Minesweeper

The purpose of this project is to explore minesweeper as a problem solvable by an algorithm, specifically I'm looking to solve it as a constraint satisfaction problem. While I take note that this has already been done before, I attempt to do it without any outside input as I imagined it before doing literature research.

1 Introduction

The focus of my project is on minesweeper, which is a video game that has existed in its present version since 1990. The problem that exists is trying to solve a game of minesweeper without hitting a mine which is considered a failure state. A normal game of minesweeper consists of a board or field which consists of squares that conceal what is underneath them. A square may contain either a mine, or a number that depicts how many of the adjacent 8 squares have mines (sometimes 0's are depicted as blanks.) A player may reveal one square at a time, and the game is considered solved when all squares except for those containing mines are revealed. Minesweeper traditionally has three different difficulties: beginner, intermediate, and expert but the only differences between difficulties is the size of the board, the number of mines, and the mine density. All three of the parameters increase as difficulty goes up. Because mine placement is random, a player or algorithm must use logic with the information the explored portion of the board provides. However, there are multiple mine patterns which can trip up any attempt at clearing the board without guessing, especially at a higher difficulty. Because of this, minesweeper algorithms have a chance to fail, and the complexity of the behavior programmed into the solving agent can have a large input on the success rate of the agent. When comparing highscores of humans, the time taken for completion is usually considered, however for differing algorithms, the rate of success is usually more important than time complexity. This problem is interesting because it has simple logic rules that are consistent and solve the game, however there are possible states that no rule can work on 100%, therefore there is guessing involved which increases the complexity of the problem.

2 Literature review

Minesweeper is a well known game, having been on windows computers for many years. The goal of minesweeper can be posed as a problem: How can we find all the mines with no prior knowledge of the field, and through uncovering only partial information throughout? Minesweeper was proven by Kaye in 2000 in his paper "Minesweeper is NP-Complete" [3]. It is an NP-complete problem because an algorithm must guess where a mine is, and then check to see if that would fit with the numbers currently being displayed on the board. This makes it part of one of the biggest open math problems trying to prove P = NP or in other words, that a problem that can be verified in polynomial time can also be solved in polynomial time. As we know, NP problems such as the traveling salesperson problem can be solved in a variety of ways.

A popular applet "programmers minesweeper" made by John D. Ramsdell [6] shows up frequently in literature addressing AI solutions to minesweeper. As demonstrated in Pedersen (2004) [5] the basic algorithm provided with programmers minesweeper is a very naive single point strategy that only uses a single square that has been probed and its neighbors to determine if a move is safe or not. While this strategy works well in beginner difficulties having upwards of 75% wins, using it on expert boards resulted in a half of a percent of all games played as wins. Ramsdell then offers a limited search strategy which enjoyed much improved results by defining a zone of interest around a square that may have a mine. A state where the square in question has a mine is proposed by the algorithm which then checks if it can then place the remaining mines around the zone of interest based off of this hypothesis. If no such state exists, the algorithm backtracks.

In the research done by [7], Studholme (2000) puts forward minesweeper as a constraint satisfaction problem. Each square on the field contains a boolean of either yes (1) there exists a mine or no (0) and that successfully revealing a square cements the boolean as 0 and gives a constraint that the surrounding squares must have boolean values that add up to this number. After simplifying constraints a much as possible, a backtracking algorithm is used to generate all possible solutions before throwing out solutions that don't contain the right number of mines. Often times, this is where one solution arises for the locations of local mines, but also often do we see situations where the algorithm cannot obtain any more information to deduce where a mine is and must take a guess [7]. However there arises a few problems with the thinking of Studholme. In cases where the first probe on a mine field can't be a mine, picking the corner because it has the highest chance of being a 0 often makes the problem much more difficult, even forgoing 100% win rates in sufficiently small fields. An article by Bufft, Lee, Lin, and Teytuad (2013) found that instead of simply using CSP algorithms with heuristics to guess mine locations, combining this type of solver with upper confidence trees greatly improved success rate on smaller boards, while only taking a slight hit in performance. Their algorithm mainly uses upper confidence trees, but with Heuristic CSP instead of the Monte-Carlo used in upper confidence trees [1]. Without the "first probe is a 0" rule however, some of these findings may turn out differently, which is something I've noticed, namely that most of these experiments assume that the minefield is generated after the first click, but some don't.

A separate way of modeling this problem is shown by the work done by Legendre, Hollard, Bufft and Dutech (2012) which models the problem as a Partially Observable Markov Decision Process

or POMDP that then utilizes heuristics. POMDP is considered a Markov decision processes but beliefs, or possible states of the problem. POMDP can be used to maximize immediate rewards, or maximize future rewards and in this paper, adjusting the reward function to 0 everywhere except when entering a failure state which then produces a -1 leads the algorithm to minimize the chance of falling into the failure state [4]. Interestingly, in [4], part of the proposed heuristic involves how much information probing a square might reveal, that squares on the edge of the field are more likely to have 0 values, squares far from the edge of the field and from the current safe area uncovered are also more likely to reveal more information because the data they might reveal goes in all 8 directions, and that squares on the edge of the safely uncovered area are likely to give less information on the whole, but more information on squares suspected to have mines. I presume this is because POMDP is heavily influenced by the number of possible states the board can be in and how better states lead to solutions. With this heuristic in mind, if the field is mined before the first probe, probing the edge or corner is better as previously mentioned, otherwise it is best to probe the center.

Other previous work on this problem have lead to some interesting implementations, such as having an AI learn to play minesweeper with multirelational learning and reaching 52% success rate in Castillo, Wrobel (2003) [2].

3 Approach

For this to be a constraint satisfaction problem, we need constraints. Each square on the map will be given a boolean value if it is a mine or if it isn't. Each square that isn't a mine also will contain a number that tells how many mines surround this square. The main constraint is that for each number N held by a square, the number of mined squares around the square must be equal to N. We don't want to open mined squares, and a square with this number of neighbors unopened has its constraint satisfied. If a square has unopened neighbors, but it already knows where all of its neighboring mines are, it's neighbors should be opened. Each square's constraint needs to be satisfied before be no longer consider probing it, and when a square is complete, we pick a new square to complete. If a square's constraint cannot be completed we take note of the number of uncertain mine placements and use that number as a flag when returning to the square. We then backtrack to the previous square and select a new square to try to satisfy. In this way, when a square is completed we can move onto its neighbors unless there are no more neighbors to move to, then we backtrack, before finally revisiting incomplete squares in case there is new information. If the board is unsolved but we have no safe moves we can make a guess on a square, or attempt to open a square separate from our frontier depending on the chance of hitting a mine in the remaining unopened squares and the current mine density in those squares. If a new frontier is created, we can proceed as normal from there. If a guess needs to be made, a random guess on a square with the lowest unfound mines to unopened neighbor ratio might be favorable, however this isn't optimal because there are some tricks when it comes to "guessing" like when you have a 2,1,and 2 square in a line and the unopened frontier is adjacent and parallel to these squares. In this case, the first 2 square might have neighbors a+b+c=2 (one of these is a mine) the 1 square has neighbors b+c+d=1 and the second 2 square has neighbors c+d+e=2. These cases need to be

recognized and in this case, the c square needs to be opened. If there aren't any remaining safe moves and these cases are ruled out, then a guess needs to be made to move on.

4 Design

Having found Ramsdell's app, I knew that I wanted to borrow it as both a visual representation of completing a minesweeper game, and as a testing framework. Using programmers minesweeper allows me to see the algorithm as it works on completing the problem. It also already has much of a traditional minesweeper minefield coded into it, with beginner, intermediate, and expert difficulties. But most importantly it can simulate a string of games very quickly and show their aggregate stats. The main difference this implementation of minesweeper has with the version made for windows is that this version generates all the mines before the first square is probed instead of generating the field after the first square is probed so that the first move cannot result in failure. The algorithm will be tested 500 times on each difficulty to get a feel for it's success rate and then compared to results from the algorithms provided with programmers minesweeper.

References

[1] O. Buffet, C.-S. Lee, W.-T. Lin, and O. Teytuad. Optimistic heuristics for minesweeper. In *Advances in Intelligent Systems and Applications-Volume 1*, pages 199–207. Springer, 2013.

[2] L. P. Castillo and S. Wrobel. Learning minesweeper with multirelational learning. In *IJCAI*, pages 533–540, 2003.

[3] R. Kaye. Minesweeper is np-complete. *The Mathematical Intelligencer*, 22(2):9–15, 2000.

[4] M. Legendre, K. Hollard, O. Buffet, and A. Dutech. Minesweeper: Where to probe? 2012.

[5] K. Pedersen. The complexity of minesweeper and strategies for game playing. *Project report, univ. Warwick*, 2004.

[6] J. D. Ramsdell. Programmer's minesweeper. http://www.ccs.neu.edu/home/ramsdell/pgms/, 1995.

[7] C. Studholme. Minesweeper as a constraint satisfaction problem. *Unpublished project report*, 2000.