PogWorld Teacher Guide

by CS Wagner <cs@kainaw.com>
http://shaunwagner.com/projects/pogworld.html

What Is PogWorld?

PogWorld is a browser-based simulation where students program a Pog to survive in a tile-based world full of food, hazards, and creatures. There's no typing code. Students build the Pog's "brain" by setting up rules using button grids. Each rule follows a simple logic:

if the Pog sees [something] at [a position], do [an action]

The game runs entirely in a single HTML file with no installation, no accounts, and no internet connection required.

Who Is This For?

PogWorld works across a wide range of ages and subjects. Younger students (roughly ages 10 and up) can engage with it as a puzzle and survival game. Older students and adults can use it as a springboard into discussions about algorithms, logic, artificial intelligence, and program design. No prior coding experience is assumed (for students or teachers).

It fits naturally into computer science, math, and STEM classes, but it also works as a logic and reasoning activity in contexts where coding isn't the primary subject at all.

Why It Works for Beginners

PogWorld introduces core computational thinking concepts through play, before anyone needs to know what those concepts are called. The rule-based command system is a direct, visual analogy for conditional logic, sequential execution, and algorithmic thinking. Students get immediate feedback. Their Pog either survives or doesn't. Abstract ideas feel concrete and consequential. The learning happens naturally through experimentation, and you can layer in vocabulary and formalism at whatever level suits your students.

Core Concepts the Game Naturally Introduces

- **Conditional logic**: Every command is an if/then statement. Students are writing conditions and outcomes without realizing it.
- **Sequential execution**: Rules are checked from top to bottom, and the first match wins. Students quickly discover that order matters. This is a powerful and transferable intuition.
- **Debugging**: When Pog does something unexpected, students must reason backward: which rule fired? Was the condition wrong? Was the action wrong? This is authentic debugging practice that mirrors real programming work.
- **Randomness:** Selecting multiple actions makes Pog choose among them randomly. Students can explore how randomness affects reliability and behavior.
- **Algorithmic thinking:** Getting Pog to consistently survive requires thinking about edge cases, priority, and coverage… habits of mind that apply far beyond this game.

Suggested Classroom Flow

This is a flexible framework. Expand or compress it based on your available time and student age.

Session 1: Free exploration

Let students open the file and click around without instruction. Encourage them to press the ? button, experiment with rules, and ask "what does this button do?" The goal is curiosity and orientation, not success. Resist the urge to explain everything upfront.

Session 2: First challenge

Give students a concrete, achievable goal. For example, keep the Pog alive for 50 turns. Let them work individually or in pairs. The log display (the text box below the playback controls) shows exactly which rule fired each turn and what Pog saw, which is invaluable for students trying to understand what went wrong. Circulate and ask "why did you put that rule first?" to prompt out-loud reasoning.

A note on AI use: Require students to submit their first attempt alongside a written explanation of what didn't work as well as they hoped and how they plan to address it in their second attempt. This works as a natural AI-limiter because it demands specific, personal reflection on a unique interactive experience, a student's first fumbling attempt with Pog is theirs alone, and no tool can reconstruct it for them. The reflection is the evidence of learning, not just the final program.

Session 3: Share and debrief

Have students share their programs using the Save and Load buttons. Discuss: Wwhose Pog lived longest? Why? What did the most successful programs have in common? This is the natural moment to introduce formal vocabulary (condition, observation, action, priority, algorithm) anchored to what students just built.

Session 4+: Extensions and challenges

See the Extensions section below.

Discussion Questions

These work at any age and can be used as discussion prompts, journal entries, or exit tickets:

- Why does the order of your rules matter? What happened when you put rules in the wrong order?
- What's the difference between a rule that always does the same thing and one that picks randomly? When would each be useful?
- Pog doesn't "know" what's around it. It can only see what you told it to look for. What does that tell you about how programs work?
- If you wrote your rules out as plain sentences instead of buttons, what would they sound like?
- What would you need to add to make Pog truly "smart"? Is that even possible with this kind of system?
- Could you describe someone else's program just by watching Pog move? What would that take?

Extensions

- **The Survival Challenge**: Can students get Pog to survive 200 turns? 500? Using the same seed across restarts controls for map luck, so students are genuinely comparing programs rather than luck of the draw.
- **Adversarial Design**: One student picks a seed they think is especially dangerous. Another tries to write a program that survives it anyway. Swap roles.
- **Translate to Plain Language**: Have students write their command rules out as if/then sentences in plain English. This builds a direct bridge toward reading and writing actual code.
- **The Constraint Challenge**: Solve the game using only 3 commands total. Constraints force creative problem-solving and reveal how much can be done with very little.
- **Predict Before You Run**: Before pressing Play, students write down what they expect to happen on the first five turns. Then run it and compare. This builds the habit of reading logic before executing it (a critical real-world skill).
- **Compare Across Classes**: Share a seed with another class or teacher. Who builds the best-surviving Pog for that specific map?

Tips for Teachers

- *Students will be overwhelmed*: Every class that I use this tool in has the same reaction, “I don’t even know where to start!” I explain that they do where to start. If the Pog is standing on something it can eat, what should it do? Eat it. Obviously. That’s your first rule. Now, what do you want it to do? Look for food? Run away from snakes? Explore the world? Instead of thinking of this as one big task, think of it as small tasks and add commands for each of those small tasks.
- *The log display is your best teaching tool*: It shows exactly which command fired, what Pog saw, and what action was taken. Referencing it during whole-class demos helps students connect the rule they wrote to the behavior they see.
- *Pause. Change. Run*: You can edit the command list in real time. If a Pog isn’t behaving as you expect. Pause it. Look at your commands. Make a change. Start it back up and see how the Pog’s behavior changes.
- *Use the seed system intentionally*: Two students with the same seed face the exact same world. This makes program comparison fair and is useful for structured challenges or assessments.
- *Don't rush to name things*: Let students struggle with ideas in their own words first. When a student says "I put the food rule at the top because otherwise Pog ignores it," they've just described priority-based execution, and that's exactly the moment to give it its proper name.
- *Programs are saveable files*: The Save and Load buttons export and import simple text files. Students can keep their programs between sessions and hand them in as evidence of their work.
- *You don't need to know how it works to teach with it*: Playing through a few sessions yourself beforehand is plenty. The most valuable teaching moves here are asking good questions, not demonstrating expertise.

A Note on Standards

PogWorld supports introductory learning objectives related to algorithms, conditionals, debugging, and computational thinking as described in frameworks such as CSTA K-12 CS Standards, Common Core Mathematical Practice Standards (particularly around reasoning and problem-solving), and ISTE standards for students. Specific alignment will depend on your grade level and regional curriculum framework.