Heuristically Solving Minesweeper Project Report

By Jacob Singer

Abstract

This project was an insight into how the game Minesweeper is solved and implementing a way for a computer to solve the game. This project was implemented in Common Lisp. I created a standalone program to play Minesweeper, then created an additional program, a “player” which would be used to play the game.

Introduction

Minesweeper is a logic puzzle video game that was released in 1990 as part of the “Microsoft Entertainment Pack for Windows.” It was written by Robert Donner and Curt Johnson at Microsoft with the intention of “teaching people basic mouse controls in an era where most computing had been text-based” [1]. Minesweeper consists of a 2-dimensional board of tiles. These tiles start hidden and can be revealed by the player. Mines are placed randomly across the board, clicking on a mine loses the game. The player can place a flag on a tile to indicate that it is a mine. The goal is to reveal all of the tiles which are not mines. There were 3 standard difficulties in the original Minesweeper. Beginner – 10x10 board with 10 mines, intermediate – 16x16 board with 40 mines, and expert – 30x16 with 99 mines.

The goal of this project was to create a program which could solve the majority of Minesweeper games. Solving every single game is not feasible by any program with the default board generation because it is common to reach a point where a guess is the only way to solve the game.

Background

Due to its relative simplicity in design and in playing, Minesweeper has been a topic for research by quite a few different people over the years. An extensive list of these papers can be found on the “Authoritative Minesweeper” website [2]. One of the first research papers done about Minesweeper was done by Richard Kaye titled “Minesweeper is NP-Complete.” In this paper Kaye talks about the age old P = NP question. He makes the claim that Minesweeper is NP-complete, along with the statement that “it may even be that some polynomial-time algorithm is ‘good enough’ at solving the sort of Minesweeper problems that occur in practice, even though (assuming P ≠ NP) it cannot actually solve all theoretically possible configurations” [3]. This paper set the expectations for this project in that no matter how good the program is, it will not be able to solve all games of Minesweeper. But with these expectations also came the goal of an algorithm which is “good enough.”

Two other studies done on Minesweeper were the thesis “Algorithmic Approaches to Playing Minesweeper” by David Becerra [4], and a project report “The Complexity of Minesweeper and

Strategies for Game Playing" by Kasper Pedersen [5]. These papers both discuss the difficulty of solving Minesweeper and algorithms to do so. Becerra's paper built off of a few of the ideas presented by Pedersen and was my main point of reference for my implementation.

Program Description

The first half of my project involved the planning and creation of the game itself. I ended up using the Common Lisp Object System to model the tiles and the game board. To reveal tiles, I created a function which reveals one tile then recursively reveals surrounding tiles if the current tile has a value of 0. Thinking ahead, I wrote this function as a Boolean. If a mine was revealed it would return true which could be used in both the game interface and the game playing algorithms. I made a board generation function which randomly placed the mines on the board. I then made a REPL in which a user could play Minesweeper on the command line. I implemented the 3 difficulties explained in the introduction with a few small tweaks. I named them easy, medium, and hard respectively. The dimensions of hard were also changed to 24x20, which retains the same number of mines as 30x16 but looks cleaner on a terminal.

The second half of the project was dedicated to creating the heuristic player. The first thing implemented was a completely random player. I then planned to continuously add heuristic rules to the player. The first rule I implemented was one explained in Becerra's thesis which involves which tile is

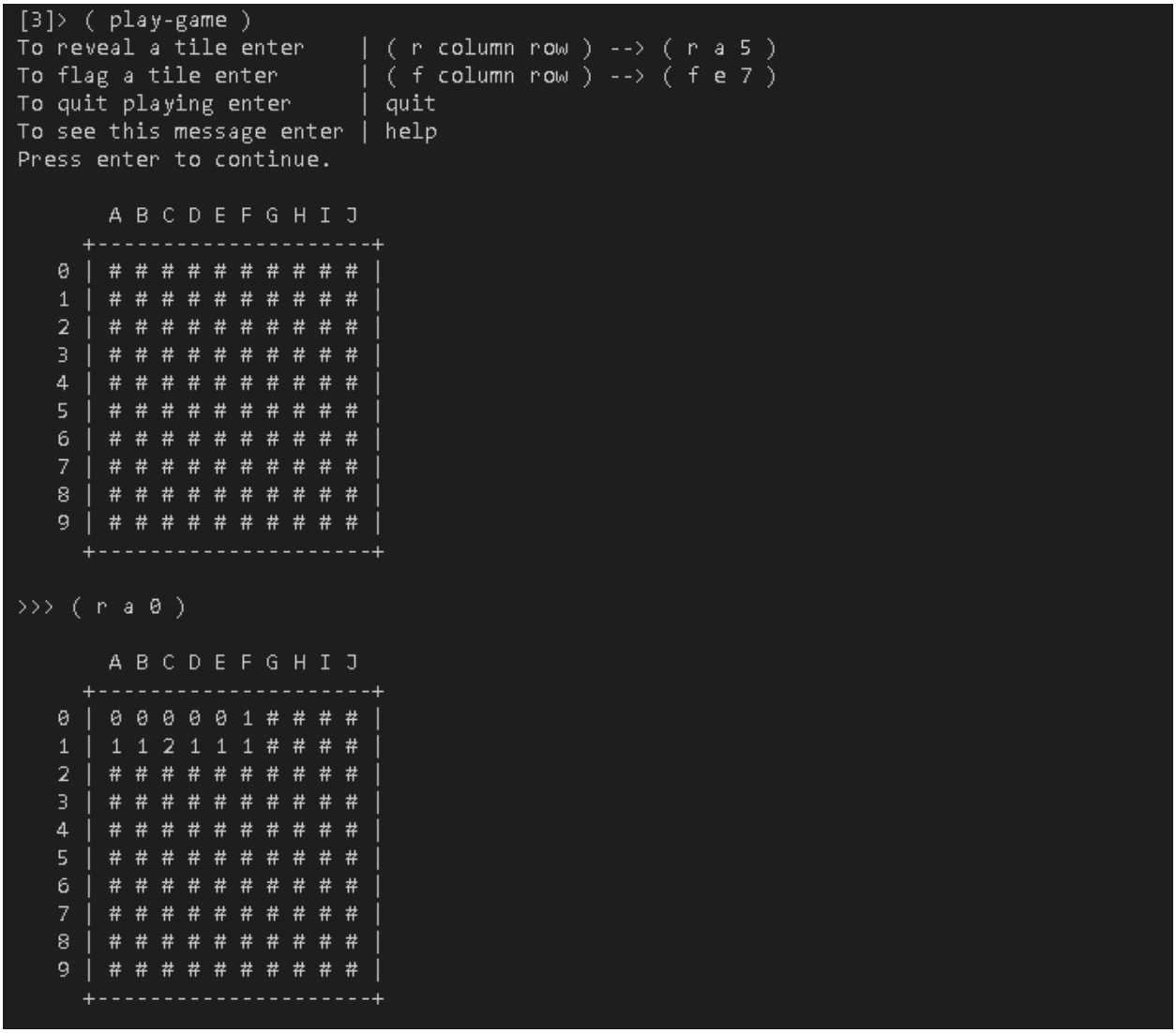
the best to reveal first [4]. Some games of Minesweeper have true random board generation, including the original Microsoft release and the one I wrote. With this fact, the first tile revealed can be a mine. This makes the first click a random guess. This fact leads to the question, which tiles are the best to click first? The answers to this question are the corner tiles. Since the corner tiles only have 3 neighbors, there is a higher chance for the corner tiles to be a 0, which would result in more of the board being revealed. The next strategy I created was one I had thought up myself, which was then later confirmed as a good strategy by both Becerra and Pederson. Pederson labeled this method as the “Single Point Strategy” [5]. This method probes a single tile and evaluates its neighbors to see if any tiles are safe to be flagged or revealed. This is as far as I was able to get with implementation.

Single point evaluation is an effective rule on most easy boards and some medium boards but was still left a lot of hard patterns which would need the information from multiple tiles to be solved. My first thought for this was to basically hard code in common patterns and search for them throughout the board. Upon further research and thought, I found that this method would not be feasible, as checking every tile for every pattern in any orientation would have been very computationally expensive. Then with research I found a paper by Chris Studholme which modeled Minesweeper as a Constraint Satisfaction Problem [6]. This method seemed promising, but because of time

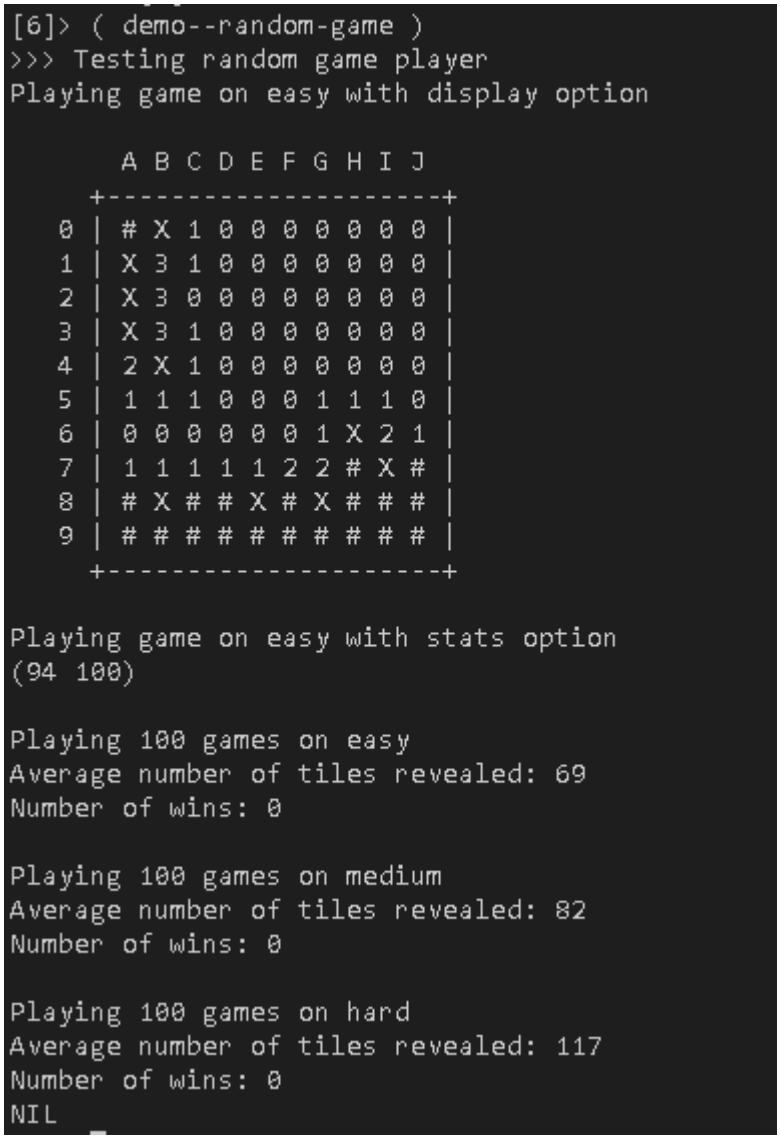
limitations and the difficulty of implementation, I was unable to finish it.

Code Demos

Game Playing Interface



Random Player



```
[6]> ( demo--random-game )
>>> Testing random game player
Playing game on easy with display option

      A B C D E F G H I J
    +----------------------+
  0 | # X 1 0 0 0 0 0 0 0 |
  1 | X 3 1 0 0 0 0 0 0 0 |
  2 | X 3 0 0 0 0 0 0 0 0 |
  3 | X 3 1 0 0 0 0 0 0 0 |
  4 | 2 X 1 0 0 0 0 0 0 0 |
  5 | 1 1 1 0 0 0 1 1 1 0 |
  6 | 0 0 0 0 0 0 1 X 2 1 |
  7 | 1 1 1 1 1 2 2 # X # |
  8 | # X # # X # X # # # |
  9 | # # # # # # # # # # |
    +----------------------+

Playing game on easy with stats option
(94 100)

Playing 100 games on easy
Average number of tiles revealed: 69
Number of wins: 0

Playing 100 games on medium
Average number of tiles revealed: 82
Number of wins: 0

Playing 100 games on hard
Average number of tiles revealed: 117
Number of wins: 0
NIL
```

Heuristic Player

```
[11]> ( demo--heuristic-game )
>>> Testing heuristic game player
Playing game on easy with display option

      A B C D E F G H I J
    +---------------------+
  0 | 0 0 1 > 1 0 0 0 0 0 |
  1 | 0 0 1 1 1 0 0 0 0 0 |
  2 | 0 0 0 0 0 0 0 1 1 1 |
  3 | 0 0 0 0 0 0 0 1 > 1 |
  4 | 1 2 2 2 1 0 0 1 1 1 |
  5 | > 3 > > 1 0 0 0 0 0 |
  6 | > 3 2 2 1 1 1 1 0 0 |
  7 | 1 1 0 1 1 3 > 2 0 0 |
  8 | 1 1 0 1 > 3 > 2 0 0 |
  9 | # 1 0 1 1 2 1 1 0 0 |
    +---------------------+

Playing game on easy with stats option
(10 100)

Playing 100 games on easy
Average number of tiles revealed: 77
Number of wins: 76

Playing 100 games on medium
Average number of tiles revealed: 148
Number of wins: 30

Playing 100 games on hard
Average number of tiles revealed: 155
Number of wins: 0
NIL
```

Reflections and Conclusions

Stats on 100 games played						
	Easy		Medium		Hard	
Iteration	Avg Tiles	Wins	Avg Tiles	Wins	Avg Tiles	Wins
Random	58	2	85	0	116	0
Corner 1st	66	10	86	0	124	0
Single Point	75	76	147	35	169	1

I would have liked to actually finish the constraint

satisfaction problem implementation as I had planned, but unfortunately it did not happen. Other than that, I am happy with the result of this project. The rules I did end up implementing worked very well on easy boards and ok on medium boards. I was able to learn more about one of my favorite games, and the surprising relation to computer science that it has. I might even continue working on this project in my spare time to try to make it “good enough” [3] at playing Minesweeper.

Bibliography

[1] R. Cobbet, "The Most Successful Game Ever: a History of Minesweeper," 5 May 2009. [Online]. Available: https://www.techradar.com/news/gaming/the-most-successful-game-ever-a-history-of-minesweeper-596504. [Accessed 12 May 2023].

[2] "Authoritative Minesweeper," [Online]. Available: minesweepergame.com/. [Accessed 5 May 2023].

[3] R. Kaye, "Minesweeper is NP-Complete," *The Mathematical Intelligencer,* vol. 22, pp. 9-15, 2000.

[4] D. Becerra, "Algorithmic Approaches to Playing Minesweeper," Hardvard College, 2015.

[5] K. Pedersen, "The Complexity of Minesweeper and Strategies for Game Playing," Department of Computer Science University of Warwick, 2005.

[6] C. Studholme, "Minesweeper as a Constraint Satisfaction Problem," University of Toronto, 2001.