inspect the same watch before the same print. A result of one shows that the program adds a point to its starting score. Simply replacing the output with a literal eight would have failed this second check.

Recorded check. On both the MC6809 and HD6309 guests in the build 30 emulator core, line 30 stopped with `S` equal to 9, 8 and 1 in the three versions. After Continue, each guest printed that same score and returned to `OK`.

Keep a field guide at hand

Chapter 8 of 9

Use this page when a conversation and the CoCo seem to disagree. Start with a question the machine can answer now.

> Identify the VCC Plus draft or project this connection shares. Report the guest CPU and the first reason a local run might be blocked.

Find the next action

What you see	What to try
ChatGPT describes a different listing.	Ask for the shared title and first numbered line through `get_capabilities` and `read_source`. Share the intended BASIC editor through its own **Local Assistant Access...**. C and assembly each use a separate project connection.
The saved connection appears in settings, but a new chat cannot read the source.	Keep VCC Plus and the shared workspace open. Use **Check Setup** in the connection panel, open a fresh **Work** chat, and check `/mcp`. If you replaced the grant file, restart the corresponding server so the helper reads the new grant.
ChatGPT can read BASIC but cannot propose an edit, or an older connection lacks newer tools.	Save a fresh connection from the current app with the permissions you intend to grant. BASIC review uses **Allow proposed edits and controls for runs I start**; C has separate Make and disposable-playtest choices. Restart the corresponding server and discover its actual tools.
A proposal seems to have gone missing.	Ask for `get_proposal` using the returned proposal ID. A pending change awaits local review. If the draft has changed since the proposal, read its new revision and ask for a new proposal.
The C project builds in the workbench but the assistant build fails.	Check whether this grant uses the restricted recipe or the approved **Use Project Makefile** mode. Read `get_build.first_error` and the retained `get_build_log`. Inspect approved inputs and refresh locally changed build controls before trying again.
Build succeeded, but there is no run ID.	Check the run blockers in `get_machine_status`. In VCC Plus start the matching reviewed listing or shared build. For an interactive C game, choose **Play in CoCo**.
A run is waiting for a key.	Read the guest prompt. For a shared single-key menu, send one permitted key, then read again. An owned disposable run can use bounded physical input or `playtest_debug` to continue a verified stop; identify which kind of run you have first.
The guest asks for a whole line, or displays only a picture.	An owned disposable playtest supports `playtest_input` with letters, `space`, `period`, `backspace` and `enter`, plus framebuffer captures. Ordinary shared `send_keys` retains its smaller key set; type a whole line locally through **Input**

What you see	What to try
	→ **Focus CoCo Display** when that is the connection you are using.
A short compiled program has already returned to BASIC.	Ask for `get_run` and its optional `final_screen_text`. This is the text captured when the program returned. Live key and screen access have ended. A graphics-only run may have no final text.
A key, run or memory read is refused after a change.	Stop sending input. Check the project revision, the guest and its media. For a new launch get the new run ID. For a new debugger stop get a new snapshot and read within one of its reported ranges.
Startup took longer than a transport request.	Keep the reserved run and original request IDs. Poll `get_run` for phase and readiness. A ready receipt is separate from reaching a breakpoint. Use `stop_playtest` to cancel; do not create a second start request to guess whether the first worked.
A C function has no mapped stop.	Read `get_build.source_debugging`. A name in `functions` supports a C-line stop; a verified entry in `function_entries` can use `function_entry` without claiming a C statement. A source-only build may expose an executable symbol with unavailable variables.
The run has reached a request or audio limit.	Read the remaining budgets. Stop and retain the owned disk with `stop_playtest`, which is exempt from execution request limits. If saving is pending, retry the original stop request; keep the app open until retention succeeds.

Keep the program you already have

An editor draft on the Mac and a listing in CoCo BASIC memory can contain two different programs. If the C workbench says BASIC is occupied, save the open editor draft first. At a supported BASIC prompt, use **Read from CoCo** in BASIC Development, then **Save As...** to preserve the guest listing in its own file. Read-back saves program text. The variables and the place where a program paused belong to that execution.

Only when you want to clear the guest listing and its variables, type `NEW` yourself at the CoCo's `OK` prompt. Ordinary C needs empty BASIC memory. The Hello Disk verification also needs an empty floppy drive 0 for the disk it creates. For a shared run, finish recordings, eject the cassette, and eject or write-protect writable floppy and hard disks. End an active Mac shell session, turn off DriveWire networking, and use a BASIC setup with only the floppy controller if the status report calls for it. Handle any save or export prompt before ejecting your media.

Applying a proposal updates an editor draft. Use the workspace's **Save** command when you want to keep that change on disk. A grouped project edit has a grouped Undo while the applied revision is current. If you have edited either file since, inspect both current drafts before deciding what to restore.

What to ask for

The table describes current tools. Each workspace and saved grant exposes its own subset; discover that subset before asking for an operation.

Task	Tools and useful limits
Identify the workspace and machine	`get_capabilities` identifies the shared draft or project. `get_machine_status` reports the emulated processor, devices, build status and run blockers.
Look up a CoCo detail	`search_help` returns matching installed manual sections. Give `read_topic` a topic ID returned by the search. If the result is marked truncated, open that section in **Help → Manual Library...**.
Read source	`read_source` reads a BASIC draft. `read_project_file` reads a named shared C or assembly file. Open editor drafts can include unsaved changes.
Propose a change	BASIC has `propose_edit` for one existing numbered line and `propose_listing` for a complete numbered ASCII listing up to 12 KiB. C and assembly have `propose_fix` for a line and `propose_project_edits` for complete replacements of one to eight existing shared files, up to 16 KiB in total. A C line repair needs a matching compiler diagnostic.
Check the review or build	`get_proposal` retrieves the native review decision. In C and assembly, `build_project` requests a build of a revision and `get_build` reads its status and diagnostics.
Refresh a C project's inputs	`read_project_inputs` pages the approved manifest, up to 256 rows per page. `refresh_project` requests adoption or local review; `get_project_refresh` polls that decision. Neither executes the recipe.
Read the build failure	`get_build` reports diagnostics, first error, status and stage when known. `get_build_log` reads retained combined output in pages up to 32,768 UTF-8 bytes; follow `next_offset` and report truncation.
Deliver declared build files	On a separately approved export connection, `get_build.exportable_outputs` lists exact output identities. `export_build_outputs` selects the original build/artifact/revision and approved destination; poll `get_build_export`, then independently verify delivered lengths and hashes. It does not rebuild or export guest saves.
Play a text game	`get_run` identifies the run you started. `read_screen_text` observes the guest text. `send_keys` accepts one ASCII letter, digit or space per ordinary key request, up to 64 distinct accepted requests per run. Give each intended press a fresh request ID. Reuse the same ID and identical key to retry an uncertain delivery.
Inspect a stop	`get_debug_snapshot` describes a verified stop, available watches and readable ranges. `read_debug_memory` reads 1–256 bytes wholly within one advertised snapshot range. Device I/O is outside those ranges.
Inspect an owned call	When advertised by a fresh C playtest grant, `get_call_context` uses the same run/snapshot IDs for a fixed 64-byte view at actual S. Treat raw words as uninterpreted; accept caller/argument labels only with verified evidence. It does not grant arbitrary stack addresses or shared-run access.

These tools require the C connection's **Allow Disposable Playtests** grant and the corresponding live capability. They operate on the assistant's separate test machine, rather than the personal CoCo used by the earlier lessons.

Task	Owned-playtest tools and useful limits
Start or check preparation	`start_playtest` binds the current project, revision and artifact. `get_run` polls its reserved run. Fresh processor-aware grants accept `MC6809` or `HD6309`; the default is MC6809.

Task	Owned-playtest tools and useful limits
Enter a line or hold a chord	`playtest_input` accepts up to 64 steps, eight distinct physical keys per chord, 1–600 fields per step and 3,600 fields per request. `period` is a physical key; `backspace` and `left` name the same physical key. Empty `keys` releases input.
Test a joystick	Fresh connections accept optional `joysticks` per input step: right/left port, integer X/Y 0–16383 (center 8191) and `fire1`/`fire2`. Omitted ports center and release. `input_trace` records delivery; inspect actual gameplay frames for the result.
Diagnose input	`get_playtest_input` reads owned input without advancing. Shared `get_machine_status.input` distinguishes host preview, last submission and actual core input, including connection/focus availability.
Capture a picture	`capture_playtest` returns the last published framebuffer without advancing. At a paused state, `capture_playtest_graphics` can compare the current register-selected display with a verified declared buffer ID, including a bounded GIME canvas.
Capture sound	`capture_playtest_audio` advances 1–120 fields and returns a PCM16 stereo WAV, then leaves the guest paused. It uses no microphone. Read actual timing and remaining audio allowance.
Record gameplay for a manual	On a video-enabled connection, `start_playtest_video` starts a bounded MP4 clip from an owned playtest. Ordinary input drives the recorded guest fields; waiting between calls adds no footage. `stop_playtest_video` completes the clip, and `read_playtest_video` returns bounded chunks for saving and verifying it. Choose CoCo audio per clip. Mac windows, microphone and host capture cues are excluded. Inspect completion/truncation, field counts, byte count and SHA-256 before including a clip in a manual.
Pause or inspect code	`playtest_debug` offers `pause`, `continue`, instruction `step`, and verified named `stop_at`. Match `stop_kind` to the build's C-line or function-entry evidence, then request a fresh debug snapshot.
Use Home and Data together	Fresh multi-drive grants expose `drive` 0–3 on `exchange_playtest_media`, defaulting to FD0. Pause with the shared controller idle; use listed private-copy IDs and current hashes, optional insert-time protection, and per-drive receipts. Never silently move one writable copy between drives.
Test Save and Load	`restart_playtest` cold-boots with the current mount map and writable copies. `stop_playtest` retains every copy; `list_playtest_disks` finds project-scoped records. Resume by retained ID and exact hash under compatible base-artifact bytes, then use the guest's Load command.

Give each intended mutation a fresh request ID. Reuse the same ID and identical arguments to recover an uncertain delivery; changing its meaning is a conflict. An owned run has 216,000 field attempts, 16 boots and 1,024 mutation requests. Breakpoints can shorten plans, so use actual returned clocks. Keys and joystick buttons are released and axes centered when an input plan finishes. These are upper bounds, not a promise that a game will reach its intended checkpoint. Stop/retention remains available for cleanup.

BASIC also accepts one fixed probe, `PRINT 7` followed by a newline, to verify that the interpreter returned to its prompt. Type your own direct-mode commands at the guest keyboard.

For a manual lookup, start with terms a CoCo book would use.

> Search the installed manuals for INKEY$. Read a matching section and explain what the function returns when nobody has pressed a key.

Close or renew a connection

Use **Revoke Access** in BASIC or C, or **Revoke Connection** in assembly, when you finish sharing. Revocation disables the grant and removes its file. Text already returned to a conversation can remain there. If you want to work together again, open the intended workspace, save a fresh connection, and restart that server in ChatGPT. A different saved path can be registered through **Add to ChatGPT**. Keep the private connection file on your Mac.

Words you will meet

Guest means the emulated CoCo. **Host** means the Mac that runs VCC Plus. A host with Apple silicon can still be running an MC6809 guest.

Grant is the private connection file that authorizes access to a selected workspace. **MCP** stands for Model Context Protocol, the interface ChatGPT uses to call the tools offered by that connection.

Revision identifies a particular version of the editor draft or shared project. A **build** translates a compiled project revision. A **run** identifies one launch on the guest.

A **breakpoint** pauses execution at a chosen location. A **watch** is a named value you want to inspect there. A **snapshot** records the verified stop and the memory ranges the assistant can read from it. On the next debugging pass, ask which run and which stop supplied the value you are comparing.

Start a project you can hand off

Chapter 9 of 9

An idea can fit in one sentence: make a falling-block game, catalogue a disk, or teach the capitals. A project needs a few more decisions. Which CoCo will run it? Which controls must work? What may the assistant change? Where should it look when it does not know an API?

Write those decisions down once. They give your first chat a useful starting point and let another chat continue without reconstructing the whole conversation. You can change them as you learn; keep the changes visible.

Copy the starter, then make it yours

Download the Project Starter folder into your new project. These are editable documents, not a finished program or an approved build recipe. Replace the bracketed fields, remove irrelevant sections, and review the rules before adopting them.

File	What belongs there
`PROJECT_BRIEF.md`	The program you want, target machine, controls, scope and acceptance checks.
`AGENTS.md`	Working rules that the owner has reviewed and adopted.
`START_HERE.md`	A short first request referring to those documents.
`MCP_SETUP.md`	Connection checks and limitations, without private grants or credentials.
`TESTING.md`	The exact program, machine and results actually tested.
`DECISIONS.md`	Measured limitations, approved exceptions and the reasons for them.

Keep the project's public `README.md` about the program: what it does, how to build and run it, its original sources and licenses, and its known limitations. Readers do not need the history of your local folders or the chat that prepared them. The starter folder's README explains the templates; replace it with your program's introduction when appropriate.

Copy an editable first prompt

After saving a connection in VCC Plus's assistant panel, choose **Copy Project Setup Prompt**. Paste it into your project chat and add your completed `START_HERE.md` request. **Edit...** opens the starter text before copying. You can change the language, controls and working rules to suit your project, or choose **Reset to Starter**. Your edits remain while that connection panel stays open, including when you close and reopen the editing sheet. Keep a durable project-specific version in your project documents.

The copied text includes the current connection name, installed helper path and exported connection-file path. It does not include the file's contents. Copying only changes the clipboard. Review the prompt before sending it: the setup permission takes effect when you give it to the assistant.

The permission lets the assistant register, check and restart the named VCC Plus connection through supported setup tools and make read-only requests to verify access. It stays within the permissions you exported. It does not create a new grant, authorize a different build recipe, or give access to unrelated projects. If a required native permission is missing, the assistant should explain the particular choice you need to make in VCC Plus.

The plain-text starter contains the same working rules without your connection details. It is also useful when preparing a project before opening its workspace. **Copy First Prompt** remains the smaller choice for checking just the current editor connection.

Check three kinds of access

A project connection and a documentation connection are not always separate servers. BASIC, C and assembly connections include Manual Library tools. Archive Search has its own read-only connection, exported from the Archive Library's connection panel. Verify the three kinds of access independently:

Scope	Ask for evidence
Project	Read the actual shared title and revision, using the advertised project tools.
Manual Library	Use `search_help`, then `read_topic` on one returned topic.
Archive Library	Search through its advertised tools and retrieve a relevant citation or document.

A configured server, tools visible in this chat and a successful request are three different facts. Record them separately in `MCP_SETUP.md`. If the project works but Archive tools are unavailable, continue project work that does not need the Archive while repairing that connection. Do not conclude that VCC Plus lacks a feature just because a chat has not discovered it.

Keep exported grants outside Git and out of messages. A setup record can say which workspace and permissions were checked without copying private connection files. The connection chapter explains supported reconnection and direct helper checks.

Use the local reference shelf first

The installed manuals describe the environment you are actually using. Start there:

1. Read the relevant VCC Plus BASIC or assembly manual, or the C API reference. Prefer the documented workspace workflow and bundled APIs.
2. For compiler switches, assembler directives and linker options, consult the complete CMOC and LWTools references. They retain the authors' manuals and identify the matching tool versions. The accompanying command-line help records options offered by the bundled tools, including options not covered in the upstream prose.
3. Use Archive Library for historical manuals, magazines and original hardware material. Preserve the publication and page citation.
4. If these sources leave a gap, describe it and consult the original author's website or another appropriate primary source. Record which version the answer covers.

For example, an unfamiliar CMOC switch is a reason to search the bundled compiler manual and help, not immediately change the Makefile to match a web example. An old magazine may describe another ROM, processor or video mode. If the installed API, manual and observed program disagree, record that disagreement and investigate it. None of them should silently disappear from the explanation.

Choose a machine you can prove

Write memory sizes in **KB** and distinguish the guest processor from the Mac's processor. A useful target might read: “CoCo 3, MC6809, 128 KB, keyboard and joystick, disk boot.” A CoCo 2 edition could have its own 64 KB target, colors and acceptance checks. These are different builds or configurations until you prove otherwise.

Record the official installed VCC Plus version and build agreed for the project. Application projects normally treat the emulator as a black box: public MCP tools, the manuals and documented APIs are available; changing the emulator or using its private internals is outside that scope. Name any additional source examples or artwork the owner has approved as references.

A successful 512 KB test does not establish 128 KB compatibility. Keyboard input does not establish joystick support. If the available automated run cannot select the required machine, mark that check pending and use an approved compatible test environment or owner-assisted check. Do not change the requirement to make the test pass.

For a joystick project, the setup prompt and templates ask for separate input acceptance. Discover the current controls, then use owned joystick input for the required port, axes and fire buttons. Include neutral/release intervals and pause/menu behavior; compare delivery traces with actual movement in captured frames. Shared input diagnostics distinguish controller preview, last submission and actual core values. An emulated analog test does not qualify a physical gamepad or high-resolution interface. If a small C reproducer is needed, run the compiled target at the project's optimization level as well as any host tests.

Begin with one runnable slice

For multi-disk software, plan the owned disk-exchange tests as part of the project brief: which disk boots, which drives hold Home and Data, where a one-drive fallback asks to swap, where saves belong, and how wrong, empty, corrupt and write-protected game data should behave. Declare every test image as a reviewed build product. Verify independent writable copies, per-drive save/read activity, save/reload across swaps and explicit retained-set resumption. Match the program's actual drive choices, including nonzero Home drives; a successful mount alone is not an acceptance test.

Ask for the smallest program that proves an important part of the experience. For a game, that might be one object you can move and stop. For a disk utility, it might be listing one known directory without altering the disk. Agree on the expected result before running it.

For a C project, start with standard CMOC functions and the documented libraries. Ordinary application logic belongs in C. If the requested experience seems to require assembly, direct hardware access or drawing outside those APIs, first measure the actual limitation. Put the proposed exception in `DECISIONS.md`: the missing capability, evidence, alternatives, smallest change and checks needed. The owner or named reviewer decides whether to approve it. An assembly project naturally starts with its agreed instruction set and hardware scope; it does not need a separate exception for every assembly instruction.

Use the project's approved build recipe. Test a source revision and its resulting artifact together. Compilation proves that a program was accepted by the tools; playing it, using its controls and checking its output establish its behavior.

Prove which physical code overlay is loaded

A packed game can replace code in the same physical RAM, rather than merely switching between different RAM banks. Discover the installed C connection's `get_capabilities` first: `banked_debug_manifest_versions` includes `2` and `verified_code_overlays` identifies this support. Use the approved Make workflow and configure `debugManifest` to name the build-produced evidence. The project needs its own source and recipe, the installed tools and public MCP interface; it does not need the emulator's private source tree.

Manifest version 2 adds explicit `overlayGroups` and an `overlay` identity on each member module. This structural fragment shows two code owners reusing the same physical page. It is **not a complete manifest or runnable project**: keep the required entry point, binary/disk/linker identities, complete source set, artifact hashes and remaining modules in the real manifest.

```
{
  "version": 2,
  "overlayGroups": [{"id": "stages", "owners": ["stage-a", "stage-b"]}],
  "modules": [
    {
      "id": "stage-a-code",
      "sections": ["stage_a_code"],
      "codeSections": ["stage_a_code"],
      "image": {"path": "build/debug/stage-a.raw", "logicalAddress": 16384},
      "pages": [{"logicalPage": 2, "physicalPage": 30}],
      "overlay": {"group": "stages", "owner": "stage-a"}
    },
```

```
    {
      "id": "stage-b-code",
      "sections": ["stage_b_code"],
      "codeSections": ["stage_b_code"],
      "image": {"path": "build/debug/stage-b.raw", "logicalAddress": 16384},
      "pages": [{"logicalPage": 2, "physicalPage": 30}],
      "overlay": {"group": "stages", "owner": "stage-b"}
    }
  ]
}
```

An owner may contain several modules. **All of that owner's code-bearing modules must be mapped concurrently and every executable guard byte must match the current RAM snapshot.** Exactly one matching owner establishes residency. Zero matches, partially installed code or multiple matching owners withhold that group's source locations, functions and variables. A familiar address or matching first opcode is insufficient. Different groups and ordinary resident modules cannot claim the same physical storage; same-owner modules cannot overlap it.

Keep executable bytes immutable in `codeSections`. Mutable data and BSS belong in separately declared sections/modules with the same group/owner identity. They inherit the uniquely proven code owner rather than being compared with old initial values; their storage must still have the declared current RAM mapping. A data/BSS-only owner cannot prove residency, and BSS needs no invented raw zeros.

Hashes still bind the complete declaration to its retained build, with independent object/relocation verification. Bounds remain explicit: 1 MiB manifest, 32 MiB per compiler `.s` file, 8 MiB per other evidence file and 128 MiB aggregate evidence; at most 64 groups, 256 owners and 256 modules. Group/owner identities are at most 128 UTF-8 bytes and contain no NUL. Declaring an overlay does not relax these checks or grant memory/register edits.

At verified owned stops, inspection reports the current module and `residency_fingerprint`. Source stepping and cached inspection receipts check fresh residency; the previous owner is not retained after code replacement. Test the real loader's A-to-B-to-A transitions, partial/wrong data and required processors using ordinary guest execution. Do not write guest RAM or registers through a debugger to manufacture a successful load or test.

Keep results another person can use

For each completed stage, record the source revision, build options, artifact name, target configuration and observed result. Use writable test copies of media. Obtain an idle-desktop window before automation that moves the user's pointer or types. Use direct MCP captures where supported; a screenshot of the CoCo display is often more useful than the surrounding Mac interface.

When measuring speed, repeat the same action on the same configuration and state the units. A redraw time, emulated cycle count and wall-clock startup time answer different questions. Keep the baseline result and any remaining limits alongside the improved result.

Finish with a reader-facing README, reproducible source, the agreed runnable artifact and useful screenshots. Preserve original attribution and file-specific license notices. Keep credentials, private ROMs, raw personal captures and temporary logs out of the repository. Commit, push, publish or message another chat only within the owner's actual authorization; do not repeatedly ask for a routine action already authorized for the same work.

For exact build delivery, separately approve a local folder through the C Assistant panel's **Allow Build Output Export**. Ask the assistant to export the original declared Release/Debug products, retain their build/artifact identities, and independently compare all receipt hashes. It can retrieve an earlier Release set after building Debug without another build. Keep delivered build bytes separate from retained writable test disks. Deliver an existing build

For a manual demonstration, the emulator header's camera saves the actual CoCo picture using the remembered None/5/10-second screenshot timer. Selecting a timer changes the setting; clicking the camera takes the picture. The separate video control records the CoCo picture and optional generated sound to MP4 when stopped. Screenshots and videos share one remembered Save To folder, Desktop by default. Both exclude System artwork, Mac controls and the desktop. The local shutter and recording-start cues never enter the output.

Capture controls and recovery

A video-enabled MCP connection also lets your assistant record a short gameplay demonstration from its own disposable playtest. Ask it to use normal keyboard or joystick input, record the relevant route or interaction, retrieve the completed MP4 and verify its byte count and SHA-256 before adding it to your manual assets. Choose CoCo audio per clip. Mac windows, microphone and capture feedback sounds are excluded. Check the recording's field count and truncation status so the description matches what the clip actually shows. Waiting between tool calls adds no footage. These tools serve any project; they require no game-specific support in VCC Plus. Inspect live capabilities before requesting a recording.

A local working disk can also become a **Personal Library** entry without uploading or publishing it. For wider distribution, the software creator can deliberately choose **Library Actions → Submit Software…**, review the permission terms and offer the package for editorial inclusion. Include the required notices and obtain permission for every supplied asset; adding a private entry is not permission to redistribute purchased software.

Private imports and public submissions

Before handing the project to another chat, update the brief, connection checks, decisions and tests. Then ask the new chat to read them, verify its own access and continue from the last demonstrated working stage. That is a more reliable starting point than asking it to infer the project from a long conversation.

Publication and artwork notices

Original VCC Plus prose and instructional images © 2026 Brian K. Holman. Original component and example notices remain with their respective material.

Except where otherwise identified, original artwork owned by Brian K. Holman is © 2026 Brian K. Holman. All rights reserved.

Permission is granted to use and redistribute this artwork unchanged with VCC Plus, including modified versions of its GPL-licensed software. Other reproduction, modification or redistribution requires prior written permission, except as permitted by applicable law. Third-party materials remain subject to their respective licenses and rights.

The CoCo 3 and CM-8 monitor presentation in VCC Plus's skeuomorphic System view was inspired by SockMaster's "pic of CoCo 3 and CM-8 monitor," published on CoCopedia. VCC Plus's artwork was designed and rendered independently. The remaining hardware illustrations were independently designed using historical photographs and renderings as references.

Read this book online

CMOC license notice · LWTools COPYING notice