4511W Final Project Proposal

Nadya Postolaki
Fatima Rahman

March 25, 2020

1 Introduction (Fatima)

Though artificial intelligence research has tackled games such as Go, chess, and Sudoku in the past, our project aims to implement an AI to defeat the seminal Minesweeper. A single-player video game, Minesweeper has eluded both the young and old since its inception in the 1960s. Despite its deceptively simple premise, where all one has to do is dig around randomly placed mines, the game requires a degree of intuition and guesswork alongside logical thinking. One false click can trigger a series of explosions and ultimately your loss. In essence, we strive to implement an efficient take on Minesweeper by addressing it as a constraint satisfaction problem and using local consistency measures such as the AC-3 algorithm (and variations such as MAC), forward checking, and the min-conflicts algorithm. We will calculate said measures' efficacy based on the win ratio of this implementation under specified time limits.If time permits, we will also try our hand at developing Minesweeper specific heuristics to use during search.

2 What is Minesweeper? (Fatima/Nadya)

Using Stuart Russell and Peter Norvig's terminology[1], Minesweeper can be classified as a partially observable, deterministic, and static game. Though the environment is "known" in the sense that the game board operates by a strict set of rules, the player has no access to the full, unveiled Minesweeper board until the end, thus classifying it as "uncertain." But how does one play the game? First, let us visualize the game board:

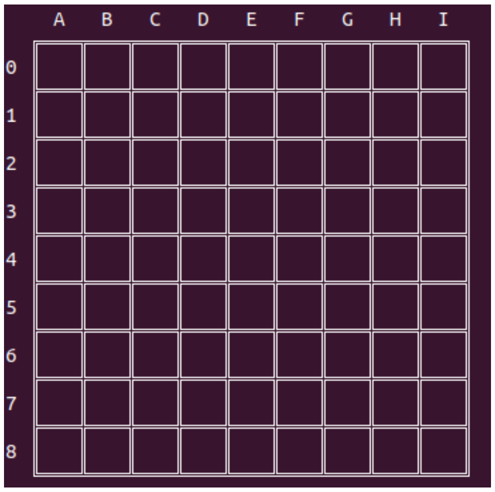
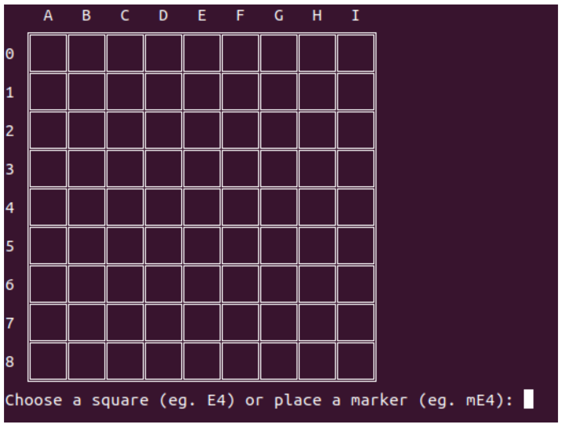


Figure 1: Blank Minesweeper Board

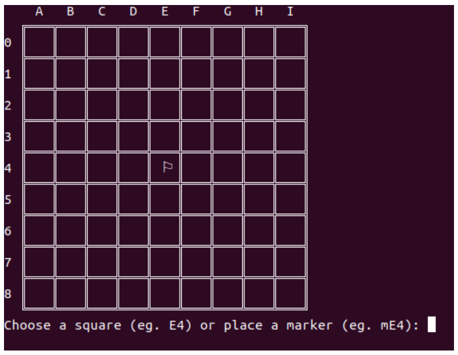
Before the player makes any moves, the game's initial state is a board (or "grid") of covered cells. Depending on one's chosen difficulty level (easy, medium, or hard), the board can be anywhere around 8x8 squares to 16x32 in most Minesweeper programs. Our project implementation will use a relatively easy size of 9x9.

The player can then click anywhere on said board, revealing a block of numbered and empty cells — the key is that one's first click will never hit a mine. Once this first click is made, an agent has two possible actions, either left or right click. A left click is used to "shovel" a cell, i.e. reveal its contents. A right click, on the other hand, "flags" a cell as a potential mine location.



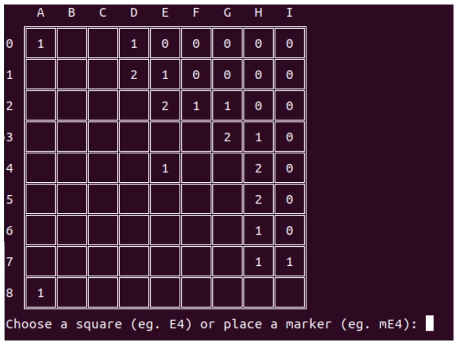
```
Choose a square (eg. E4) or place a marker (eg. mE4):
```

Figure 2: Instructions



```
   A   B   C   D   E   F   G   H   I
0 |   |   |   |   |   |   |   |   |   |
1 |   |   |   |   |   |   |   |   |   |
2 |   |   |   |   |   |   |   |   |   |
3 |   |   |   |   |   |   |   |   |   |
4 |   |   |   |   | ⚐ |   |   |   |   |
5 |   |   |   |   |   |   |   |   |   |
6 |   |   |   |   |   |   |   |   |   |
7 |   |   |   |   |   |   |   |   |   |
8 |   |   |   |   |   |   |   |   |   |
Choose a square (eg. E4) or place a marker (eg. mE4):
```

Figure 3: Placing a Marker



```
   A   B   C   D   E   F   G   H   I
0 | 1 |   |   | 1 | 0 | 0 | 0 | 0 | 0 |
1 |   |   |   | 2 | 1 | 0 | 0 | 0 | 0 |
2 |   |   |   |   | 2 | 1 | 1 | 0 | 0 |
3 |   |   |   |   |   |   | 2 | 1 | 0 |
4 |   |   |   |   | 1 |   |   | 2 | 0 |
5 |   |   |   |   |   |   |   | 2 | 0 |
6 |   |   |   |   |   |   |   | 1 | 0 |
7 |   |   |   |   |   |   |   | 1 | 1 |
8 | 1 |   |   |   |   |   |   |   |   |
Choose a square (eg. E4) or place a marker (eg. mE4):
```

Figure 4: Partial Board Reveal

Our case deviates from the classic implementation in that the player must type in a location they wish to either reveal or flag. If a cell is chosen to be revealed, the game checks whether the value of the cell is a bomb or not. If it is a bomb, an explosion is triggered and you will automatically lose the game. If not, it will reveal a numbered cell. If not, the value of the cell is revealed by showing the amounts of mines surrounding that cell in particular. If the value of the cell is zero, then other surrounding blocks are also revealed.

The number (in the domain 1-8) signifies how many mines border the cell, in any direction (horizontal, diagonal, vertical). For example, if a cell is marked "1," there is only one mine that touches the cell. All other surrounding squares are not mines. Minesweeper's terminal test in essence is either failure (i.e. explosion triggered) or the goal state, denoting the user's success in correctly identifying all mine sites and digging up all other cells. Not only is this game zero-sum, but it is also randomized. Once one replays the game, the board is wiped clean and the mine placement is once again rearranged.

3 Implementation (Fatima/Nadya)

Using pre-existing code in Repl.it's Python tutorial for Minesweeper with modifications and AIMA's CSP functions, we plan to conceptualize Minesweeper as a constraint satisfaction problem (CSP).[5] We hope to focus on the uses of AC-3 in problem solving within the game of Minesweeper, as well as exploring other algorithms with which we will test against each other to find the best one for solving a randomized game. Because Minesweeper is NP-complete[2], meaning it runs in non-deterministic polynomial time, guessing is required to implement a truly optimized algorithm.

Following the University of Nebraska-Lincoln's Constraint Systems Laboratory approach as well as University of Toronto's Chris Studholme's methodology[4], each cell will be considered a Boolean variable, 0 indicating a "safe" square and 1 a mine. We will enact "sum constraints," which "[specify] the total number of constraints in the adjacent squares. For example, a square labeled 3 yields a constraint stating that the square be surrounded by 3 mines." [3] This indicates that each uncovered variable poses a constraint on its 8 "neighbors," i.e. the surrounding covered cells. The sum of all the board's must be equal to the number of mines.[6]

We will evaluate our solution on the proportion of wins to losses in an allotted time. If the constraint propagation works successfully, our AI-based Minesweeper solver will work over half the time.

4 Conclusion (Fatima)

Though we have no preliminary results yet, we are excited to see where the CSP implementation will take us! We both had a strong desire to not only explore measuring algorithm efficiency and general-purpose problem solving, but study the efficacy of constraint propagation methods such as arc consistency. Our general time frame is to finish building the general Minesweeper program by April 3, based on the Repl.it implementation. Then by April 10, we plan to have finished the implementation of CSP construction. If we have time after finishing the writeup, we will implement Minesweeper at a grander scale by upgrading the GUI and scaling up the sizes (adding a 16x16 board with 40 mines for the "medium" difficulty option and a 30x16 board with 99 mines for the "hard" option).

References

[1] Stuart J. Russell, Peter Norvig. Artificial intelligence. In Marcia Horton, editor, *A Modern Approach Third Edition*, chapter 1, pages 8–9. Pearson Education, New Jersey, 2010.

[2] Richard Kaye. Minesweeper is np-complete, 2000. http://simon.bailey.at/random/kaye.minesweeper.pdf.

[3] Josh Snyder Ken Bayer and Berthe Y. Choueiry. An interactive constraint-based approach to minesweeper, unknown. https://www.aaai.org/Papers/AAAI/2006/AAAI06-344.pdf.

[4] Chris Studholme. Minesweeper as a constraint satisfaction problem, 2007. http://www.cs.us.es/cursos/ia1-2007/trabajos/trabajo-2/minesweeper-toronto.pdf.

[5] ThomasS1. How to program minesweeper in python, 2018. https://repl.it/talk/learn/How-to-program-MineSweeper-in-Python-fully-explained-in-depth-tutorial/9397.

[6] Stanford University. Minesweeper ai, 2018. http://web.stanford.edu/class/archive/cs/cs221/cs221.1192/2018/restricted/posters/thowarth/poster.pdf.