ChipWits
Bet You Can't Build Just One

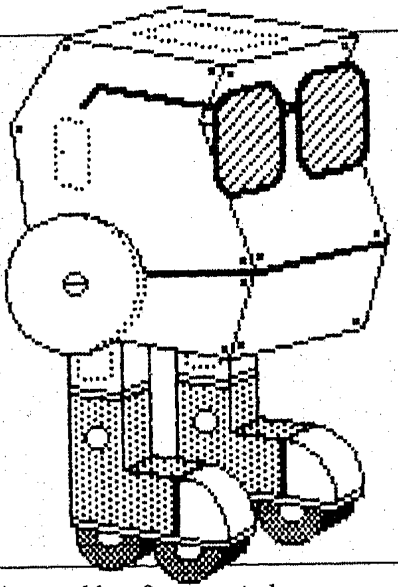
An engaging robot game that teaches the fundamentals of programming

John J. Anderson

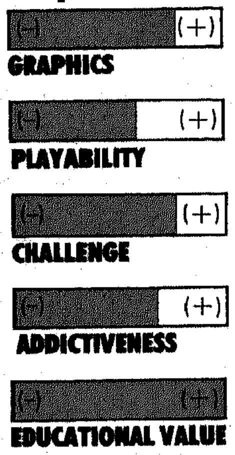
It's getting harder and harder to get me to spend any time playing games on a computer—especially the Mac—lately I'd rather knock about exploring, programming, or online. Don't misunderstand: there are plenty of great games to be found. I've just been preoccupied as of late. Now how many of you are tuned in well enough to guess what I'll say next? You got it: I have found a game that has me hooked. It is called *ChipWits*, which is not to be confused with Chipwiches, those delectable cookie and ice cream snacks, or Chiclets, those little buttons of chewing gum used to make lousy computer keyboards, or Nitwits, those people who think that games can't be educational or that education can't be fun.

It is quite rarely, I'll admit, that a game is great fun to play, while imparting truly valuable experiential learning in the process. The last one that had me hooked was *Mule*, from Electronic Arts (reviewed in the December 1983 issue of *Creative Computing*). *Mule* set standards for the educational microcomputer game—it was whimsical, while firmly rooted in sound principles; it was difficult, but offered a steady learning curve; it was fun, but enlightening—not just junk food for the brain. And most important, the more you played it, the more you understood it's subject matter (in that case, principles of a market economy). We were talking not merely about painless learning. We are talking about *pleasurable* learning.

Enter *ChipWits*. Here we have a world of whimsical robots wearing sunglasses and rollerskates. They make funny noises, snap their jaws, even sing a song now and then. They inhabit maze-like worlds of connected rooms, each filled with an odd assortment of junk. Some of the junk is good junk: there are oilcans and diskettes, which are worth treasure points. There are mugs of coffee and slices of pie, which the robots need to provide the nervous energy to make their rounds. Some of the junk is bad junk: there are crabs and bouncers, which can hurt robots and sap their strength. There are even bombs, which blow our little friends to smithereens. Their little world is a lot like ours, really—they must go for the goodies while avoiding death for as long as possible. There is a philosophical message in here somewhere, I think.



ChipWits



System and Price: C64 disk $29.95; Macintosh disk $39.95
Summary: One of the best combination education/entertainment packages of the year
Manufacturer, Macintosh version: Brainworks, Inc.
24009 Ventura Blvd.
Calabasas, CA 91302
(818) 884-6911
CIRCLE **412** ON READER SERVICE CARD
Commodore version: Epyx
1043 Kiel Court
Sunnyvale, CA 94089
CIRCLE **413** ON READER SERVICE CARD

So, I had just booted the *ChipWits* disk when my colleague, Master Linzmayer, tossed himself cavalierly into the driver's seat. He picked up the joystick. Our diminutive robot friend, named Greedy, remained stock still. "Hey, how the heck do you get this guy to move?"

Answer: you must program him. That's the object of the game. His brain is in your hands. You must build his brain from the ground up. The better the brain you build, the more successful your ChipWit will be. Owen got up—not his cup of tea. He is a twitch gamer at heart, one of the best. I sat down. Now I'm addicted.

IBOLing It

If you have never written a program in your life, *ChipWits* is the place to start. You can begin to program ChipWits in just a few minutes, and you don't even have to know how to read, because ChipWits make use of a rare hieroglyphic language called IBOL, which stands for Icon Based Operating Language. These instructions tell a given robot how to behave, and are represented as icon "chips," which can be arranged in sets of up to 60 on each of a maximum of eight "circuit board" panels.

Figure 1 is an abbreviated list of operators and arguments in IBOL. They represent the sensory inputs, computations, and action outputs of which ChipWits are capable. From the "workshop" mode, you design the "circuit" using the mouse. When you specify an operator, the possible arguments present themselves as a submenu. By pointing and clicking, you construct a program. That program explicitly specifies what your robot can do on the playing field.

Take a look at Figure 2. It is a diagram of the mind of one of my earliest creations, called Mr. Psycho. He is an exercise in economy as well as violence, running in a mere 13 chips. The philosophy of Mr. Psycho is very straightforward: if the road ahead is clear, take it. Otherwise, vaporize. Hence his name. But Mr. Psycho shows some rudimentary intelligence: he does not bump into things. And that's important. Because a ChipWit's short life, sane or not, is fraught with hazards. Bumping into things damages him, and too much damage leaves him dead in the water.

Form Follows Unction

The single most important educational side effect of the game is that it imparts the urge, and leads the way, to systematic thinking. ChipWits must monitor their energy levels, avoid bumping into walls and bad objects, score points, and navigate. Each mission has a fixed number of command cycles associated with it, and each command costs a certain number of those cycles. So you as robot designer are faced with the task of trading off these component considerations, in the attempt to collect the highest score in a given environment.

The neatest way to systematize a robot brain is to use subpanels. These nest from the main panel (they cannot nest within each other) and allow a modular approach to brain architecture. Figure 3 shows part of a movement subpanel for one of my more recent creations, named Lefty, because he makes only left turns. Figure 4 shows part of a ranging subpanel, which allows Lefty to determine when and where there are goodies ahead, and how to get them. Figure 5 shows the main panel of the latest contender I have wrought, Righty-O. As you can see, his main panel is made up entirely of calls to subpanels. Righty-O makes only right turns, but navigates nicely using the loop shown here as Figure 6.

To make matters all the more interesting, *ChipWits* offers eight different environments within which you can test your creations. Each scenario offers its own peculiarities and its own challenges. You start out in Greedville, which is a simple four room environment with no bad objects. You may work your way to the Mystery Matrix, with 100 rooms of good and bad objects, and finally to Boom Town, where the risk of sudden death looms constantly.

Clear Snailing

All new programs need debugging, and ChipWit panels are no exception. Fortunately, there are a number of debugging tools that really help you find and correct the problems with your ChipWits. While running a mission, a special window alongside the main window traces the program graphically through its execution. By selecting the snail icon (see Figure 7) you can slow program execution to a pace at which the trace can be followed by eye. Alternatively, you may choose the footprint icon, to "step" through each command individually with a press of the mouse button. By using these methods, you can quickly isolate the bugs in your panel or subpanel and correct them in the "workshop" mode.

Last night my wife came downstairs at 4:30 a.m. only to witness me cursing at a cartoon robot. I cleared my throat. . .

Up to 16 different robots can be stored to disk with the names you give them (Greedy and a somewhat advanced fellow named Mr. CW are provided on the program disk as starting points for your own programming).

Chip Nits

Not everything is right with *ChipWits*, and there are several improvements that spring to mind for future versions. Although "editing" the panels works the way it should in a desktop metaphor such as the Mac environment, there is no way to select and move parts of circuits on a single panel. This makes the job of polishing a design much tougher. Sometimes you would simply like to pull a single board apart to insert a few new components. Unfortunately, *Chip Wits* is not *MacPaint*. The redesign will entail reconstruction rather than quick alteration.

It would also be nice to be able to step backwards through program execution in the debug window. Many times you see something funny happen, but there is no time to determine just what it was. You may select "snail" mode, but then the phenomenon does not repeat itself for five full minutes. You should be able simply to back up a trace to see what has just occurred, as a good chess program would let you back up through the moves of a game.

There should be an option to create your own environments, as well. This would add to the challenge and the scope of *Chip Wits*, and make it possible to create "test tracks," to evaluate the specific skills of your robots.

The documentation could be about five times better.

A Grown Man, Yet

But these criticisms are due only to my burgeoning addiction to the game. Once you get going with it, *Chip Wits* becomes obsessive—something that is true of all programming. Last night my wife came downstairs at 4:30 a.m. only to witness me cursing at a cartoon robot. I cleared my throat.

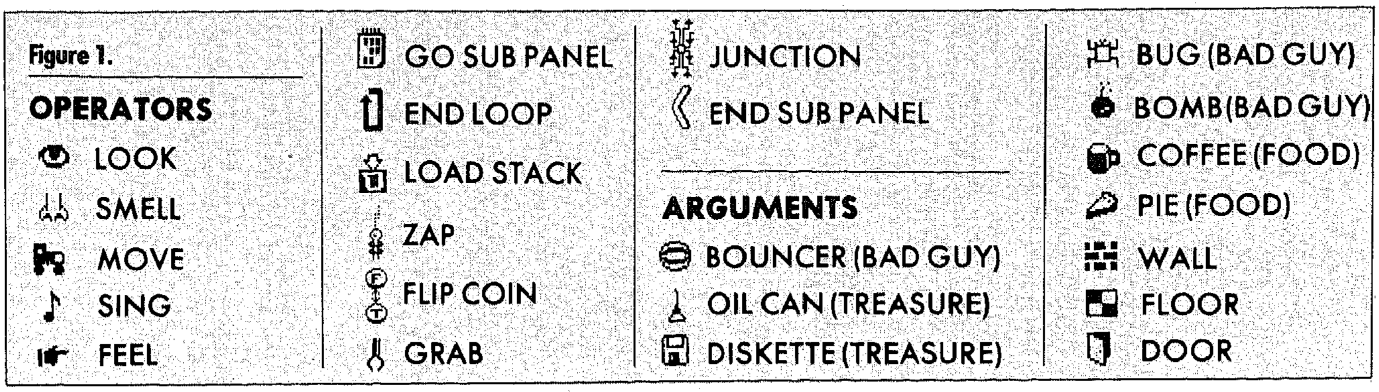
"Righty-O can defend himself, and he gets around pretty well. But he goes really heavy on his cycles, what with those 360° scans he does—32 cycles per pirouette! He's indiscriminate about whether he's picking up food or treasure, and he spends too much time rooting around in nearly empty rooms."

"That's nice. Why do you do this all night when you end up cursing, anyhow?" she asks in a neutral tone.

She just doesn't understand.

We looked at *Chip Wits* in two separate incarnations: one from Brainworks for the Macintosh, and another from Epyx for the Commodore 64. We found the versions roughly equivalent. The Commodore 64 version runs in pseudo-Mac form with windows, pull-down menus, and point/click commands. Its desktop interface is well done, and the color and sound will enhance the appeal for youngsters. The Commodore version is deficient in one way, however: it has a bowdlerized debug mode that is much less helpful than the one that appears in the Macintosh version. ■

Figure 1.



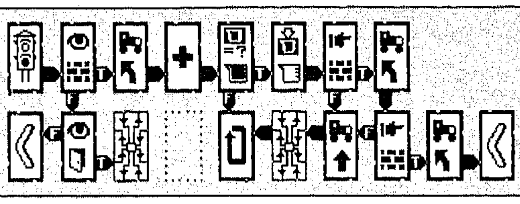


Figure 2. The mind of Mr. Psycho. If it's floor, take a step. If it's a wall, make a left. If it's a door, flip a coin, and either go through the door, or make a left. If it's anything else, kill it.

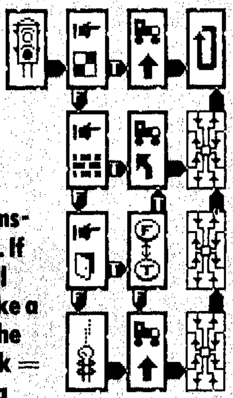


Figure 3. The transmission of Lefty. If you can see wall dead ahead, make a left. Increment the stack. If the stack = 8, you've made a complete pirouette, so clear the stack, and feel for a wall. If you feel a wall, keep turning left. If you have a clear path ahead, take it.

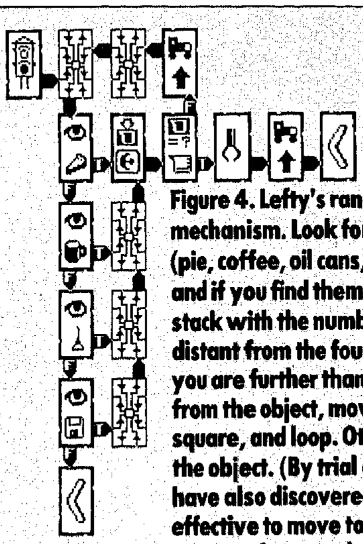


Figure 4. Lefty's ranging mechanism. Look for good things (pie, coffee, oil cans, diskettes) and if you find them, load the stack with the number of squares distant from the found object. If you are further than one square from the object, move ahead one square, and loop. Otherwise grab the object. (By trial and error, I have also discovered it is effective to move to the object square after a grab.)

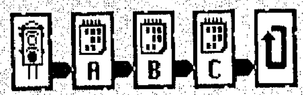


Figure 5. Main panel of Righty-O consists entirely of subpanels. Subpanel A deals specifically with security (are you threatened by a bad guy?). Subpanel B deals with movement (and is similar to the panel in Figure 3). Subpanel C deals with getting good things (and is similar to the panel in Figure 4).

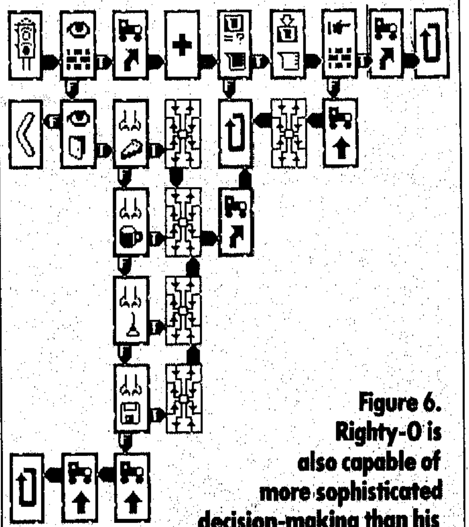


Figure 6. Righty-O is also capable of more sophisticated decision-making than his predecessors. The smell command allows him to choose to leave a room based on whether goodies remain to be claimed there.

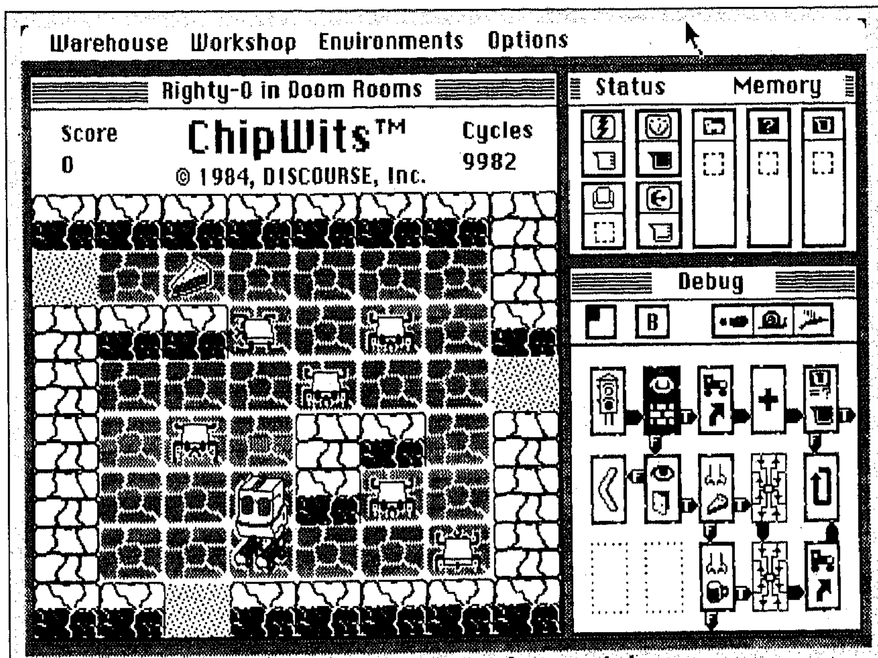


Figure 7. Overview of game board with debug and status windows open.