Minesweeper AI Solver

Constraint Satisfaction and Probabilistic Inference

Implementation available at:
`https://github.com/artzhuravel/minesweeper-ai-solver`

1 Problem Definition

Minesweeper is a grid-based, partially observable puzzle: cells hide mines, and the player learns only by revealing cells. A safe reveal shows a number 0–8 (mines in its 8-neighborhood); 0-cells typically trigger flood fill that reveals the connected 0-region and its boundary numbers. This project builds an AI agent that plays Minesweeper end-to-end, from the first move on a fresh board until termination in a win or a loss. The core difficulty is incomplete information: the mine configuration is hidden, so decisions must be made from the visible constraints.

1.1 Initial State and Input

The initial state is an unrevealed $W \times H$ board with M mines. Because an unrestricted first move can lose immediately, mine placement follows one of two rules:

1. *Safe first action rule*: the first revealed cell is guaranteed safe [1].
2. *Safe neighborhood rule*: the first revealed cell and its neighbors are guaranteed safe [1].

All remaining mine locations are sampled uniformly over the other eligible cells.

1.2 Goal State

The goal is to reveal all non-mine cells without ever revealing a mine. Each run ends in one of two terminal states:

- *Win (code* 1*)*: all non-mine cells are revealed.
- *Loss (code* -1*)*: a mine is revealed.

The solver is evaluated on complete games (full action sequences), not just single-step recommendations.

1.3 Obstacles and Sources of Complexity

Minesweeper is difficult because it combines: partial observability; overlapping local constraints that form a coupled constraint satisfaction problem; multiple global boards consistent with the same visible state (logical undecidability of some moves); unavoidable guessing in some states; and rapid growth in scale with board size (e.g., Beginner 9×9 with 10 mines, Intermediate 16×16 with 40 mines, Expert 16×30 with 99 mines).

1.4 Why AI Approaches Fit

The task naturally combines (i) logical constraint solving for forced safe/mine moves, (ii) search over consistent mine assignments to uncover non-local forced variables, and (iii) decision-making under uncertainty via probability-based guessing. Accordingly, my goal is to implement such a solver and evaluate how far it can progress with local inference before requiring more costly global inference or riskier probability-based guesses, and then measure performance on the three standard board settings (Beginner, Intermediate, Expert).

2 Solution Specification

2.1 Minesweeper Implementation

The Python class `Minesweeper` implements the core Minesweeper mechanics described above. The only omitted feature is marking mines on the underlying board; this is not needed for the AI, since it records mine marks in its internal `knowledge` state. After each attempted reveal, `Minesweeper` returns a status code indicating the game outcome: 0 for a non-terminal reveal, 1 for a win, and -1 for a loss, along with a payload dictionary. For 0, the payload includes `"revealed_cells"`, a list of newly revealed triples `(x, y, value_str)`. For -1, the payload includes `"all_mines"` (all mine coordinates) and `"revealed_cells_count"` (how many safe cells were revealed before the loss). For 1, no additional information beyond termination is returned.

To test the class, I used the `play_cli` function, which allows playing the game in the terminal and observing the engine's behavior.

2.2 MinesweeperSolver Implementation

The Python class `MinesweeperSolver` implements the AI solver. It takes a `Minesweeper` instance as an argument to `__init__` and stores it as the instance attribute `game`, which the solver uses to call `reveal`.

2.2.1 State Representation and Frontier Updates

`MinesweeperSolver` maintains an internal representation of the game state that is separate from the underlying `Minesweeper` board.

Knowledge grid. The main state is `knowledge[y][x]`, where each cell is either `None` (unknown), `'M'` (marked mine), or a string `'0'–'8'` (revealed number). After each reveal, `reveal_cell` updates `knowledge` for all cells returned in `payload['revealed_cells']`.

Frontier as a constraint system. To support reasoning, the solver maintains two coupled "frontier" structures that encode the current constraints implied by revealed numbered:

- `revealed_frontier[(x,y)] = [U, k]` where (x, y) is a revealed numbered cell, U is the set of adjacent unknown cells, and k is the number of mines still required among those unknown neighbors. Only the revealed cells that have at least one undetermined neighbor are kept in this frontier.
- `unrevealed_frontier[(ux,uy)] = S` where (ux, uy) is an unknown frontier cell and S is the set of revealed numbered cells that currently constrain it. This frontier only includes the unknown cells that have at least one revealed neighbor.

During DFS and probability calculations such a frontier structure is used as a bipartite graph whose nodes are revealed numbered cells and adjacent unknown cells, with edges indicating which unknown cells appear in which numbered constraints; two revealed numbered cells can be treated as 'connected' if they share at least one adjacent unknown cell; the same idea applies in reverse to the unknown cells stored in `unrevealed_frontier`.

2.2.2 First Move Logic

As already noted, `Minesweeper` supports two mine-generation modes: "the safe first action" rule and the "safe neighborhood" rule. The solver can read which mode is active and chooses its first move accordingly, following the strategy described in [1]:

- Under the safe first action rule, the solver reveals a corner cell. This increases the chance that the first reveal is a `"0"` and therefore increases the chance of triggering a flood fill.
- Under the safe neighborhood rule, the solver reveals a cell near the center of the board, which maximizes the number of safe cells revealed on the first move.

2.2.3 Local Rules Inference

Local inference is implemented with two queues that are populated during frontier updates. The `single_inference_queue` stores revealed numbered cells that become trivial: if a constraint has $k = 0$, all adjacent unknowns are safe; if $k = |U|$, all adjacent unknowns are mines. The `paired_inference_queue` stores candidate pairs of revealed numbered cells whose constraints overlap; these pairs are also generated in `update_frontier` by connecting a revealed constraint to nearby revealed constraints that share at least one adjacent unknown cell.

Single inference repeatedly dequeues a trivial constraint and applies the forced action to all cells in its current unknown-neighbor set. Paired inference compares two overlapping constraints by analyzing their intersection and the cells unique to each constraint: bounds on how many mines can lie in the intersection can sometimes force some "only-in-one" cells to be all safe or all mines.

2.2.4 SAT/DFS Inference

When local rules produce no forced moves, the solver runs a SAT-style DFS inference over the current frontier (implemented in `sat_dfs_infer`). Unrevealed frontier cells are treated as mine/safe variables, and each revealed numbered cell is a constraint on its adjacent variables. The algorithm follows these steps:

1. *Split into connected subgroups.* The solver partitions the constraint system into connected components with `_get_frontier_subgroups`. Two revealed numbered cells are connected if they share at least one adjacent unknown. To control runtime, any component with more than `max_dfs_subgroup_len` revealed constraints is skipped for DFS inference, and its variables are tracked as `extra_variables`.
2. *Apply forced moves.* Given the satisfying assignments, `_identify_forced_cells` marks any variable that is always a mine or always safe. Forced mines are applied via `mark_cell`, and forced safes via `reveal_cell`. If anything is forced, the method returns status 2 so the main loop restarts local inference.
3. *Build global configurations.* If no subgroup yields forced cells, `_get_possible_mine_configurations` combines subgroup configurations into global mine configurations (with pruning by the global mine count). `sat_dfs_infer` then calls `_identify_forced_cells` again on the combined space; if still nothing is forced, it returns the configuration data for probability-based guessing in `guess_with_probabilities`.

2.2.5 Probability-Based Guessing

If SAT/DFS inference returns no forced moves, the solver makes a guess by estimating mine probabilities and revealing the lowest-risk cell (implemented in `guess_with_probabilities`).

When SAT/DFS produces global mine configurations, the solver uses `_get_probabilities_configs` to estimate mine probabilities from those configurations, following the configuration-weighting approach described on page 5 of [2]. When no configuration data is available, it falls back to `_get_probabilities_no_configs`, which estimates risk using local constraint densities around frontier variables.

`guess_with_probabilities` selects the candidate with the minimum estimated mine probability. If the minimum is the aggregate option `"floating_tiles"`, it tries a floating unknown cell that borders the frontier the most (to maximize expected information gain); otherwise a floating tile is chosen at random.

If the guess was successful, the loops starts over by going to local inference.

3 Analysis of Solution

3.1 Testing Methodology

To evaluate the solver's performance, I ran Monte Carlo simulations across standard Minesweeper difficulty settings (Beginner 9×9 with 10 mines, Intermediate 16×16 with 40 mines, Expert 16×30 with 99 mines), using `run_solver_expert_level_analysis` function, and compared results across mine-generation rules. `summarize_inference_mix` was also used to aid the analysis. For the testing, `max_dfs_subgroup_len` parameter was set to `float('inf')`, as under this setting we would expect the highest correct guess probability. For each generation rule, I executed 10000 independent games (runs) per board configuration and computed average metrics from the solver's terminal payload.

3.2 Results

Under the safe first action rule, on the Beginner setting, the win rate is 0.9014. On the Intermediate setting, the win rate drops to 0.7534. On the Expert setting, the win rate decreases to 0.3651. These results are comparable to the best-performing algorithm reported in [1], which achieved 0.879, 0.782, and 0.397 for the Beginner, Intermediate, and Expert settings, respectively.

Under the safe neighborhood rule, on the Beginner setting, the win rate is 0.9596. On the Intermediate setting, the win rate drops to 0.8694. On the Expert setting, the win rate is 0.4931. Again, these results are similar to the best-performing algorithm reported in [1] – 0.964, 0.863, and 0.456 for the Beginner, Intermediate, and Expert levels, respectively.

While the results differ across board settings, some common patterns are:

- $90 - 95\%$ of all inferences were made using `single_infer`, $3 - 6\%$ using `paired_infer`, $0.6 - 1\%$ using `sat_dfs_infer`, and $1 - 2\%$ were guesses. These results are expected, given that the solver always prioritizes `single_infer` as the cheapest method whenever any cell is revealed/marked.
- Only `_get_probabilities_configs` was used in these tests, and it correctly identified safe cells with $0.83 - 0.89$ chance across all levels. `single_infer` forced ≈ 0.4 reveals/marks per call; for `paired_infer` this result was $0.17 - 0.21$; for `sat_dfs_infer` it was $0.47 - 0.51$ across all levels.

To see more data and figures supporting the analysis, please refer to Appendix A.

References

[1] Chang Liu et al. "A solver of single-agent stochastic puzzle: A case study with Minesweeper". In: *Knowledge-Based Systems* 246 (2022), p. 108630. DOI: 10.1016/j.knosys.2022.108630. URL: https://www.sciencedirect.com/science/article/pii/S0950705122002842.

[2] Luke Videckis. *Calculating Mine Probability in Minesweeper*. SlideShare (PDF upload). Accessed 2025-12-17. July 2020. URL: https://www.slideshare.net/slideshow/calculating-mine-probability-in-minesweeper/237009910.

Appendix

Appendix A: Solution Data & Figures

Listing 1: Monte Carlo results (10,000 runs) under "safe_first_action_rule". run_solver_expert_level_analysis was used to run the tests and generate Figures 1–4 below.

```
{'beginner': {'avg_reveal_moves_count': 19.5726,
  'avg_revealed_cells_count': 65.6306,
  'avg_markings_count': 9.1917,
  'avg_max_unrevealed_frontier': 16.1329,
  'avg_max_revealed_frontier': 19.7317,
  'avg_iferred_single_infer_count': 25.8566,
  'avg_attempted_single_infer_count': 67.7182,
  'avg_iferred_paired_infer_count': 2.1848,
  'avg_attempted_paired_infer_count': 9.8474,
  'avg_iferred_dfs_infer_count': 0.3404,
  'avg_attempted_dfs_infer_count': 1.1568,
  'avg_probabilistic_guesses_config_count': 1.0417,
  'avg_probabilistic_guesses_no_config_count': 0.0,
  'win_rate': 0.9014,
  'single_infer_per_attempt': 0.3818264513823463,
  'paired_infer_per_attempt': 0.22186567012612465,
  'dfs_infer_per_attempt': 0.2942600276625173,
  'avg_guesses_total': 1.0417,
  'avg_guesses_failed': 0.0986,
  'guess_failure_rate': 0.09465297110492464},
 'intermediate': {'avg_reveal_moves_count': 70.455,
  'avg_revealed_cells_count': 181.3455,
  'avg_markings_count': 33.3677,
  'avg_max_unrevealed_frontier': 34.5394,
  'avg_max_revealed_frontier': 38.4313,
  'avg_iferred_single_infer_count': 101.313,
  'avg_attempted_single_infer_count': 251.9053,
  'avg_iferred_paired_infer_count': 4.4002,
  'avg_attempted_paired_infer_count': 23.3367,
  'avg_iferred_dfs_infer_count': 0.6867,
  'avg_attempted_dfs_infer_count': 1.9674,
  'avg_probabilistic_guesses_config_count': 1.7502,
  'avg_probabilistic_guesses_no_config_count': 0.0,
  'win_rate': 0.7534,
  'single_infer_per_attempt': 0.4021868535517117,
  'paired_infer_per_attempt': 0.18855279452536133,
  'dfs_infer_per_attempt': 0.34903934126258007,
  'avg_guesses_total': 1.7502,
  'avg_guesses_failed': 0.2466,
  'guess_failure_rate': 0.1408981830647926},
 'expert': {'avg_reveal_moves_count': 132.991,
  'avg_revealed_cells_count': 240.9073,
  'avg_markings_count': 61.4788,
  'avg_max_unrevealed_frontier': 37.3598,
  'avg_max_revealed_frontier': 34.3699,
  'avg_iferred_single_infer_count': 182.0014,
  'avg_attempted_single_infer_count': 462.5964,
  'avg_iferred_paired_infer_count': 13.7523,
  'avg_attempted_paired_infer_count': 83.0272,
  'avg_iferred_dfs_infer_count': 1.8334,
  'avg_attempted_dfs_infer_count': 4.5877,
  'avg_probabilistic_guesses_config_count': 4.0136,
  'avg_probabilistic_guesses_no_config_count': 0.0,
  'win_rate': 0.3651,
  'single_infer_per_attempt': 0.3934345360231943,
  'paired_infer_per_attempt': 0.16563608070608185,
  'dfs_infer_per_attempt': 0.3996338034309131,
  'avg_guesses_total': 4.0136,
  'avg_guesses_failed': 0.6349,
  'guess_failure_rate': 0.1581871636436117}}
```

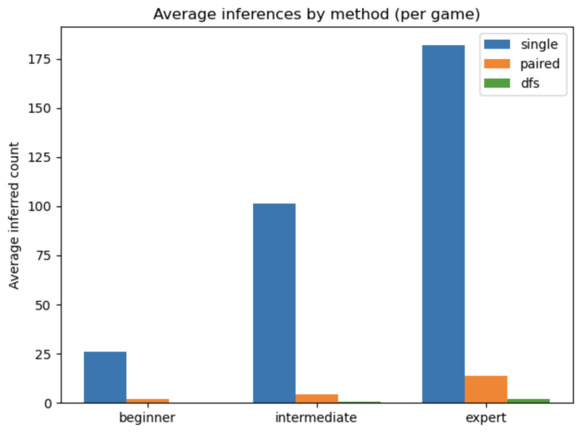

Figure 1: Average number of inferred cell assignments per game under the "safe_first_action_rule", broken down by inference method (single-cell, paired overlap, and DFS/SAT-style), across Beginner, Intermediate, and Expert board settings. For more details, see Listing 1.

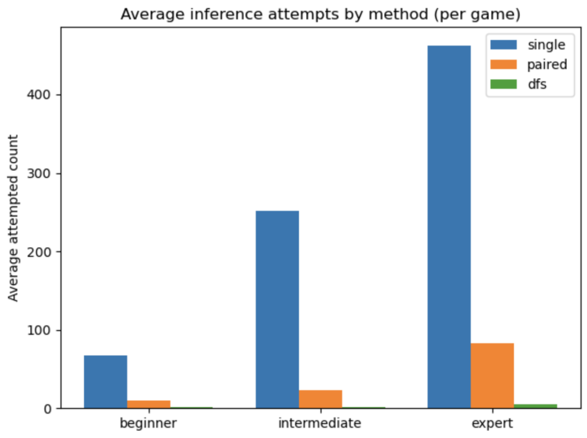

Figure 2: Average number of inference attempts per game under the "safe_first_action_rule", broken down by inference method (single-cell, paired overlap, and DFS/SAT-style), across Beginner, Intermediate, and Expert board settings. For more details, see Listing 1.

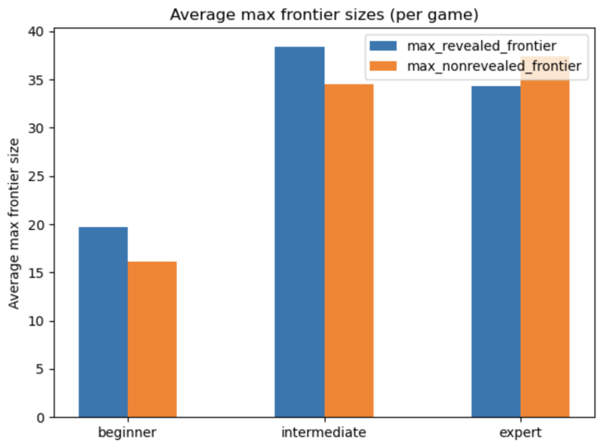

Figure 3: Average maximum frontier sizes per game under the `"safe_first_action_rule"`, showing the peak number of active revealed constraints (`max_revealed_frontier`) and peak number of adjacent unknown frontier variables (`max_unrevealed_frontier`) across Beginner, Intermediate, and Expert board settings. For more details, see Listing 1.

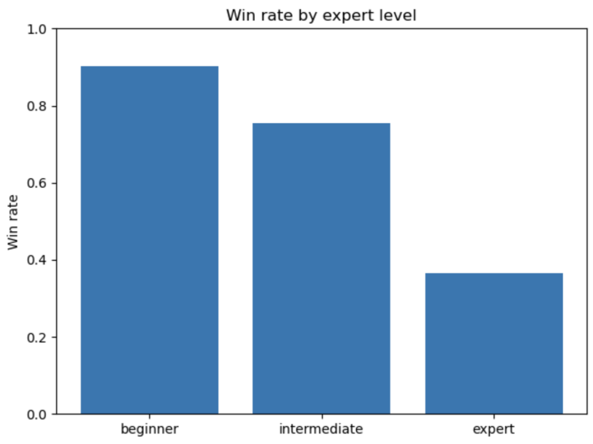

Figure 4: Win rate by difficulty level under the `"safe_first_action_rule"` across 10000 Monte Carlo runs per setting (Beginner 9×9 with 10 mines, Intermediate 16×16 with 40 mines, Expert 16×30 with 99 mines). For more details, see Listing 1.

Listing 2: Monte Carlo results (10,000 runs) under `"safe_neighborhood_rule"`. `run_solver_expert_level_analysis` was used to run the tests and generate Figures 5–8 below.

```
{'beginner': {'avg_reveal_moves_count': 17.8356,
  'avg_revealed_cells_count': 70.8255,
  'avg_markings_count': 9.9186,
  'avg_max_unrevealed_frontier': 18.9416,
  'avg_max_revealed_frontier': 22.3613,
  'avg_iferred_single_infer_count': 27.004,
  'avg_attempted_single_infer_count': 61.0379,
  'avg_iferred_paired_infer_count': 0.7186,
  'avg_attempted_paired_infer_count': 3.9549,
  'avg_iferred_dfs_infer_count': 0.298,
  'avg_attempted_dfs_infer_count': 0.216,
  'avg_probabilistic_guesses_config_count': 0.1357,
  'avg_probabilistic_guesses_no_config_count': 0.0,
  'win_rate': 0.9596,
  'single_infer_per_attempt': 0.44241364791383714,
  'paired_infer_per_attempt': 0.1816986523047359,
  'dfs_infer_per_attempt': 1.3796296296296295,
  'avg_guesses_total': 0.1357,
  'avg_guesses_failed': 0.0404,
  'guess_failure_rate': 0.29771554900515845},
 'intermediate': {'avg_reveal_moves_count': 79.1233,
  'avg_revealed_cells_count': 214.0991,
  'avg_markings_count': 39.4178,
  'avg_max_unrevealed_frontier': 45.3792,
  'avg_max_revealed_frontier': 47.5959,
  'avg_iferred_single_infer_count': 119.3996,
  'avg_attempted_single_infer_count': 284.4659,
  'avg_iferred_paired_infer_count': 2.8304,
  'avg_attempted_paired_infer_count': 14.8354,
  'avg_iferred_dfs_infer_count': 0.6864,
  'avg_attempted_dfs_infer_count': 0.7177,
  'avg_probabilistic_guesses_config_count': 0.5327,
  'avg_probabilistic_guesses_no_config_count': 0.0,
  'win_rate': 0.8694,
  'single_infer_per_attempt': 0.4197325584542822,
  'paired_infer_per_attempt': 0.1907869016002265,
  'dfs_infer_per_attempt': 0.9563884631461613,
  'avg_guesses_total': 0.5327,
  'avg_guesses_failed': 0.1306,
  'guess_failure_rate': 0.24516613478505725},
 'expert': {'avg_reveal_moves_count': 189.4571,
  'avg_revealed_cells_count': 355.5616,
  'avg_markings_count': 90.6778,
  'avg_max_unrevealed_frontier': 60.053,
  'avg_max_revealed_frontier': 53.1926,
  'avg_iferred_single_infer_count': 269.269,
  'avg_attempted_single_infer_count': 664.0054,
  'avg_iferred_paired_infer_count': 16.7343,
  'avg_attempted_paired_infer_count': 94.8963,
  'avg_iferred_dfs_infer_count': 2.1971,
  'avg_attempted_dfs_infer_count': 3.4979,
  'avg_probabilistic_guesses_config_count': 2.8806,
  'avg_probabilistic_guesses_no_config_count': 0.0,
  'win_rate': 0.4931,
  'single_infer_per_attempt': 0.40552230448728277,
  'paired_infer_per_attempt': 0.17634301864245497,
  'dfs_infer_per_attempt': 0.6281197289802453,
  'avg_guesses_total': 2.8806,
  'avg_guesses_failed': 0.5069,
  'guess_failure_rate': 0.17597028396861764}}
```

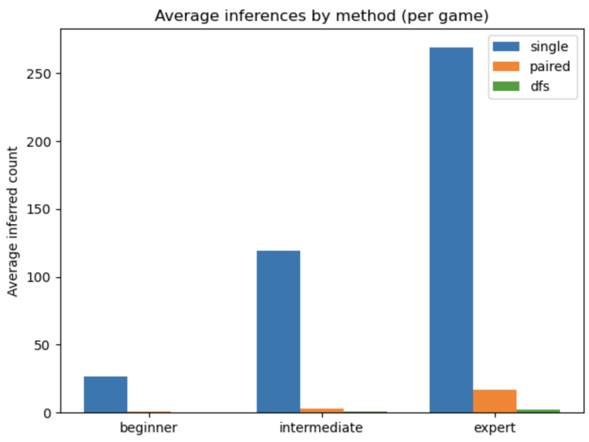

Figure 5: Average number of inferred cell assignments per game under the "`safe_neighborhood_rule`", broken down by inference method (single-cell, paired overlap, and DFS/SAT-style), across Beginner, Intermediate, and Expert board settings. For more details, see Listing 2.

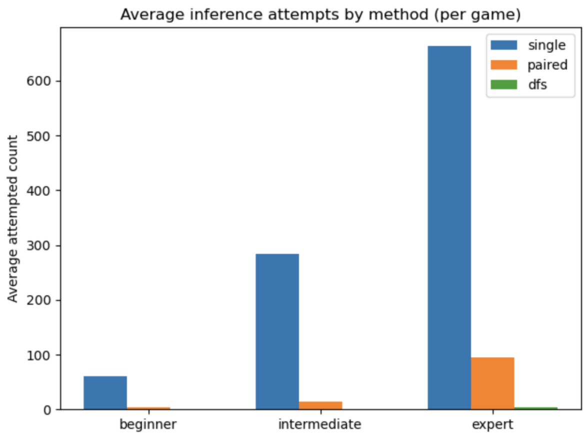

Figure 6: Average number of inference attempts per game under the "safe_neighborhood_rule", broken down by inference method (single-cell, paired overlap, and DFS/SAT-style), across Beginner, Intermediate, and Expert board settings. For more details, see Listing 2.

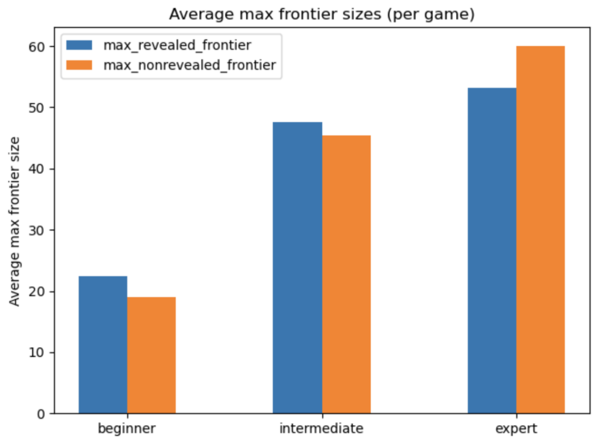

Figure 7: Average maximum frontier sizes per game under the "safe_neighborhood_rule", showing the peak number of active revealed constraints (max_revealed_frontier) and peak number of adjacent unknown frontier variables (max_unrevealed_frontier) across Beginner, Intermediate, and Expert board settings. For more details, see Listing 2.

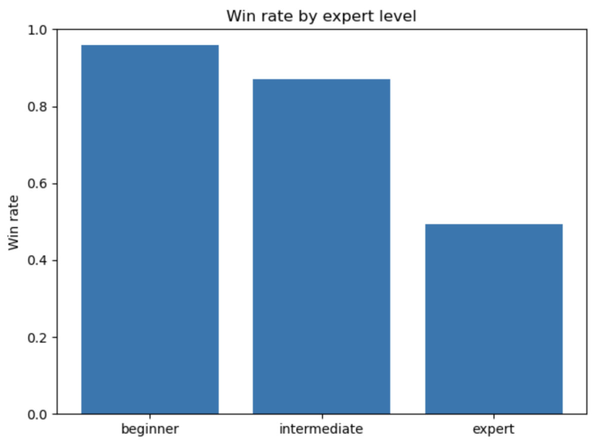

Figure 8: Win rate by difficulty level under the "safe_neighborhood_rule" across 10000 Monte Carlo runs per setting (Beginner 9×9 with 10 mines, Intermediate 16×16 with 40 mines, Expert 16×30 with 99 mines). For more details, see Listing 2.

Listing 3: Composition and effectiveness of solver decisions on the Beginner board (9×9, 10 mines), computed by `summarize_inference_mix` using results from Listings 1 & 2 as input: fraction of forced moves produced by single/paired/DFS inference and probabilistic guessing, with per-call forced-move rates for each inference method and average guess success probability (10000 runs, `safe_first_action_rule`).

```
{'single_frac': 0.9180407017738861, 'paired_frac': 0.05042393339330807, 'dfs_frac': 0.011087221560338869,
    'guess_frac': 0.020448143272467082, 'guess_success_prob': 0.8819432648207916,
    'total_infer_plus_guesses': 28.7899, 'single_forced_per_call': 0.410548315769117,
    'paired_forced_per_call': 0.2103562449736638, 'dfs_forced_per_call': 0.465034965034965}
```

Listing 4: Composition and effectiveness of solver decisions on the Intermediate board (16×16, 40 mines), computed by `summarize_inference_mix` using results from Listings 1 & 2 as input: fraction of forced moves produced by single/paired/DFS inference and probabilistic guessing, with per-call forced-move rates for each inference method and average guess success probability (10000 runs, `safe_first_action_rule`).

```
{'single_frac': 0.9529937927246726, 'paired_frac': 0.031220315096079775, 'dfs_frac': 0.005928776956051661,
    'guess_frac': 0.009857115223195934, 'guess_success_prob': 0.8347715624863112,
    'total_infer_plus_guesses': 115.79960000000001, 'single_forced_per_call': 0.41149226505822833,
    'paired_forced_per_call': 0.18942106931502328, 'dfs_forced_per_call': 0.5113776023239357}
```

Listing 5: Composition and effectiveness of solver decisions on the Expert board (16×30, 99 mines), computed by `summarize_inference_mix` using results from Listings 1 & 2 as input: fraction of forced moves produced by single/paired/DFS inference and probabilistic guessing, with per-call forced-move rates for each inference method and average guess success probability (10000 runs, `safe_first_action_rule`).

```
{'single_frac': 0.9159471520862251, 'paired_frac': 0.06187889665883673, 'dfs_frac': 0.008180738192630252,
    'guess_frac': 0.013993213062307773, 'guess_success_prob': 0.8343825244408343,
    'total_infer_plus_guesses': 246.34085000000002, 'single_forced_per_call': 0.4005589197531905,
    'paired_forced_per_call': 0.17134667427293193, 'dfs_forced_per_call': 0.4984787770851885}
```