Minesweeper Solver: Enhancing Performance through Probabilistic Methods

by: Ollie Bland & Shawn Frye

Introduction:

Minesweeper is a classic logic puzzle game that demands strategic thinking and careful analysis. Our objective for this project was to develop an Artificial Intelligence that could effectively navigate and excel at the Minesweeper game, by overcoming the challenges and randomness of the game. This report outlines our journey, the challenges we faced, and the strategies implemented to enhance the AI's performance in Minesweeper.

Development Process:

We started our journey by discovering a basic console-based Minesweeper game through an extensive Google search and an exploration of GitHub repositories. The game served as an excellent foundation, yet it required substantial modifications to be optimally compatible with our AI's gameplay mechanics. Unlike other Minesweeper variations that reshuffle the board to prevent first-move losses, our version maintains the initial configuration, leading to a higher probability of losses based on luck, especially in high-density boards.

Once the AI completes its initial move, the board exhibits a slight opening, where the AI loops through the board looking for definitive safe moves or mines' locations based on corresponding neighbors. In the absence of guaranteed moves, the AI resorts to a probabilistic approach, choosing the cell with the most favorable odds. For instance, if a decision boils down to two options - one adjacent to a "2" and the other to a '4' - the AI will go for the cell next to the '2', given the lower probability of being a mine.

We initially integrated an A* heuristic into our algorithm, intending to enhance performance after the definitive and neighbor checking and before the probabilistic evaluation. However, the integration led to a decline in the winning percentage of the AI, prompting us to omit the heuristic in favor of using solely the probabilistic strategy.

Code:

`Minesweeper` Class:

- `__init__`: Initializes a Minesweeper game with a specified number of rows, columns, and mines, and optionally a callback function.
- `external_move`: Processes a move made externally (such as by the solver), applying a flag or revealing a cell, and updating the game state.
- `generate_board`: Generates the Minesweeper board with mines randomly placed, and numbers indicating nearby mines.
- `display_board`: Displays the current state of the Minesweeper board to the console.
- `play`: Starts a manual game of Minesweeper, allowing a user to make moves until the game is won or lost.

- `reveal`: Reveals a cell and recursively reveals adjacent cells if they are empty.
- `current_board`: Returns a string representation of the current board state.

`Solver` Class:

- `__init__`: Initializes a Minesweeper solver corncerning the Minesweeper game it will solve.
- `playGame`: Plays the Minesweeper game, deciding on and making moves until the game is over.
- `heuristic`: Calculates a heuristic value for a given cell based on its distance to revealed numbers.
- `decide_move`: First determines if it is the opening move of the game, in which case it calls the pick_random_corner method. Then, it loops through the adjacent cells to identify definite moves or mines based on the current board state. If no definite move, the method transitions to a probability-based strategy to make an informed decision.
- `makeMove`: Makes a move in the Minesweeper game by calling `external_move` on the Minesweeper instance.
- `gameCallback`: A callback function that can be used to display the Minesweeper board after each move.
- `make_random_move`: Chooses a random cell to reveal.
- `pick_random_corner`: Chooses a random corner cell to reveal.

`__main__` Block:

- Initializes and plays a specified number of Minesweeper games, size of the board, and mine density using the `Solver` class.
- Calculates and displays the win percentage of the solver.

Challenges and Solutions:

In developing the AI, we encountered several challenges, particularly about the inherent randomness of Minesweeper and the need for the AI to make educated guesses under uncertainty. We addressed these challenges through a combination of strategies, including the implementation of a probabilistic approach to guess-making and the careful analysis of board patterns.

Analysis Factors and Testing:

We decided to conduct tests on our AI's ability to solve Minesweeper puzzles at the different board-level difficulties of beginner, intermediate, and advanced. We based our board-level difficulties on the same standard as other minesweeper games we found online. The beginner difficulty was a 10 by 10 grid with a total of 10 mines. The intermediate difficulty was a 16 by 16 grid with 40 mines. The advanced difficulty was a 30 by 16 grid with a total of 99 mines.

We also looked at whether the AI's starting location would have an impact on the overall game-win percentage. The three starting locations we tested for the AI were the top left corner, a random corner, and any random location. The theory behind the different starting locations was to see if the AI experienced better or worse luck as its first move became less structured and more random.

We performed these tests over a total of 100, 1000, and 10000 games to determine the win percentage of the AI, and we tracked the time it took the AI to play through the different number of games. We conducted a total of 27 different combinations of tests three times each, changing one variable at a time while keeping everything else constant.

Analysis Results:

Minesweeper requires both human and AI players to make the best random or logical guess they can based on limited to no information. The first move of a player or AI is provided with no information on which area of the grid is safe to click making the first move completely a random guess and dependent on luck. Since there is no way to ensure that a player or AI survives their first move, it is impossible to achieve a 100%-win rate in Minesweeper. We tested this theory with our three different starting move locations, which concluded similar results for all three difficulty levels. However, after the first move is discovered to be safe, the AI can use logic to locate potential mines and make smarter decisions on which areas are safe. Our test concluded that our AI was able to solve beginner difficulty boards at 75- 85% success and drop to 30-40% for intermediate-level difficulty boards. Our AI was able to solve advanced difficulty boards but at a low percentage of 1-4%.

Future Improvements:

In the future, to significantly enhance our AI's performance in Minesweeper on larger and densely populated boards, we are planning to integrate advanced pattern-matching techniques. This will allow for a comprehensive analysis of complex board patterns, aiding the AI in solving configurations that are currently challenging, and enabling a more strategic approach.

The implementation of these pattern-matching techniques is hoped to transform the AI’s gameplay, improving its ability to navigate through challenging scenarios, and increasing its overall win rate and efficiency. This is not just about increasing the AI’s performance; it’s about deepening its understanding of the game’s mechanics and strategies.

Also, we are exploring the incorporation of machine learning techniques to enable the AI to learn from past games and adapt its strategies over time. This approach will result in a Minesweeper AI that is both reactive and proactive, capable of anticipating challenges and adjusting its strategies in real time.

Conclusion:

Through research and algorithmic optimizations, we have developed an AI capable of playing Minesweeper at a high level of proficiency. While the game's randomness does introduce an element of luck, our AI has demonstrated the ability to make intelligent decisions and maximize its chances of success. We believe that the insights gained from this project will prove invaluable in future endeavors involving strategic gameplay and AI development.

References

Anand, A. (2020, June 24). *Create minesweeper using python from the basic to advanced*. AskPython. https://www.askpython.com/python/examples/create-minesweeper-using-python

Candra, A., Budiman, M. A., & Pohan, R. I. (2021). Application of A-star algorithm on Pathfinding Game. *Journal of Physics: Conference Series*, *1898*(1), 012047. https://doi.org/10.1088/1742-6596/1898/1/012047