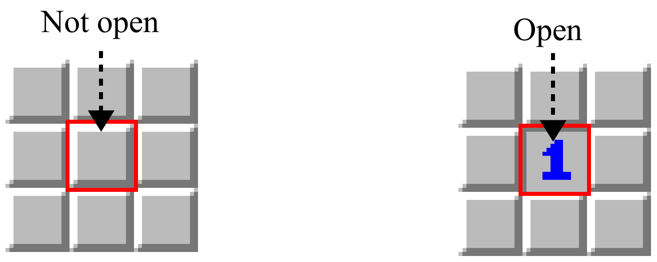
Table of Contents

Game Mechanics

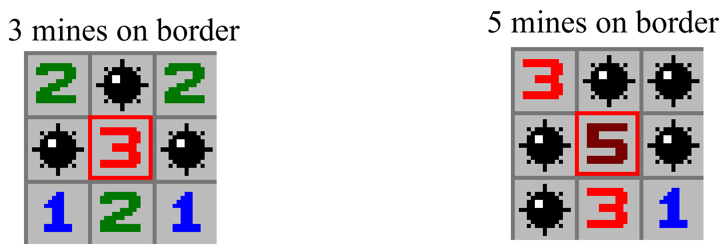
Minesweeper is a game in which the player must correctly deduce the positions of mines in a minefield. The minefield is represented by an *m* by *n* grid. The cells of the grid are referred to as locations and this grid of locations will henceforth be referred to as the board. Each location has two states, open or not open, as depicted in Figure 1.

Figure 1: Not Open and Open



Additionally, each location has a numeric value in the range 0 – 9 associated with it. A value of 9 denotes that the location harbors a mine.* The values 0 – 8 denote the number of mines that are in the location's neighborhood, where a location's neighborhood is defined to be the eight locations that touch it.† It is important to note that a location's value is not known to the player until the player opens that location. However, should the

Figure 2: Three and five mine border configurations



* The use of the number 9 here is simply a convention, any value other than 0 – 8 could have just as easily been used. In the figures a mine icon is used instead of the number 9.

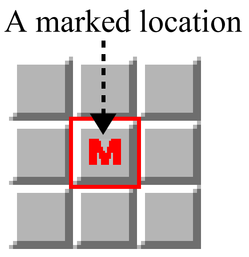
† Note that this is the equivalent of the Moore Neighborhood in Cellular Automata research.

player open a location with value 9, the player is said to have triggered a mine and therefore loses the game. Thus an alternate way to state the goal of the game is to say that the player must open all locations with values in the range 0 – 8 and no locations with value 9.

Game Play

When a new game is created the player is presented with an *m* by *n* board with *b* locations harboring mines. The mines are distributed such that all locations have an equal probability of harboring a mine. Initially none of the locations on the board are open, thus the player knows nothing of the positions of any of the mines. The game is played as a series of steps, at each time step the player must choose one location to open. If a player opens a location that contains a mine then the game is over. However, if the player opens a non-mine location they learn the location's value and can use that location's value to try to deduce the positions of other non-mine locations. At each time step the player also knows the size of the board, the number of mines on the board, and the values of all open locations. Finally, in most implementations, players are permitted to note which not open locations they believe are mines, see Figure 3. This gives the player the opportunity to

Figure 3: Marks



store information on the board that would otherwise have to be recalculated each time a location was encountered. Minesweeper is a game of information, the more information that is available to the player at each time step, the more likely it is that they will be able to find the next non-mine location.

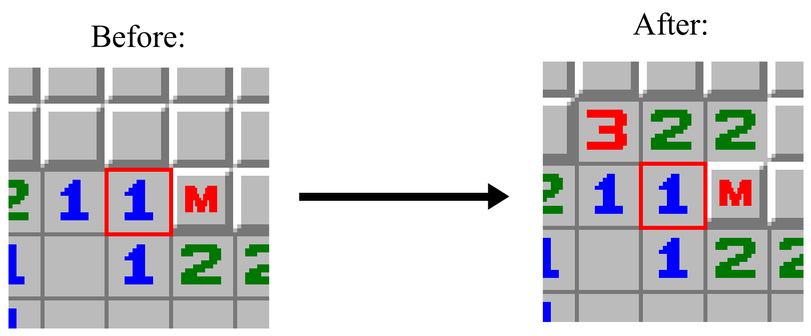
Game Strategy

A typical game of Minesweeper is played on a 16 by 30 board with 99 mines. Thus the probability that a given location contains a mine is roughly .20. Suppose one's strategy was to simply open locations at random on this 16 by 30 board. Then one would have a 79% chance of choosing a non-mine on the first step, 79% on the second, 77% on the fiftieth, and 74% on the hundredth. However, one's chances of opening one hundred, or roughly a quarter, of the non-mine locations consecutively would be roughly one in one hundred billion. Clearly a better strategy is necessary.

Truth

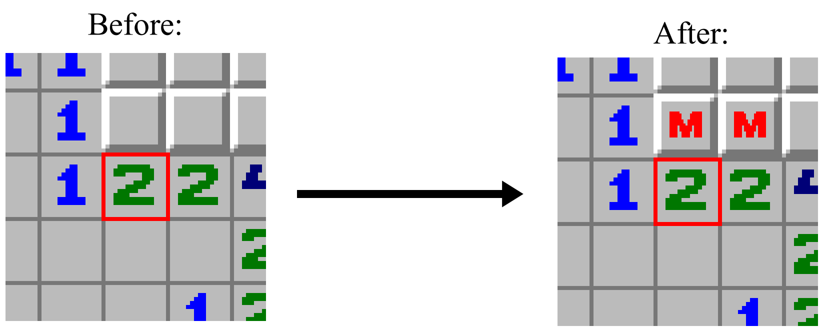
Truth is the most basic strategy that a typical player uses during a game of Minesweeper. It involves analyzing a given location L to determine whether the locations in its neighborhood must be all open, all marks, or neither. More precisely, if L is open

Figure 4: Neighborhood safe to open



then the value *v* of *L* is known and therefore the number of mines in the neighborhood is known. If the number of marked locations in *L*'s neighborhood is equal to *v* then any remaining not open locations can be safely opened, see Figure 4. Alternatively, if *v* minus the number of marks is equal to the number of unknowns then the unknowns have to be mines, see Figure 5. Finally, if neither of these conditions are true then nothing can be determined about *L*'s neighborhood. This result should be obvious to Minesweeper players of almost any aptitude, but it is so fundamental to solving the puzzle that it is by far the most heavily used of the five algorithms. It is also very fast, the algorithm must only examine the eight neighbors of the location *L* to determine what, if any, conclusion can be drawn.

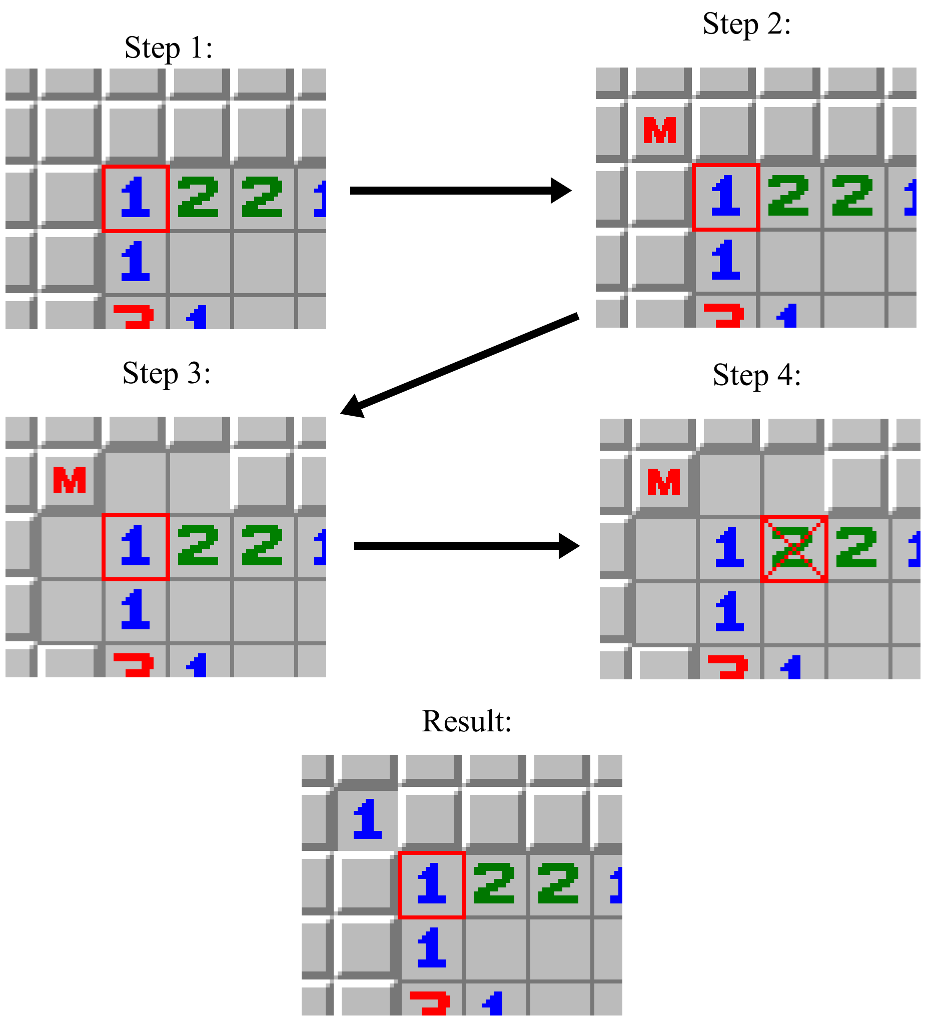
Figure 5: Neighborhood must be mines



Contrapositive

Contrapositive is a slightly more complex algorithm. It is based upon the logical axiom known as modus tollens. Modus tollens states that if *p* implies *q* then not *q* implies not *p*. Accordingly, Contrapositive makes an assumption about the placement of a mark and then checks the implications of this assumption. If the assumption forces the board into a contradiction, such as a location having too many mines, then by modus tollens we know that the assumption was false. More specifically, contradiction chooses a location *L* and places marks in its neighborhood so as to fulfill its mine requirement. It then applies the truth algorithm to *L*, but instead of opening any of *L*'s neighbors it only notes those positions as being open and treats them as if they were really open although their value is not discovered. Then truth is applied in the same way to each neighbor of *L* whose value is known. This process continues recursively until a contradiction is discovered or all neighbors have been exhausted. If a contradiction is discovered then the original assumption must be false, and therefore the location that was marked must be safe to

Figure 6: Find a contradiction

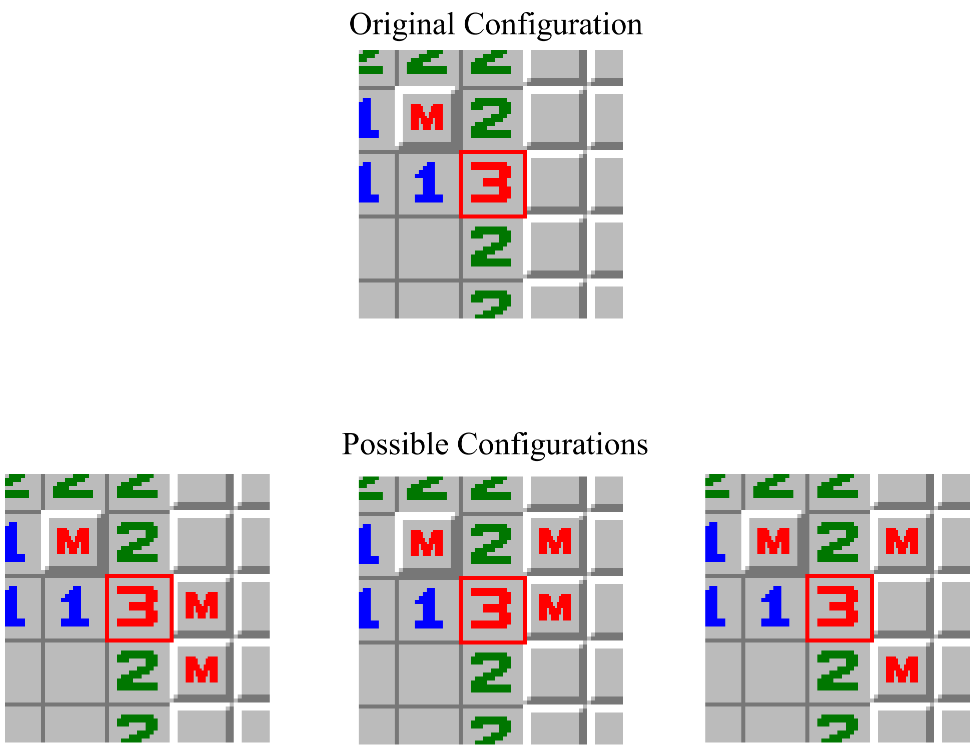


open, see Figure 6. In this process Contrapositive may make up to eight assumptions and execute the Truth algorithm on up to *n* locations for each assumption it makes, where *n* is the number of locations in the board. Unfortunately if the assumption does not result in a contradiction then nothing is learned about the true value of the assumed mark, but the same number of locations must be examined.

Exhaustion

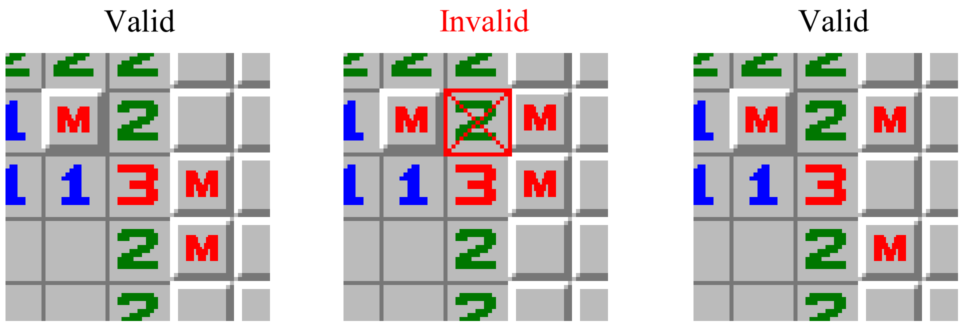
The third algorithm, like the contrapositive algorithm, tests assumptions to try to find out information about the board. Exhaustion works by identifying all valid mark configurations for a given location and examining the neighborhoods for similarities, in other words it exhausts all possibilities. Given a location *L* generate all possible mark configurations that complete the mine requirement for *L*, see Figure 7. For each mark

Figure 7: Enumerate possibilities



configuration *C* apply the truth algorithm to *L* as in the contrapositive algorithm. That is, note any neighboring locations that would be opened, but do not actually open any neighbors. Then apply this technique to each of *L*'s neighbors. Continue recursively until all neighbors have been checked or a contradiction is found, see Figure 8. If no

Figure 8: Check Validity



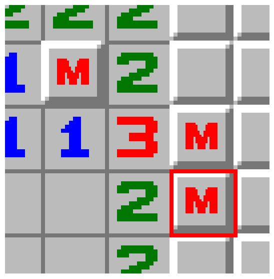
contradiction is found add *C* to the set of valid configurations. When each mark configuration has been analyzed for validity, examine each unknown neighbor *N* of *L*. If the corresponding location in every valid mark configuration is not marked then *N* must be safe to open. Conversely, if in every valid mark configuration a neighbor of *L* is marked, then *N* must be marked, see Figure 9. Like Contrapositive, this algorithm has a running time bounded by the product of the number of assumptions and the number of locations that must be examined. However Exhaustion is much more intensive because the number of assumptions is given by *m* choose *u*, where *m* is equal to the number of mines needed to satisfy *L* and *u* is the number of unknown neighbors of *L*.

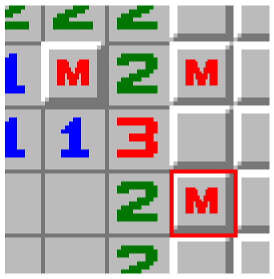
Burnout

Burnout, unlike the first three algorithms, is not targeted at one particular location on the board. Instead burnout tries to find a minimal mark configuration that satisfies all remaining unsatisfied locations. If the minimum number of marks necessary to satisfy these locations is equal to the number of remaining marks then any unknown location

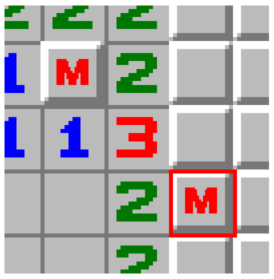
Figure 9: Look for similarities

Valid configurations





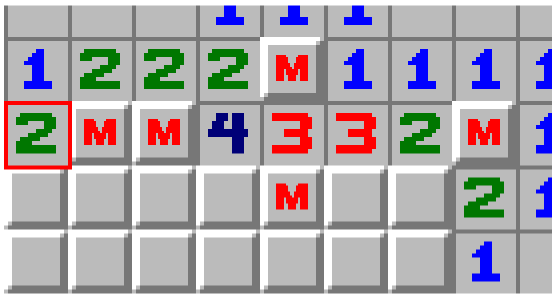
Result



beyond the neighborhoods of these unsatisfied locations most not have any mines. Given a board with unknowns, choose a location *L* that has unknowns in its neighborhood. Define a function *f* to return the minimum number of additional marks needed to satisfy a board. If the parameter to *f* is a board with all locations satisfied *f* returns zero. Then if we consider each valid mark configuration of *L* to be a separate board, the minimum number of mines needed to satisfy the original board is given by *L* minus the number of marks in *L*'s neighborhood plus the minimum over all valid mark configurations *C* of *f*(*C*).

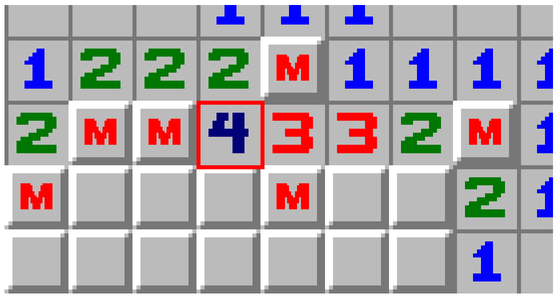
Consider the bottom left corner of a Minesweeper board for which there are only two marks remaining. If the minimum number of mines required to satisfy this board is equal to two then it may be possible to find more locations that are safe to open. To determine the minimum number of additional marks needed, we select a location L that is unsatisfied, Figure 10 represents such a situation (the selected location L is denoted with

Figure 10: Bottom left corner with L *selected*



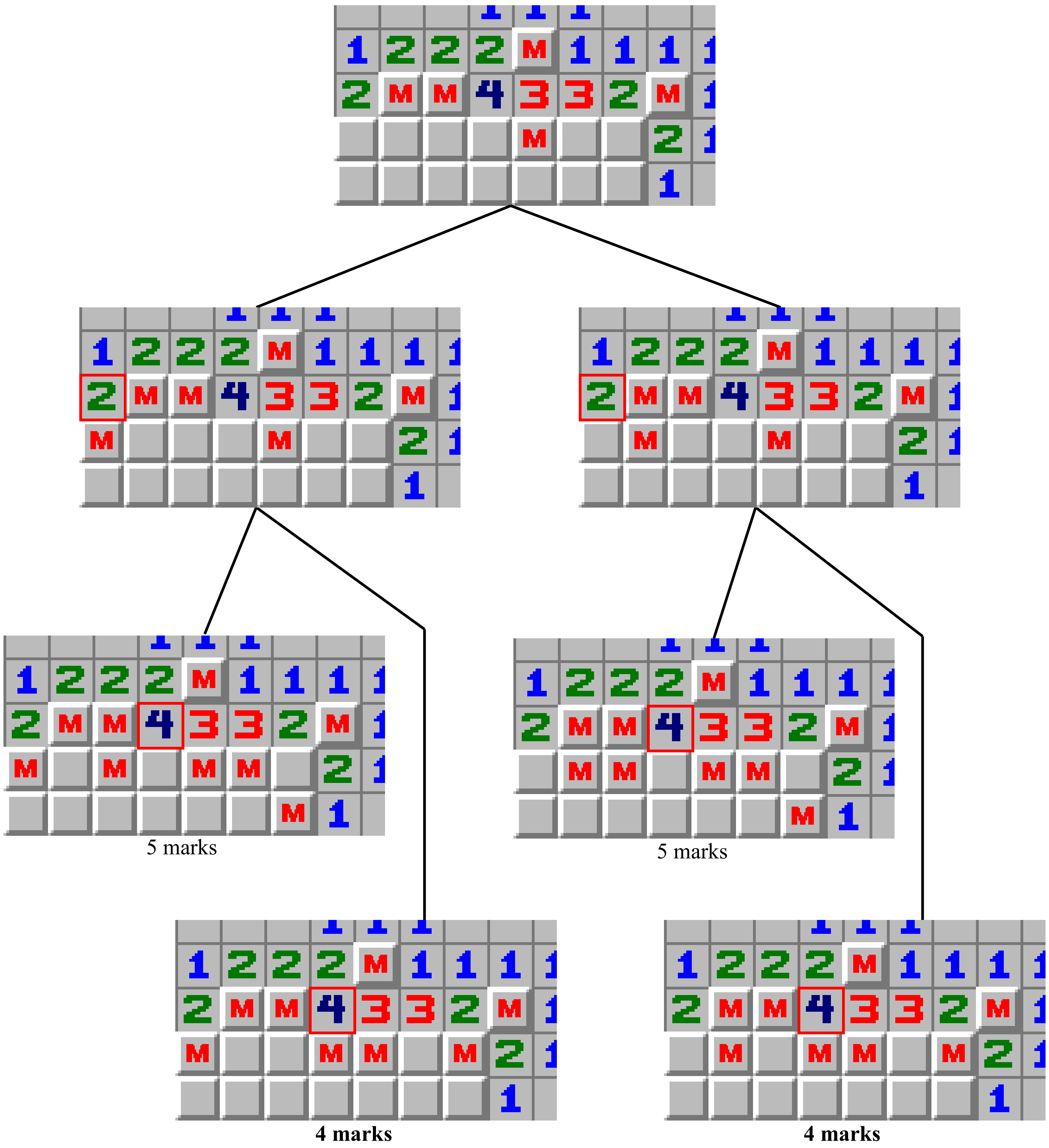
a red border). Begin by choosing a valid mark configuration for L, then find the minimum number of mines needed to satisfy that board by selecting an unsatisfied location L', see Figure 11. Then choose a valid mark configuration for L' and find the minimum number

Figure 11: Bottom left corner with L *satisfied and* L' *selected*



of mines needed to satisfy that board by selecting another unsatisfied location L''. Continue recursing in this manner until no more unsatisfied locations remain. If one of the unsatisfied locations has multiple valid mark configurations try each one before returning. This process builds a tree containing all possible mark configurations for the board, see Figure 12. This is, by far, the most intensive algorithm, in order to generate all possible mark configurations it must enumerate m choose u mark configurations for up to

Figure 12: Mark configuration tree



n locations where m is equal to the number of mines needed to satisfy L and u is the number of unknown neighbors of L and n is the number of locations in the board. As a result this algorithm, unlike the previous three, will typically take a couple of seconds to execute and will sometimes take upwards of ten seconds to execute.

Game Difficulty

Minesweeper is undoubtedly a difficult game to win, but because there are no record keeping facilities built in to the game it is difficult to obtain any sort of meaningful quantification of Minesweeper's difficulty. If a large set of game outcomes could be obtained then a plethora of different metrics could be computed the least of which being the probability of winning a game. But in order to generate such data sets something would have to play hundreds of games of Minesweeper and keep track of the outcomes. A human-like, but fully automated minesweeper player would be ideal for an investigation of this kind.

Such a player has been constructed by utilizing the four strategies detailed above. The automated player, or MineBot, attempts to solve a game of Minesweeper by applying the four strategies in ascending order of complexity. The MineBot, therefore, begins by applying the truth strategy to every location on the board. If, over the entire board, truth fails to reveal at least one safe location or place at least one mark then MineBot considers the strategy to have failed. When a strategy fails MineBot switches to the next least complex strategy which is contrapositive, in this case. If contrapositive fails to yield any progress from any location on the board MineBot will apply exhaustion and if exhaustion fails MineBot will apply burnout. Finally, if burnout fails MineBot simply opens one nonopen location at random. However, if any one of these strategies succeeds in identifying a mark or opening a location then MineBot switches back to the least complex strategy and begins the search anew.

Using my MineBot I performed three experiments: The first investigated the difficulty of an expert game, 99 mines on a 16 by 30 grid, the second investigated how the difficulty changes as the number of mines increases on a board with fixed dimensions, and the third investigated how the shape of the board changes the difficulty of a game when maintaining the same number of mines.

Experiment One

An investigation of the difficulty of solving a standard expert board: 99 mines on a 16 by 30 grid.

Hypothesis

I hypothesized that Minesweeper was a very difficult game to win consistently. I knew from my own experience playing the game that successfully solving the board was difficult and that the difficulty arose from the guesses that Minesweeper players often make. Every player is forced to make at least one guess, and that is choosing the first location to open. When choosing this location one has a 99 in 480, or approximately a 20% chance of failing. Thus no player can possibly win very much more than 20% of the games in a large data set. Also there is usually at least one more guess later in the game that offers a 50% chance of failing. Thus I hypothesized that an expert player should be expected to win roughly 10% of its games.

Method

The experiment to quantify the difficulty of a game of Minesweeper was very simple, I merely set up the MineBot to play 500 expert boards and recorded the outcome of each game. My goal was to obtain a large sample of outcomes from which I could reliably derive the chance of winning a single game by dividing the number of games won by the number of games played, this is called the win rate.

Results

Using the MineBot a win rate of 5.6% was observed on the expert board configuration.

Conclusion

The MineBot's win rate was surprisingly low. This means that either the MineBot is flawed, it is missing a technique, or that my estimated win rate was too optimistic. However the MineBot never loses except when it is making a guess, thus it must not be flawed in a way that causes it to incorrectly identify mines. The only other flaws that can exist involve not utilizing all the information from the board that is available and this is equivalent to the MineBot missing technique. The possibility of the MineBot missing a technique can be tested by allowing a human to play wherever the MineBot would normally have guessed. If the human/MineBot team still fails to achieve a win rate closer to 10% we can assume that the MineBot is at least as good as the human and therefore that the no techniques are missing. Then the only remaining conclusion would be that the estimate was too optimistic.

Experiment Two

An investigation of the change in difficulty as a result of changing mining densities on a fixed board size.

Hypothesis

I hypothesized that increasing the number of mines would increase the difficulty and that as the number of mines increased there would be a point at which the number of wins would suddenly drop off. In other words that, at a certain number of mines, the board becomes much more difficult to solve than it was previously. I refer to this point as the crystallization point.

Method

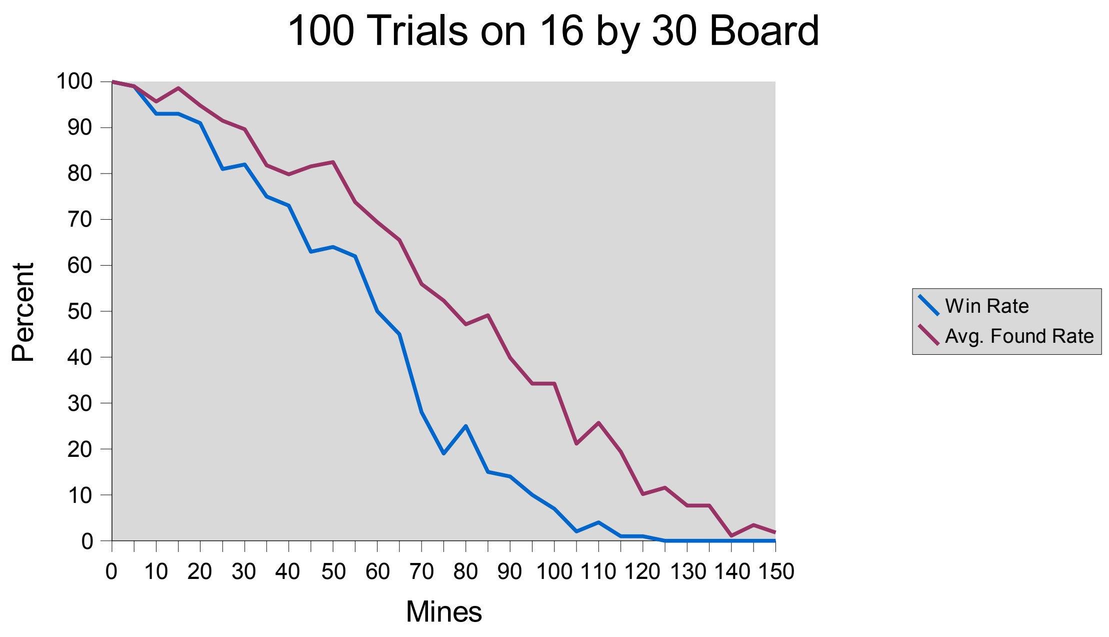
In order to test this hypothesis it was necessary to obtain measurements of the difficulties of a range of different board configurations. Originally the experiment was set up to run on boards of width 1 to 21 with 0 to 100 mines, at 5 mine intervals with 50 trials per configuration. However the data from this experiment contained many outliers. In an effort to smooth out the data more trials were added per configuration and the intervals were narrowed. But even with 200 trials per configuration and intervals of 2 mines there was no significant improvement in the data. In fact the only significant change was the running time, which increased dramatically. Due to this dramatic increase in processor time the experiment was restricted to boards of width 1 to 21 with 0 to 100 mines, at 5 mine intervals with 100 trials per configuration. One final tweak was made to include mine densities up to 150 mines to force the MineBot to have a consistent 0% win rate for all board shapes.

In addition to win rate another metric called average found rate was introduced for each set of trials. Average found rate was calculated by averaging the found rates for each trial, where the found rate for a trial was computed by dividing the number of mines identified by the total number of mines on the board. The purpose of adding average found rate was to ensure that the inconsistencies found in the win rate were not caused by that metric but were inherent in the game, if average found rate showed similar inconsistencies then the win rate metric itself could be ruled out as a possible cause of the outliers.

Results

Plots of the two metrics show a linear decrease in win rate and average found rate between 30 and 110 mines. Between 0 and 30 mines there is a short lead-in period where the slope of the metric increases, that is, the difficulty is increasing at an increasing pace. Between 110 and 150 mines there is a similar lead-out period where the slope of the metric decreases, the difficulty increases at a decreasing pace. See Illustration 1.

Illustration 1



Conclusion

The statistics gathered by the MineBot failed to show any evidence of a crystallization point. Instead they showed that, in general, difficulty increases roughly linearly with an increase in the number of mines. The shape of the win rate curve seems to indicate that the probability of solving a board is a function of the probability of picking a safe location at random when no other information is available. That the average found rate declines in exactly the same fashion confirms that this is the case.

Experiment Three

An investigation of how the shape of the board while maintaining the same number of mines changes the difficulty of a game.

Hypothesis

I hypothesized that square boards would be easier to solve than rectangular boards and that there would be a linear correlation between the width of the board and the percentage of wins. In other words, as the width of the board increases the difficulty of the board should increase in direct proportion.

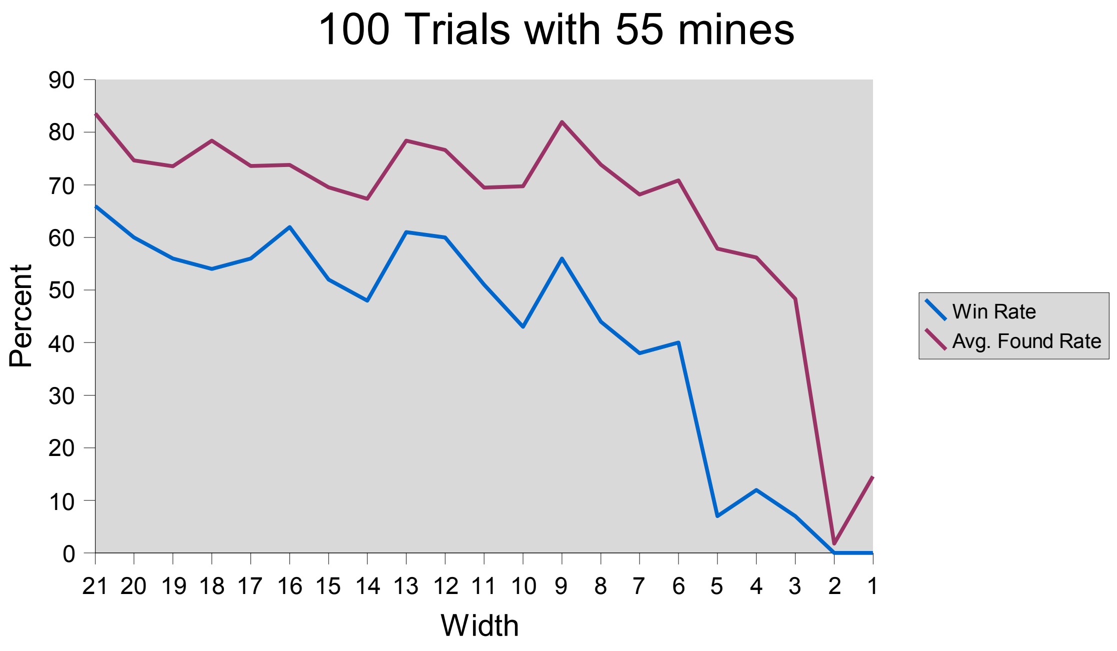
Method

The data used for this experiment was exactly the same as in Experiment Two.

Results

The plots of the two metrics for most of the configurations show almost no significant variation in win or average found rate until the board becomes less than or equal to 5 units wide. In general, after 5 units wide the win and average found rates plummet and for all cases, except where the number of mines was zero, the win rate for boards of width 2 was 0%. However, in many of the configurations this was followed by an increase in the win and average found rate at width 1. See Illustration 2.

Illustration 2



Conclusion

Again, the statistics gathered failed to support the hypothesis. The shape of the board, except in extreme circumstances, seems to have absolutely no effect on the difficulty of the game. What probably makes the narrower boards extreme is that they have a higher probability of being divided into sections by a chain of mines. When a chain of mines completely surrounds an area of the board then it causally separates the inside of the area and its surroundings. So no information can be gained about the area from the outside and no information can be gained about the outside from the area. Whenever this situation occurs it forces the player to make a guess and, as we saw in Experiment Two, the difficulty of a board is a function of the probability of picking a safe location at random when no other information is available. A board that is more likely to have causally separated sections is therefore a board which is more difficult to solve.

Further Research

The most important piece of information that this research lacks is a concrete statistic measuring the probability that a human player will win any given game. Until this statistic is available it will be impossible to truly determine how the MineBot compares to a human player and thereby provide evidence for the completeness of the MineBot. Even without this knowledge some improvements to the MineBot have been identified. The first improvement is a fix to the Burnout algorithm. A flaw exists in Burnout that prevents all safe locations from being identified. Another improvement is the addition of an algorithm that looks for similarities across all possible mine configurations on the remaining unknowns on the board. That is to say, an algorithm that investigates whether there is only one valid way to place all the remaining mines. Finally, the last planned improvement is to devise an “educated” guess algorithm, that guesses among the locations with the least probability of harboring a mine.

Other areas of possible research include: Investigating the theoretical lower bound on the asymptotic complexity of these algorithms and whether the current implementations can be improved to more closely approach this lower bound. Investigating whether the algorithms developed could be generalized for other Minesweeper-like games, perhaps one with a different kind of neighborhood? Investigating whether the algorithms developed could be be applied in non-Minesweeper application domains? And finally, investigating whether it would be possible to teach an artificially intelligent agent to play Minesweeper. For example could a neural net be trained to play Minesweeper by feeding it boards and informing it when it had won/lost?