Logic validity testing

How to use it, what does it do?

When starting a new `Minesweeper()` session with the following parameters:

```
Minesweeper(expert[0], expert[1], expert[2], csp_on=True,
minecount_demo_number=None, logic_testing_on=True,
unnecessary_guesses=False)
```

which runs an expert game (30x16, 99 mines) with `logic_testing_on=True`, **every lost game is checked for missing logic.** This is done in **constraint_problem_solver_for_testing.py** by utilizing

`python-constraint==1.4.0`

(which is included in 'requirement.txt'), utilizing `Problem` from `constraint`. This essentially brute-forces all possible answers for the situation *right before hitting the mine*; this is done by taking all equations from `self.solver` by accessing `self.solver.unique_equations` (`self.solver` refers to the class `CSP_solver`). After getting all the answers, it checks if a variable was always 0 or always 1. If so, it could have been solved, and "missing logic: 0" will turn to "missing logic:1" in the game.

For convenience: use Intermediate, and press 'i' and 'a' in the game to toggle perpetual playing ('i' again to toggle it off). With expert games, you'll quickly run into situations that are too large for the inspector to inspect (and when you *do* use Expert, if the inspection starts taking long, just quit and start again).

Testing the tester: easy

When you run it with the following settings:

```
Minesweeper(expert[0], expert[1], expert[2], csp_on=True,
minecount_demo_number=None, logic_testing_on=True,
unnecessary_guesses=True)
```

that is, `unnecessary_guesses = True` compared to the previous settings, you can validate the tester itself; now that unnecessary guesses are done, the counter will start rising, as the `check_logic_completeness` verifier will regognize that logic was left unused. Whenever it loses a game, it will stop, and you can look at the situation c:

Note

The tester is slow; it gets all possible combinations of answers for the combined equations in the situation where the game was lost. Because of this, it WILL consume a lot of CPU and RAM in the worst cases (I had a case where the process was auto-killed when the verifier was running, and another one where 9 Gb of RAM was gobbled up before I killed the process).

⇒ Strong recommendation for testing: Intermediate games

For convenience: use Intermediate, and press 'i' and 'a' in the game to toggle perpetual playing ('i' again to toggle it off)

Intermediate games never cause stalling for the inspector `check_logic_completeness`, whereas the longest streak with expert I ever had was 566 games in a row without unbearingly slow inspection process or the inspection being killed. This is simply because the more variables there are, the slower the inspection process, which indeed does get all possible answers (therefore, exponential time complexity regarding input size (=number of variables)).

There should be no limitation for the 'complexity' of the required logic situations itself posed by Intermediate when compared to Expert, since with a large enough number of games, Intermediate is suitably mine-dense enough to eventually provide quite complex scenarios.

Beginner is not a good idea, however, as with 95.9% win rate, it takes forever to lose games compared to Intermediate and Expert.

Results (23.10.2024)

So far, (23.10.2024), according to the results from the verifier `check_logic_completeness` in **constraint_problem_solver_for_testing.py**

- out of 1018 lost Expert games, zero had unused logic
- out of 3424 lost Intermediate games, zero had unused logic

These tables (with also win and loss %) are in **/Testing/Logic validity testing/Logic validity testing.pdf**.

In addition, I ran 100 002 Expert games in one continuous succession with win percentage 38.7%, 148 ms/game average (see **Testing/Win_percentages_and_average_time_per_game.pdf**). This notably higher than 32.90% in Becerra's 100 000 game run (Becerra, David J. 2015. Algorithmic Approaches to Playing Minesweeper. Bachelor's thesis, Harvard College). Likewise, for Beginner and Intermediate I obtained higher win percentages (95.9% and 83.6% respectively). The most probable reason that I can think of is that Becerra doesn't mention minecount situations at all. Without minecount, an equation is missing from the total set of equations, leading to lower win rates.

What if it finds missing logic?

It will stop the perpetual mode (the 'i' mode in botGame), and you can see the the situation for yourself. This is handy when inspecting the inspector itself – otherwise I have not run into such a situation yet.