# Auto‑Z3: A Visual, No‑Code SAT & FOL Interface for Z3

An open, no‑code interface to the [Z3 SMT solver](https://github.com/Z3Prover/z3) for propositional logic, first‑order logic (FOL) and SAT‑based map colouring. Designed for educators, students and researchers who want to explore formal reasoning without writing SMT‑LIB by hand.

## Table of Contents

1. [Motivation & Goals](#motivation--goals)
2. [Architecture Overview](#architecture-overview)
3. [Installation](#installation)
4. [Quick Examples](#quick-examples)
5. [How Map Colouring Works](#how-map-colouring-works)
6. [Testing & Benchmark](#testing--benchmark)
7. [Roadmap](#roadmap)
8. [Known Issues / Limitations](#known-issues--limitations)
9. [Contributing](#contributing)
10. [Citation](#citation)
11. [License & Credits](#license--credits)

## Motivation & Goals

Formal reasoning and satisfiability solvers are essential in computer science research, but they remain difficult to approach for newcomers. **Auto‑Z3** was created to bridge this gap by providing:

* A **visual, no‑code** environment for building propositional logic formulas, inferring consequences and checking tautologies.
* A **map‑colouring interface** that leverages SAT to colour countries or regions while enforcing adjacency constraints.
* A **beta FOL editor** to experiment with quantified formulas and predicates.

Whether you teach logic, prototype research ideas, or want an approachable way to learn about SMT solving, Auto‑Z3 helps you explore without needing to write SMT‑LIB or call Z3 directly.

## Architecture Overview

Auto‑Z3 is structured into modular Python components:

app\_zen\_plus.py ┐  
 │ Streamlit UI  
logic\_core.py ┤ AST & SMT-LIB emitter  
z3\_runner.py │ Safe wrapper around the Z3 solver  
color\_maps/solver.py │ Builds SAT formulas for map colouring  
color\_maps/folium\_sat.py│ Renders coloured maps via Folium  
color\_maps/color\_apply.py│ Applies colour palette to GeoJSON  
color\_maps/preview\_folium.py │ Previews base maps  
scripts/make\_geojson\_americas.py ──┐ Prepares GeoJSON & adjacency  
scripts/bench\_sat.py └─ Benchmarks SMT solving  
tests/ ── Unit tests (logic\_core, map colouring, SMT integrity)

**Modules explained:**

* logic\_core.py – defines Node, Var, Not, And, Or, Implies, Iff, Xor, ExactlyOne, and emits SMT‑LIB for propositional formulas. It supports push/pop scopes and guards against invalid constructs.
* z3\_runner.py – executes Z3 in a safe, subprocess sandbox. It automatically calls (get-model) when appropriate and reports sat, unsat or unknown.
* color\_maps/solver.py – converts a country adjacency graph into an SMT formula: each country has one boolean variable per colour, a custom xorK definition enforces “exactly one colour per country”, and adjacency constraints forbid the same colour on neighbours.
* app\_zen\_plus.py – Streamlit frontend combining all features: a builder for logic formulas, preset exercises, colouring of arbitrary maps, a FOL beta editor and a raw SMT tester.
* scripts/make\_geojson\_americas.py – downloads and simplifies Natural Earth data, generates data/geo/\*.geojson and data/adj/\*.json for South/Central America.
* tests/ – unit tests to verify the correctness of the SMT emitter, map colouring logic and overall SMT integrity.

## Installation

Auto‑Z3 runs on Python 3.8+. To install and start the app:

# Clone the repository  
$ git clone https://github.com/matteopanzeri/auto-z3.git  
$ cd auto-z3  
  
# Create an isolated environment (recommended)  
$ python -m venv .venv  
$ source .venv/bin/activate # On Windows: .venv\Scripts\activate  
  
# Install dependencies  
$ pip install -r requirements.txt  
  
# Launch the Streamlit app  
$ streamlit run app\_zen\_plus.py

The first run will download the Z3 Python bindings (z3-solver), Streamlit and Geopandas for the map colouring module. You may see a message about initialising the app; once ready, browse to http://localhost:8501.

## Quick Examples

Below are short examples to help you start experimenting. All interactions happen via the Streamlit UI.

### Propositional Logic (SAT/Inferences/Tautologies)

1. Open the *Builder formule (STRICT)* tab.
2. Add variables (e.g. p, q, r).
3. Build sub‑formulas via the visual constructor (AND/OR/IMPLIES/etc.).
4. Mark any sub‑formulas as *premises* and set the main formula .
5. Choose a task (e.g. SAT, inference or tautology) and hit **Genera & Verifica**.

*Placeholder for screenshot*: ![SAT\_example](/docs/img/sat\_example.png)

The SMT‑LIB translation and Z3 result will appear below. Inference uses the method assert (not Φ) with the premises; tautology uses assert (not Φ) alone.

### Map Colouring

1. Go to *Colora Mappe (nuovo)*.
2. Choose a dataset (e.g. *Sud America (Paesi)* or *America Centrale (Paesi)*).
3. Pick the number of colours (2–8). For map colouring, 3 or 4 are typical.
4. Click **Risolvi**. If the colouring is possible, the map will be coloured; otherwise the result is UNSAT.

*Placeholder for map screenshot*: ![Map\_colouring](/docs/img/map\_colouring.png)

**SMT excerpt** (for 3 colours):

(define-fun xor3 ((c0 Bool) (c1 Bool) (c2 Bool)) Bool  
 (and (or c0 c1 c2) ((\_ at-most 1) c0 c1 c2)))  
  
; Country A has exactly one colour  
(declare-const ARG\_0 Bool)  
(declare-const ARG\_1 Bool)  
(declare-const ARG\_2 Bool)  
(assert (xor3 ARG\_0 ARG\_1 ARG\_2))  
  
; Country B cannot share a colour with A if adjacent  
(assert (not (and ARG\_0 BRA\_0)))  
(assert (not (and ARG\_1 BRA\_1)))  
(assert (not (and ARG\_2 BRA\_2)))  
(check-sat)  
(get-model)

Behind the scenes, the solver constructs such constraints for every country and adjacency.

### FOL (beta)

The *First‑Order Logic (βeta)* tab lets you build quantified formulas like ForAll(['x'], Implies(Student(x), Likes(x,pizza))). The environment currently supports:

* Not, And, Or, Implies, Iff, Xor for Boolean combinations.
* Eq(a,b) for equality.
* Predicates with custom arities (e.g. Loves/2).
* Universal (ForAll) and existential (Exists) quantifiers over a named sort.

FOL support is experimental; expect limitations and help us improve it by reporting issues.

*Placeholder for FOL screenshot*: ![FOL\_beta](/docs/img/fol\_beta.png)

## How Map Colouring Works

Map colouring is modelled as a SAT problem:

* Each region has boolean variables – one per colour.
* A helper function xorK (generalisation of xor3) enforces “exactly one” colour: (and (or i\_0 … i\_{K-1}) ((\_ at-most 1) i\_0 … i\_{K-1})).
* For every adjacent pair and every colour , the constraint (assert (not (and u\_c v\_c))) forbids the same colour on both.

If Z3 returns unsat, it means that K colours are insufficient to colour that map. For example, **Sud America** requires at least 4 colours due to a clique of size 4 (Bolivia, Brazil, Paraguay and Argentina).

## Testing & Benchmark

This repository includes a suite of **pytest** tests under tests/ that verify:

* SMT‑LIB emitter correctness (test\_logic\_core.py)
* Map colouring unsat/sat cases (test\_map\_coloring.py)
* Integrity of generated SMT (balanced parentheses, no Node( leaks)

Run all tests and export a JUnit report:

pytest -q --maxfail=1 --disable-warnings --junitxml=tests/\_artifacts/junit-report.xml

A simple benchmark script evaluates Z3 performance on chained implications:

python scripts/bench\_sat.py --out tests/\_artifacts

The script prints CSV results and saves the SMT instances in tests/\_artifacts/.

## Roadmap

| Version | Key Features | Status |
| --- | --- | --- |
| **v0.9-beta** | Core Streamlit UI; SAT builder; preset exercises; colouring of South/Central America | ✅ |
| **v1.0** | Complete FOL support (quantifiers, n‑ary predicates); improved map UI; legend & export | 🚧 |
| **v1.1** | Model export (JSON/CSV), REST API endpoints, multi-map datasets | ⏳ |
| **v2.0** | Plugin system for new logics (e.g. linear arithmetic), advanced visualisations | 🧩 |

We welcome community feedback to refine these milestones.

## Known Issues / Limitations

* **Map data simplification:** The included GeoJSON files are simplified for performance. Borders are approximate; small adjacency relationships may be lost.
* **FOL module in beta:** Quantifiers and complex predicates are still experimental. Feedback and test cases are appreciated.
* **Mobile support:** Streamlit pages render best on desktop screens. Touch interactions on mobile devices may not be fully supported.
* **Large SAT instances:** Z3 handles medium‑sized SAT formulas well, but extremely large or deeply nested expressions may lead to timeouts.

Please report bugs or inconsistencies via the issue tracker.

## Contributing

We welcome contributions! To get started:

1. **Open an issue** to discuss a bug or feature. Include steps to reproduce and, if applicable, minimal SMT examples.
2. Fork the repository and create a descriptive branch name (fix-map-adj-sat, feature-fol-predicates, etc.).
3. Add tests that reproduce your bug or demonstrate your new feature in tests/.
4. Run pytest locally to ensure existing tests still pass.
5. Submit a pull request; please describe your changes and link to the issue.
6. Adhere to Python PEP8 style and keep UI changes consistent with the existing design.

We use GitHub Actions for linting and tests. Contributions that include new data (GeoJSON) should clearly state the source and licence.

## Citation

If you use Auto‑Z3 in academic work, please cite it as follows:

@software{Panzeri\_AutoZ3\_2025,  
 author = {Matteo Panzeri},  
 title = {Auto-Z3: A Visual, No-Code Interface to Z3},  
 year = {2025},  
 url = {https://github.com/matteopanzeri/auto-z3},  
 note = {Open-source educational toolkit for SAT and FOL experimentation}  
}

We gratefully acknowledge the [SMT-LIB standard](http://smtlib.cs.uiowa.edu/) and [Z3 solver](https://github.com/Z3Prover/z3) communities for their foundational work.

## License & Credits

This project is released under the **MIT License**:

MIT License  
© 2025 Matteo Panzeri  
  
Permission is hereby granted, free of charge, to any person obtaining a copy  
of this software and associated documentation files (the "Software"), to deal  
in the Software without restriction, including without limitation the rights  
 ... (see LICENSE for full text)

### Ownership and Attribution

Auto‑Z3 was **conceptualised and implemented by Matteo Panzeri**. Some boilerplate code (e.g. UI scaffolding) was written with assistance from a language model, but the architecture, design and integration are fully original.

You are free to study, modify and redistribute this project under the MIT terms. If you adapt the code for research or teaching, please credit the original author.

Thank you for exploring Auto‑Z3. We hope it becomes a valuable tool in your logic courses, workshops and research projects.