Project 2 (CSP – NQueens)

Instructor: *Dr. Amrinder Arora*
Course: Artificial Intelligence
Course-Code: 202501 - CSCI_6511_ADA
GWID: G33160048

Overview

The N-Queens puzzle requires placing `n` queens on a `nxn` chessboard so that:

- No queens are in the same column and diagonal

In this project, we solve N-Queens as a Constraint Satisfaction Problem (CSP). Our solution uses:

- **Search Algorithm:** A backtracking approach that incrementally assigns each row a column.
- **Heuristics:**
  1. Minimum Remaining Values (MRV) to pick which row to assign next.
  2. Least Constraining Value (LCV) to pick the column for that row that eliminates the fewest possible columns in other rows.
  3. A Tie-Breaking rule for rows with the same domain size (example, picking the one with more neighbors).
- **Constraint Propagation:** AC3 to prune inconsistent column choices from row domains.

File Structure

File Structure

main.py

- Parses n from the command line
- Builds domains & neighbors
- Runs AC3 and backtracking
- Prints solution or "no solution"

ac3.py

- AC3 algorithm (revise function)
- Checks for domain conflicts (same column/diagonal)

heuristics.py

- MRV row selection
- LCV ordering of columns

backtracking.py

- Backtracking with domain updates
- Calls AC3 at each assignment

nqueens_csp.py

- Coordinates the integration of AC3 and backtracking algorithms.
- print_solution for final output.

test_nqueens.py

- Simple unittest for correctness checks

Algorithm Summary

Backtracking: tries columns per row, backtracks on conflicts
MRV: picks the row with the fewest valid columns left
LCV: picks the column that leaves the most options available for the remaining rows
AC3: prunes domain values that can't be consistent with neighbor rows

Run Program

```
python main.py 10
```

Run Tests

```
python -m unittest test_nqueens.py
```