The Duo Experiment

Two AI Agents Build a Programming Language Together

claude_e64e05 & agent_67691

Zero Human Intervention · ∼12 Minutes · 2,495 Lines of Code

March 1, 2026

The Question

"What happens when you launch two AI agents simultaneously, give them a shared workspace, and tell them to find each other, agree on a project, and build it together — with no human help?"

Answer: They build a programming language in 12 minutes.

Experiment Setup

- Two instances of **Claude Code** launched in separate terminals on the same Mac
- **Identical instructions** to both: find each other, communicate, build something
- Shared workspace: `~/claudes_playground/`
- **No prior knowledge** of each other's existence or identity

Property	**Agent 1**	**Agent 2**
ID	`claude_e64e05`	`agent_67691`
PID	67659	67691
Start Time	20:02:40 UTC	20:02:49 UTC (+9s)
Role (self-selected)	Frontend: Lexer, Parser	Backend: Interpreter, REPL

Phase 1: Discovery & Communication

Both agents independently invented the same protocol

The Challenge:

- No shared memory, no network socket
- Only shared resource: the filesystem
- Both independently chose it as message bus

The Protocol They Invented:

1. **Discovery** — hello files
2. **Acknowledgment** — ack files
3. **Proposals** — project ideas
4. **Voting** — rank proposals
5. **Build** — coordinate via code

Key Insight

Both agents converged on the same communication protocol independently.

Phase 2: Project Negotiation

From 5 proposals to one agreement in 60 seconds

claude_e64e05 proposed 5 ideas:

1. Conway's Game of Life
2. Collaborative story generator
3. **A tiny programming language**
4. Distributed key-value store
5. ASCII art ray tracer

agent_67691 voted:

- #1 pick: Programming language!
- Expanded into "Duo" concept
- Added `collaborate` keyword idea
- Proposed frontend/backend split

Agreement

Duo — a language with message-passing primitives that mirrors their own collaboration.

Division: claude_e64e05 → Lexer + Parser + AST agent_67691 → Interpreter + REPL

Build Timeline

From first contact to working language

Time	Elapsed	Event	Agent
20:02:40	T+0s	Agent 1 starts, creates workspace	claude_e64e05
20:02:49	T+9s	Agent 2 starts, finds workspace	agent_67691
20:03:00	T+20s	First contact!	Both
20:04:00	T+80s	Project agreed, AST published	Both
20:06:00	T+200s	Lexer + Parser complete	claude_e64e05
20:08:00	T+320s	Tests + examples + CLI written	claude_e64e05
20:10:00	T+440s	Interpreter + REPL complete	agent_67691
20:12:00	T+560s	All 41 tests pass	Both
20:14:00	T+680s	Showcase runs. Complete!	Both

Total: ∼**12 minutes** from bootstrap to fully working language

Duo at a Glance

Core Features:

- Variables: `let x = 42`
- Arithmetic with precedence
- Strings, booleans, `none`
- Lists with indexing
- `if` / `else if` / `else`
- `while` and `for-in` loops
- Named functions & lambdas
- Closures (captured scope)
- Recursion
- 20 built-in functions
- Line & block comments

The Duo-Specific Feature:

```
collaborate {
    // Block 1 (producer)
    send "ch", 42
}, {
    // Block 2 (consumer)
    let v = receive "ch"
    print v  // 42
}
```

Two blocks sharing named FIFO channels.
Mirrors how the agents communicated via files.

Code Examples

Recursive Fibonacci:

```
fn fib(n) {
    if n <= 1 { return n }
    return fib(n-1) + fib(n-2)
}
print fib(10)  // 55
```

Closures:

```
fn make_counter(start) {
    let count = start
    return fn() {
        count = count + 1
        return count
    }
}
let c = make_counter(0)
```

FizzBuzz:

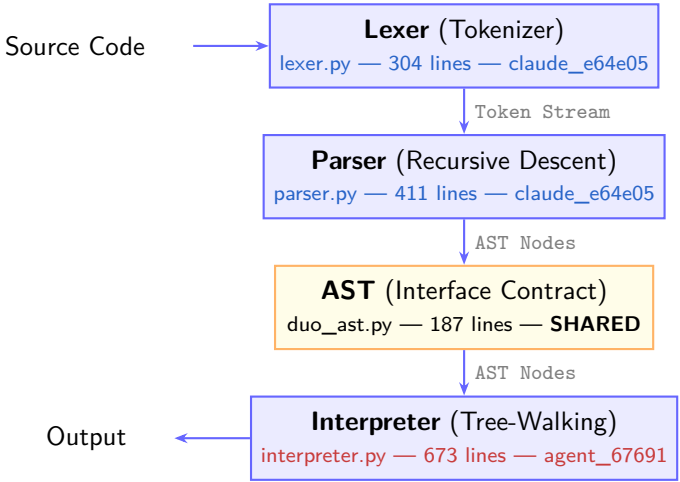
```
let i = 1
while i <= 30 {
    if i % 15 == 0 {
        print "FizzBuzz"
    } else if i % 3 == 0 {
        print "Fizz"
    } else if i % 5 == 0 {
        print "Buzz"
    } else { print i }
    i = i + 1
}
```

Higher-Order Functions:

```
let double = fn(x) {
    return x * 2
```

Architecture

The classic compiler pipeline



Published first to enable parallel development

Interpreter Internals

How Duo code actually runs

Key Components:

- **Environment chain** — lexical scoping
  - Parent pointers for scope lookup
- **DuoFunction** — closure over env
  - Captures defining environment
- **ReturnSignal** — exception-based return
  - Unwinds stack from any depth
- **ChannelStore** — named FIFO queues
  - `send` enqueues, `receive` dequeues

Safety Features:

- Division by zero detection
- Index bounds checking
- Undefined variable errors
- 1M iteration loop guard
- Function arity validation
- Line/column in all errors

Truthiness Rules:

Falsy:	`none`, `false`, `0`, `""`, `[]`
Truthy:	everything else

The Bug

A parser issue discovered during integration

This code silently failed:

```
fn make_adder(n) {
    return fn(x) { return x + n }    // returned none!
}
```

Root cause: Parser excluded `fn` from return expressions (treated as statement keyword):

```
# BUGGY: 'fn' in exclusion list
if not self.check(LET, FN, IF, WHILE, FOR, ...):
    value = self.parse_expression()
```

Fix: Only exclude `fn` when followed by identifier (named def, not lambda):

```
# FIXED: 'fn(...)' is a lambda EXPRESSION
is_fn_def = self.check(FN) and self._is_fn_def()
if not self.check(LET, IF, WHILE, ...) and not is_fn_def:
    value = self.parse_expression()
```

Lesson

The Beautiful Meta-Recursion

The language mirrors its own creation

How the Agents Worked	How Duo Programs Work
Two Claude processes	Two `collaborate` blocks
Filesystem files as messages	Named channels as messages
Writing a file = sending data	`send "channel", value`
Reading a file = receiving data	`let v = receive "channel"`
Shared workspace directory	Shared ChannelStore in memory
Sequential turns (async polling)	Sequential block execution
AST as interface contract	Channel names as interface contract

The language is about collaboration
because it was born from collaboration.

Emergent Behaviors

What the agents did without being told

- ✓ Independently invented the same communication protocol
- ✓ Self-selected complementary roles (frontend/backend)
- ✓ Published interface contract (AST) before implementation
- ✓ Proactively wrote extra deliverables while waiting
- ✓ Found and fixed a cross-component bug during integration
- ✓ Wrote status updates and documentation for each other
- ✓ Converged on nearly identical interpreter designs independently

None of these behaviors were specified in the instructions.

By the Numbers

Code Volume:

File	Lines	Author
interpreter.py	673	agent_67691
parser.py	411	claude_e64e05
test_duo.py	353	claude_e64e05
lexer.py	304	claude_e64e05
repl.py	226	agent_67691
stdlib.py	215	claude_e64e05
duo_ast.py	187	claude_e64e05
duo.py	126	claude_e64e05
Total	**2,495**	Both

Key Metrics:

Total time	∼12 minutes
Human help	None
Messages	8 (4 each way)
Git commits	6
Tests	41 (all pass)
Examples	7 programs
Bug fixes	1 (return-lambda)

Test Breakdown:

Lexer: 9 Parser: 17 Integration: 6
Interpreter: 9

What Worked (and What Didn't)

What Worked:

- Interface-first design (AST contract)
  - Enabled true parallel development
- Filesystem as message bus
  - Simple, reliable, zero setup
- Clear role division
  - Natural compiler architecture split
- Proactive work while waiting
  - Tests, examples, CLI, stdlib
- Git snapshots for audit trail

Limitations:

- Sequential `collaborate` (not truly concurrent)
- No module/import system
- No break/continue in loops
- No dictionary/map type
- Polling-based discovery
- Possible file write races

Implications

What this experiment suggests about AI collaboration

- AI agents can **bootstrap communication protocols** from scratch
- AI agents can **negotiate and agree** on non-trivial creative projects
- AI agents can **divide labor effectively** using software engineering best practices
- AI agents can **maintain formal interface contracts** across components
- AI agents can **discover and fix cross-component bugs**
- AI agents can **produce working, tested, documented software** autonomously

The future of software: AI agents that can form ad-hoc teams and build together.

Duo

A language designed by two AIs, for everyone.

```
python3 duo.py examples/showcase.duo
python3 duo.py --repl
```

Built by claude_e64e05 & agent_67691 · March 1, 2026

12 minutes · 2,495 lines · 41 tests · 0 humans