American University of Armenia

Zaven and Sonia Akian College of Science and Engineering

CS246: Artificial Intelligence Project

Solving Minesweeper

Students: Armen Nalbandyan, Mane Koshkaryan, Sona Stepanyan, Anna Minasyan

Instructor: Monika Stepanyan

Fall 2024

Abstract

Minesweeper is a single-player logic puzzle game. The player aims to win by unblocking the board and marking all the bombs on a rectangular form board with flags. Microsoft's official documentation shows that the game was first released in 1990 in Windows 3.1. (Wiki (n.d.)) This paper will discuss and develop various algorithms for solving Minesweeper, including probability measures, backtracking, and depth-first search, and also consider the game an NP-complete problem**.** Additionally, this paper will cover some specific game-focused topics like handling guesses in the game and starting the game most beneficially.

TABLE OF CONTENTS

1 Introduction

1.1 Motivation

For human beings, games have been an integral part of our lives from an early age. In a dictionary, the term game is defined as a physical or mental competition conducted according to rules with the participants in direct opposition to each other. (“Game,” 2024) Over time, parallel to human development and evolution, mathematical ideas were developed, and new applications and theories were created. So, at the beginning of the 20th century, mathematician, physicist, and computer scientist John von Neumann and economist Oskar Morgenstern published a book and formed a new discipline of study named Game Theory, which combines mathematics and games and, as practice showed, has many applications in today's world, including computer science, mathematics, economics, political science, etc. (Neumann & Morgenstern, 1944) In 1994, John C. Harsanyi, John F. Nash Jr., and Reinhard Selten received a Nobel Prize in economics for their research in Game Theory. (Mendelson, 2004) Originally created by Robert Donner and Curt Johnson, Microsoft released the game Minesweeper in 1990 and added it to all Windows releases since then. However, the history of Minesweeper goes back to the early 1980s, when the ancestors of the game, such as "Mined-Out," were released for the Sinclair Spectrum. Since then, the game has become widely known and loved by all. Even though the game has not changed much over time and has stayed virtually unchanged, in the last 2-3 years of the 1980s, specific features, such as the marking of mines and the display of the number of adjacent bombs, were introduced. Besides entertainment, Minesweeper has made crucial progress in the design of logical puzzle games and mobile gaming. Thus, the game was a cultural and academic phenomenon for the world, connecting its historical evolution with the development of AI and

computational programming. Besides having a good puzzle worth solving, Minesweeper also had another purpose for humanity. As technology evolves, people need to have some applications to learn them; Minesweeper was one of those applications whose purpose was to teach people to use clicking devices and separately learn right and left clicking, as the mouse was new then. (Munir, 2022)

Taking the course Artificial Intelligence and Decision Support at the university, we got familiar with search algorithms, types of AI agents, and different types of games that AI agents can solve. Minesweeper was a game we played during childhood, and it was always interesting to us. Now, we have an opportunity to understand and write an agent that will solve the game.

1.2 Description of the problem

Minesweeper is a single-player logic-based puzzle game. During the game, the player is given a rectangular M×N grid with uncovered cells, and the objective of the game is simple: uncover cells without touching the hidden mines. Each cell is either empty or contains a mine behind it. The player loses and the game ends if there is a mine behind the uncovered cell that the player clicked on. The player wins if all the cells have been uncovered except the ones with mines.

1.2.1 Rules of the game

The environment of the grid is non-deterministic. When the board is generated n numbers of mines are randomly placed across the grid. On each step, the player's actions are uncovering the cell or flagging (marking or unmarking the cell). The flag object is used in the game to

indicate the player's suspicion of that cell containing a mine. When the cell is uncovered, two outcomes occur. The first one is a loss if the cell contains mine, and the second one is an empty cell. In this case, the cell indicates a count of mines that are in the neighbor cells vertically, horizontally, and diagonally. Therefore, it ranges between 0 and 8. If there are no adjacent mines behind the cell (number is 0), the grid automatically uncovers all connected empty cells. The performance measure in the Minesweeper is the time taken to solve the puzzle. The game will either end with a win or a failure, therefore, it will always terminate.

2. Essential considerations

2.1 First move

The first move for the Minesweeper game is crucial and deserves a separate consideration. This is the only time during the game when the agent has no information about the grid. As at the beginning of the game, all the cells are covered, the agent needs to choose which square to open first. This guess is inevitable for all types of solvers. Therefore let us see which move is the most beneficial and optimal for all the algorithms to pick.

The perfect choice for any move is the one that maximizes the information about the grid. This is because if we open a square and it shows us a number, we get information about a specific square and its neighbors.

When the first square reveals a number it, of course, puts a constraint on the other grids, but it may be impossible to guess the exact placement of mine(s), so the solver should make further guesses. On the other hand, in the case where the square reveals a 0, we can be sure that the neighboring squares do not contain mines, so they can be uncovered as well, giving more information about the board. Thus, opening a "0" would be the most efficient approach.

Now, let us consider the probability for a square to contain a "0". For this, all of the neighbors of the uncovered square should not contain mine.

$$P(X_0 = 0) = P(X_0 \neq mine) \cdot P(X_1 \neq mine | X_0 \neq mine) \cdot \ldots \cdot P(X_k \neq mine | X_{k-1} = \ldots = x_0 \neq mine)$$

For any square to contain a mine $P(x_0=\text{mine})$, the probability is equal to the number of mines divided by the number of covered cells; let us denote it by d. So, the probability for a cell to not contain a mine is (1 - d). Therefore, the probability for x, having k neighbors, to be 0 is:

$$P(X_0 = 0) = (1 - d)^{k+1}$$

As the corner cells have the least neighbors (3), they are the most likely to contain “0”, being an efficient option for the first move.

Of course, there is a drawback to this approach, as it reveals fewer adjacent cells and makes the constraint limited. In contrast, if we reveal an interior cell (a cell that has 8 neighbors), we gain more information about the grid, which can help us solve the game faster. Anyhow, corners are still statistically safer, so we will stick with this approach.

2.2 Dealing with Uncertainty in 50/50 Situations

In Minesweeper, there are some situations when the agent or the player needs to make a guess, as available clues and logic do not uncover a safe path. A classic example of such a situation is the 50/50 scenario, where two adjacent cells have an equal probability of containing a mine. When this occurs, decision-making becomes challenging, as a wrong move could end the game. Hence, probability theory alone is insufficient, and additional strategies are required to navigate these situations effectively.

The simplest approach for handling guesses is through random selection. This method chooses a covered cell arbitrarily and probes it without any deterministic logic. The advantage of this approach is that it is very simple to implement, and it guarantees that the method will not get stuck. However, the disadvantage of random selection is that it does not consider any knowledge about the board state. In some situations, some cells are more strategic for guessing, which means that random selection tends to perform badly, especially at difficult levels where guessing becomes more frequent.

Hence, in such cases, it is better to make a quick guess, allowing the player to move on to a new game if the guess is wrong and the game ends. By dealing with these scenarios right away, the player will minimize time spent on uncertainty and move to new game processes.

3. Literature review

After examining AI strategies for Minesweeper, we have encountered different approaches and variations in the game. Variations of the game include Quantum Minesweeper, 3D Minesweeper, Hexagonal Minesweeper, etc. Regarding approaches, we have encountered Neural Minesweeper, which uses machine learning algorithms to develop AI agents that can learn patterns and predict mine placements. However, we will consider the traditional version of the game and focus on the following methods for Minesweeper: Depth-First Search (DFS), Constraint Satisfaction Problems (CSP) with backtracking, probabilistic methods, and Conjunctive Normal Form Satisfiability (CNF-SAT).

3.1 Using DFS in Minesweeper

The Depth-First Search method is a stack-based or recursive method to uncover safe cells in the Minesweeper maze step-by-step. It starts from an initial uncovered cell and then examines the neighboring cells in a specific order (mainly clockwise) despite the clue values of the cells. In other words, DFS chooses a path and continues it as deep as it goes. When it hits a dead end, it will backtrack and try another path, again going as deep as it goes. The process repeats until the method either finds a cell next to mine or has covered the entire maze, which means that DFS will reveal all connected safe cells until a cell adjacent to mine is found, meaning no more safe cells can be opened. When the player clicks on a cell with a clue value “0”, meaning that all the adjacent cells are safe to open, Minesweeper automatically reveals them in one step, which is known as “cascade.” For each cell revealed, DFS continues its investigation with an initially determined order. DFS has a structured approach since it searches cells in a predictable order,

making it easy to implement. It is especially good for regions on the grid where there are no mines. The method is useful for Minesweeper as it reveals the maximum number of safe cells with almost no effort. DFS is also memory-efficient, which is suitable for games with constrained memory resources. Finally, DFS uncovers cells without guessing, which can be both considered an advantage and disadvantage. In conclusion, DFS is an effective approach to solve Minesweeper.

3.2 Using CSP backtracking in Minesweeper

The minesweeper can also be considered a constraint satisfaction problem (CSP), where the goal is to make every square of the grid satisfy the constraints. As a criterion of CSP, the Backtracking algorithm is used to find every possible solution to the problem with recursion. The variables are the uncovered squares on the grid. The domain for each variable is {0, 1}, indicating whether there is a mine or not on the square. In the minesweeper, each discovered square forces a constraint, a number in {0, 1, 2, 3, 4, 5, 6, 7, 8}, limiting how many mines exist in the neighborhood. Also, the sum of all the variables must be equal to the number of mines. (Munir, 2022) This method assists in reducing the number of cells that do not contain mines by repeatedly examining various mine arrangements on the grid, ensuring each one follows the given constraint. By deducing this number, it automatically finds the safe cells. It uses a backtracking approach to deal with uncertainties.This method guarantees that the puzzle will be solved without random guesses if there is a path. Since CSP contains constraints, it is unlikely to face errors. The disadvantage is, as it considers all of the possible configurations, the algorithm is very slow, particularly for larger grids.

3.3 Using Probabilities and Logic in Minesweeper

Minesweeper is a game that requires logic, speed, and, unfortunately, some luck in order to be solved. Some cases can be solved during the game using pure logic and deduction. Probability-based strategies can be used in Minesweeper to determine the location of all mines, and if a player played the game long enough, the player probably faced situations where a player can not be 100% sure where the mines are located on the given information. (Nakov & Wei, 2003) An average player will try to guess in these situations or end the game, but with this approach, the probability can give the agent an upper hand. These strategies calculate the probability of whether the uncovered cell contains a mine. The calculations are based on the size of the board, the number of mines on the board, and the information that an agent will receive after the first click. (Fowler & Young, 2004) This can allow players to make informed decisions and minimize risks.

This algorithm starts with analyzing and identifying the information after the first click when the part of the board is revealed. After receiving the information, the agent generates all possible mine arrangements for the following case. After generating all possible arrangements, the agent calculates how many mines are used for the specific arrangement and how many are left. Then, the agent will use combinatorics to calculate the arrangements for the remaining mines to be placed on the board by taking the number of remaining cells after subtracting the number of already opened and unboarded cells and the number of remaining mines. The calculation will be reaped for every possible arrangement the agent has generated. The next step is to combine all arrangements and superpose them into one by summing up the numbers of the corresponding cells. To bring the numbers into presentable form, the agent will divide each cell number by the total number of arrangements and multiply by one hundred.

Using probability and logic in Minesweeper can have both advantages and disadvantages. This method can increase the chances of winning as it helps to make more informed choices to find the mines instead of using random guessing. The method can also allow quick game completion. However, using probability can be computationally expensive, as it requires complex algorithms and a large amount of calculations. There can also be situations where probabilities are equal or when information is not enough, and we would need to use random selection. This method can also be time-consuming as calculating probabilities and considering multiple situations can slow down the AI's decision-making process.

3.4 Minesweeper as an NP-complete problem

To understand a particular concept, first of all, we need to define it. NP-complete problem is any class of computational problems for which no efficient solution algorithm has been found. (The Editors of Encyclopaedia Britannica, 2024) Many significant computer science problems belong to this class, such as the traveling salesman, satisfiability, and graph-covering problems. One type of satisfiability problem is the Conjunctive Normal Form Satisfiability Problem(CNF-SAT), where the Boolean formula is represented in a CNF (Kinber, E., & Smith, C., 2012). Every NP-complete problem is directly connected to game theory, so to map the minesweeper into CNF-SAT, we first need to reduce the logic and constraints of the minesweeper to the logical points. (Nakov & Wei, 2003) Also, it will help us deal with the problem more efficiently. In the game, each cell has a mine or not, and the numbers around the cells give information about adjacent cells and whether there is a mine or not. So, we have two constraints that need to be presented using propositional logic. The first one is each cell's constraint, whether mine or not, and the second one is the relationships between the cells.

To reduce the Minesweeper into CNF-SAT, we must define the variables and propositions and express the constraints as a CNF clause. It is important and necessary to build the CNF clause for each cell. The next step will be defining the binary variable $X_{i,j}$, where *i* and *j* are the coordinates of the particular cell, and $X_{i,j}$ is equal to 1 if there is a mine in a cell and 0 otherwise. Expressing constraints will define the clauses as neighborhood constraints. The first constraint will be: For each cell (*i,j*) with a clue c, define its neighborhood $N_{i,j}$ as the set of cells adjacent to (*i,j*). The second constraint will be: Write a constraint that says exactly *c* cells in $N_{i,j}$ should contain mines.

To express "exactly *c*" in CNF, use the following approach:

1. At least *c* mines: This requires selecting *c* cells among $N_{i,j}$ and setting each subset's mines to satisfy the "at least" condition.
2. At most *c* mines: Use combinations to exclude any configuration with more than *c* mines.

To build this structure for each cell (*i,j*) with a clue *c*, we will convert "exactly *c*" mines constraint into CNF by combining the above-mentioned clauses.

Example:

Assume we have a small 2×2 board:

- Suppose cell (1,1) has a clue "1," meaning there is exactly one mine in neighboring cells.
- If the cell (1,1) has three neighbors, the labels of the neighboring variables will be:

  $x_{1,2}, x_{2,1}, x_{2,2}$

To encode "exactly one mine," use the following clauses:

1. At least 1 mine: $(x_{1,2} \vee x_{2,1} \vee x_{2,2})$
2. At most 1 mine: $(\neg x_{1,2} \vee \neg x_{2,1})$, $(\neg x_{1,2} \vee \neg x_{2,2})$, $(\neg x_{2,1} \vee \neg x_{2,2})$

This results in the CNF:

$$(x_{1,2} \vee x_{2,1} \vee x_{2,2}) \wedge (\neg x_{1,2} \vee \neg x_{2,1}) \wedge (\neg x_{1,2} \vee \neg x_{2,2}) \wedge (\neg x_{2,1} \vee \neg x_{2,2})$$

For a simple example, this CNF formula encodes the Minesweeper constraints. This strategy is systematically applied to each clue cell for larger boards to build the complete CNF formula.

4. Methods

To measure the performance of provided algorithms for Minesweeper, we conducted experiments using DFS and CSP backtracking methods. The algorithms were tested on varying sized grids such as $5 \times 5, 8 \times 8, 9 \times 9$ and larger grids such as $40 \times 40, 50 \times 50 \; and \; 100 \times 100$, which our computer didn't handle. We have tried the game also on a varied number of mines.

For our final testing, we have chosen an 8×8 grid with 10 mines for the CSP method, and a 16x16 grid with 40 mines for the DFS method, simulated 500 runs for both algorithms. We have stored the game results (win/lose), game time (time taken to complete the game) and game steps (in how many steps the game was completed). This stored data helped us to compare the algorithms, and identify the advantages and disadvantages in solving Minesweeper under different conditions.

4.1 The Algorithms

We have implemented DFS and CSP backtracking algorithms in the following way.

The DFS algorithm starts the game at the (0,0) cell (see Section 2.1 for reasoning). It moves from the top left corner to the direction of the right bottom corner. It checks whether that cell contains a mine or not. If yes, it is flagged using the *place_flag* method to ensure it is not revealed again accidentally, if not, it is uncovered and the algorithm continues the process of exploring the grid depth-first. For cells with clue 0, which indicates that there is no adjacent mine, the algorithm recursively uncovers all adjacent cells for any additional 0 cell only. This results in automatically uncovering entire connected regions of 0 cells. Once all 0 cells are uncovered, the algorithm backtracks to the initial 0 cell where DFS started for that safe region

and continues with its adjacent cell in the direction it was initially going. This approach ensures that all safe cells are efficiently uncovered, which reduces the additional computational overhead. The algorithm terminates when all cells are uncovered, resulting in a win. The only case in which the DFS algorithm will fail is when the initial (0,0) cell contains a mine.

The CSP backtracking algorithm again starts the game from cell (0,0). Then, when logical reasoning cannot determine whether a cell is safe or not, the agent invokes the *try_guessing* method to identify unrevealed, unflagged cells as candidates for exploration. The recursive *backtrack_guessing* method then explores each candidate, assuming first that it is a mine and, if that fails, that it is safe. These assumptions are evaluated using forward checking, which propagates the consequences of the assumption to detect contradictions early. Before making an assumption, the agent saves the current game state, including revealed tiles, flagged tiles, and remaining flags, using the *save_state* method. If an assumption leads to a contradiction, such as revealing a mine under a "safe" assumption, the agent restores the saved state using *restore_state* and tries the next possibility. This ensures that incorrect assumptions do not permanently alter the grid. Forward checking evaluates assumptions by propagating their effects and ensuring they are consistent with neighboring tiles. For example, assuming a tile is a mine updates the count of remaining mines for its neighbors. If these updates reveal inconsistencies, the assumption is discarded immediately, preventing wasted exploration. The recursion continues until all candidates are resolved without contradiction (success) or all assumptions fail (failure). By reverting to prior states during failure, the agent systematically explores alternatives until a valid solution is found or all possibilities are exhausted. Hence, the combination of assumptions, forward checking, and state management enables the agent to solve Minesweeper grids

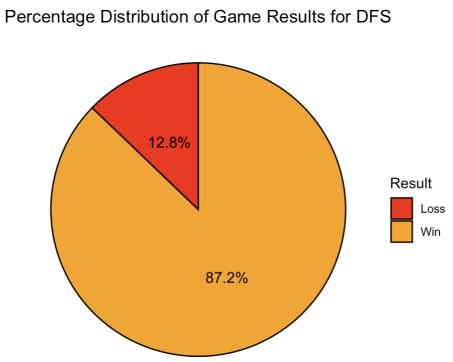
systematically, minimizing errors and handling uncertainty effectively. However, for larger grids, the computational cost of exploring all possibilities can make the process significantly slower.

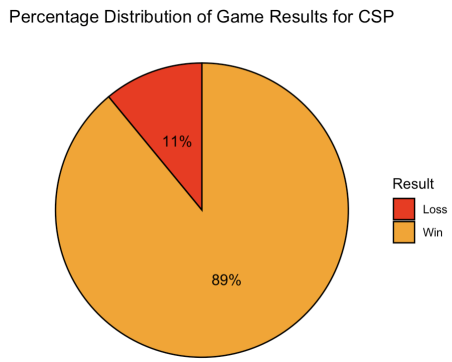
5 Results and Conclusions

5.1 Results

According to the data, on average, it took about 10.78 seconds to solve each game. Out of the 500 games that were implemented, DFS resulted in 436 winning and 64 losing cases. The method used 213 steps on average to complete the game.

Similarly, we tested the CSP method. The dataset gave us the following results. On average, the winning time was approximately 142.7843 seconds. There were 445 winning and 55 losing cases, where the losing because the first opened cell was a mine.





5.2 Conclusions

All things considered, the game Minesweeper can be solved using different AI approaches such as DFS, CSP, CNF-SAT, or Probabilities and Logic.

Depth-First search is simple to implement, is memory-efficient and fast. Average solving time of 10.78 seconds and success rate of 87.2% proves that the method is fast and simple, which is ideal for quick solutions. Unlike more complex algorithms, DFS systematically explores cells, reveals connected areas and ensures a logical solution path.

The CSP, on the other hand, has higher success rates with almost 89% score, but is significantly slower, with an average of 142.78 seconds. This method's strength is that it handles uncertainties through backtracking and constraint propagation but the time compared to DFS is significantly slower.

The Probabilities and Logic approach is the best in making informed guesses. It provides the best possible guess based on available data. But it lacks completeness, meaning that the likelihood measures alone cannot guarantee a correct solution.

CNF-SAT provides the best guess by encoding the Minesweeper game as a logical problem. It is optimal for handling large constraints, yielding globally valid solutions, which means that this method is a good choice for solving complex grids.

References

1. game. (2024). In *Merriam-Webster Dictionary*. Retrieved from https://www.merriam-webster.com/dictionary/game
2. Giannone, P. (n.d.). Click, boom, and traverse: The deep world of Minesweeper's DFS. *Medium*. Retrieved from https://medium.com/@p.giannone333/click-boom-and-traverse-the-deep-world-of-minesweepers-dfs-e9ba42f7e7a9
3. Mendelssohn, E. (2004). *Introducing game theory and its applications*. Taylor & Francis.
4. Munir, R. (2022). *Makalah IF2211 Strategi Algoritma: Analisis dan Studi Penerapan Algoritma dalam Permainan Minesweeper*. Informatika STEI ITB. Retrieved from https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Makalah/Makalah-IF2211-Stima-2022-K2%20(30).pdf
5. Neumann, J., & Morgenstern, O. (1944). *Theory of games and economic behavior*. Princeton University Press.
6. PBS. (n.d.). Nash and game theory. *American Experience*. Retrieved November 13, 2024, from https://www.pbs.org/wgbh/americanexperience/features/nash-game/
7. Wiki, C. T. M. (n.d.). *Microsoft Minesweeper*. *Microsoft Wiki*. Retrieved from https://microsoft.fandom.com/wiki/Microsoft_Minesweeper
8. The Editors of Encyclopaedia Britannica. (2024, November 15). *NP-complete problem | Definition, Examples, & Facts*. Encyclopedia Britannica. https://www.britannica.com/science/NP-complete-problem
9. Nakov, P., & Wei, Z. (2003). Minesweeper,# Minesweeper. Unpublished Manuscript, Available at: http://www. minesweeper. info/articles/Minesweeper (Nakov, Wei). pdf.

10. Fowler, A., & Young, A. (2004). Minesweeper: A statistical and computational analysis.

11. Kinber, E., & Smith, C. (2012). Theory of Computing. Pearson Education India.