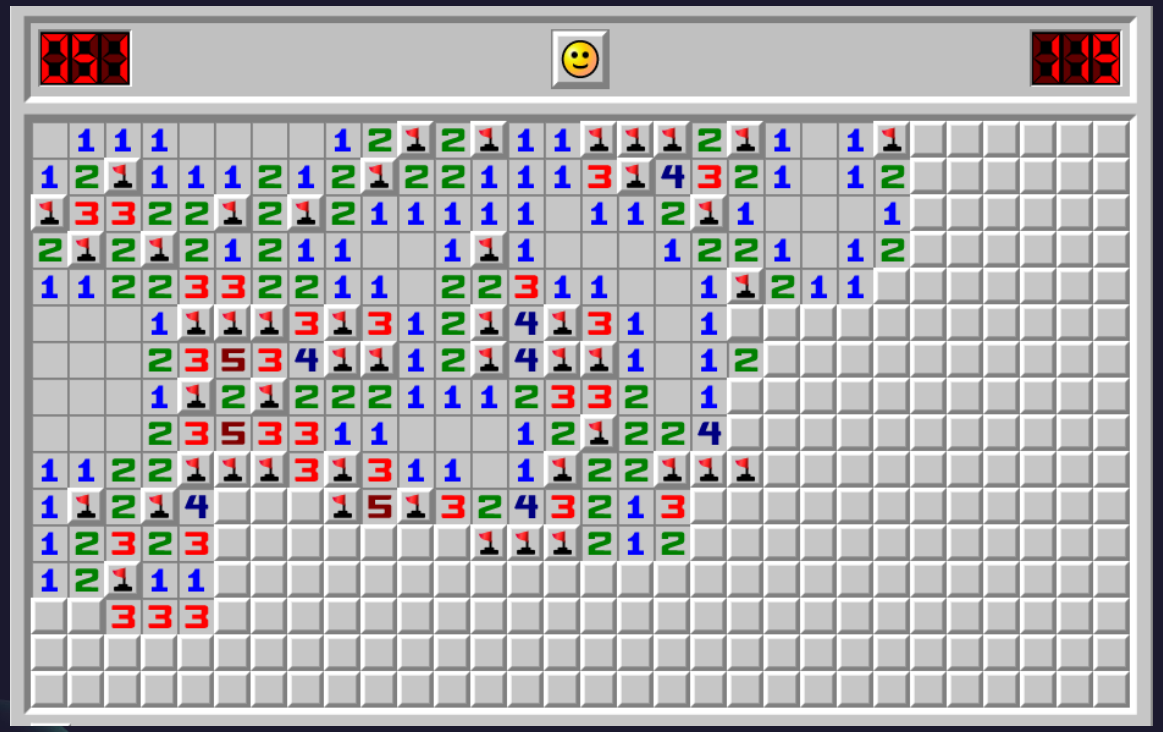
Heuristically Solving Minesweeper



What is Minesweeper?

- If you ever used a Windows machine from Windows 3 to Windows 7 its likely that you have seen and maybe even played Minesweeper.
- Minesweeper is a logic puzzle video game that was released by Microsoft in 1990 as part of the “Microsoft Entertainment Pack for Windows”
- It was developed as a way to teach Windows users how to use the relatively new invention of the computer mouse.

Core Gameplay Concepts

- The game is made up of a 2-dimensional board of tiles.
- Tiles start hidden until the player reveals them.
- Some tiles contain mines.
- When a tile is revealed, it may contain a number, this indicates how many tiles are adjacent to it.
- Tiles can be flagged to indicate the player knows a mine is there.
- If a mine is revealed the game is lost and all mines are shown.
- The player wins the game when all the tiles that are not mines are revealed

Goals of This Project

- Implement Minesweeper in CLISP.
- Create a random player.
- Create a simple heuristic player.
- Improve the heuristic player with advanced rules to win more games.
- Gather and compare statistics from each iteration of the player.

Game Implementation

- The tiles and board were implemented with the use of CLOS.
- The board can be randomly generated or given a list to generate a custom board.
- Tiles are recursively revealed.
- Simple command line REPL for users to play the game.

Game Demo

```
[2]> ( play-game )
To reveal a tile enter    | ( r column row ) --> ( r a 5 )
To flag a tile enter      | ( f column row ) --> ( f e 7 )
To quit playing enter     | quit
To see this message enter | help
Press enter to continue.

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

```
>>> ( r a 0 )

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

```
>>> ( f c 1 )

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

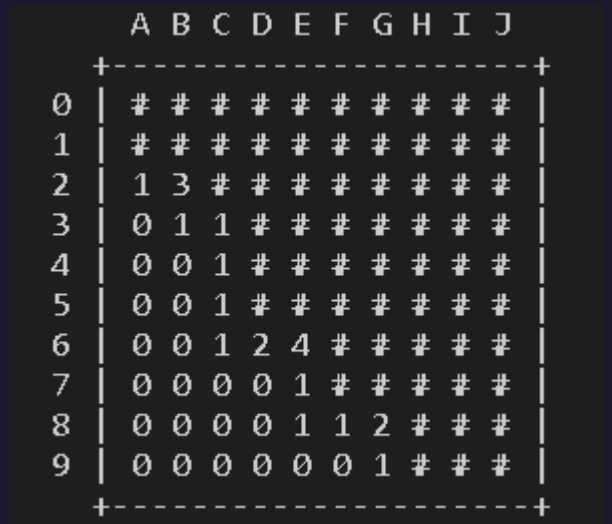
First Move Rule

- The first move in Minesweeper is always a guess.
- Some versions have more sophisticated board generation which make sure the first click is not a mine, but more traditional versions like in this project is completely random.
- This can lead to the first move being a loss.
- But with this comes a heuristic rule to help get started.
- Revealing a corner has a higher chance to cause a bigger potion of the board to be revealed.

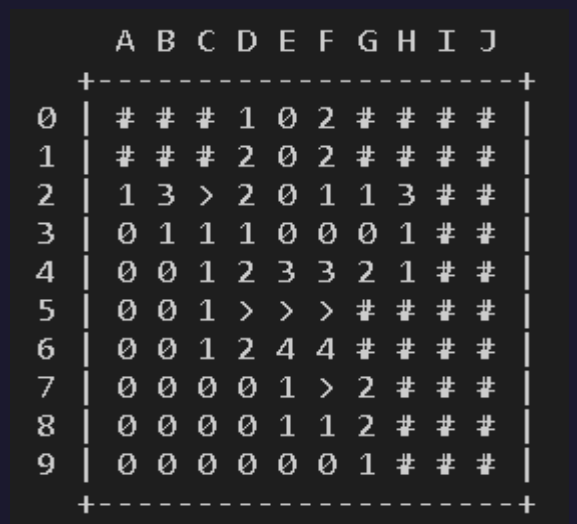
Single Point Evaluation

- This rule involves probing a single tile and checking its neighbors.
- If the value of the tile is equal to the number of neighbors then all of the neighbors are mines.
- If the value of the tile is equal to the number of flagged neighbors then the unflagged neighbors are safe to be revealed.

Single Point Demo

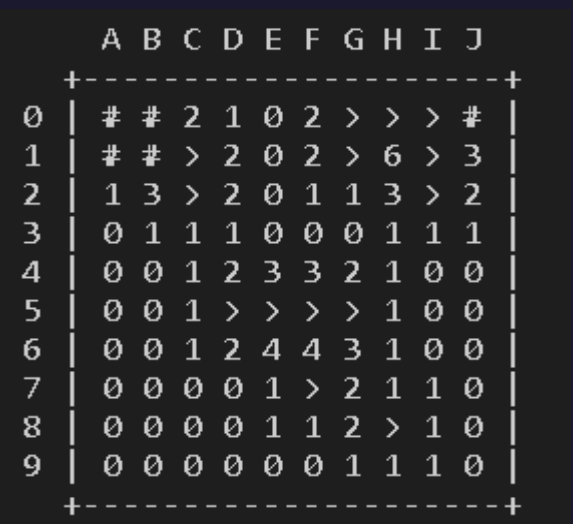


```
   A B C D E F G H I J
  +--------------------+
0 | # # # # # # # # # # |
1 | # # # # # # # # # # |
2 | 1 3 # # # # # # # # |
3 | 0 1 1 # # # # # # # |
4 | 0 0 1 # # # # # # # |
5 | 0 0 1 # # # # # # # |
6 | 0 0 1 2 4 # # # # # |
7 | 0 0 0 0 1 # # # # # |
8 | 0 0 0 0 1 1 2 # # # |
9 | 0 0 0 0 0 0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

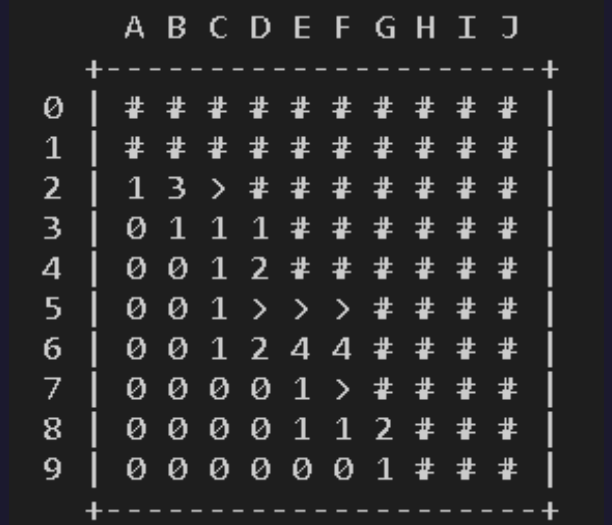


```
   A B C D E F G H I J
  +--------------------+
0 | # # # 1 0 2 # # # # |
1 | # # # 2 0 2 # # # # |
2 | 1 3 > 2 0 1 1 3 # # |
3 | 0 1 1 1 0 0 0 1 # # |
4 | 0 0 1 2 3 3 2 1 # # |
5 | 0 0 1 > > > # # # # |
6 | 0 0 1 2 4 4 # # # # |
7 | 0 0 0 0 1 > 2 # # # |
8 | 0 0 0 0 1 1 2 # # # |
9 | 0 0 0 0 0 0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

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



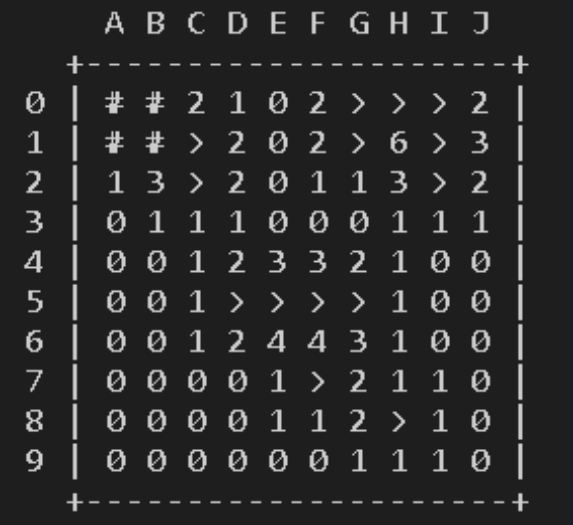
```
   A B C D E F G H I J
  +--------------------+
0 | # # 2 1 0 2 > > > # |
1 | # # > 2 0 2 > 6 > 3 |
2 | 1 3 > 2 0 1 1 3 > 2 |
3 | 0 1 1 1 0 0 0 1 1 1 |
4 | 0 0 1 2 3 3 2 1 0 0 |
5 | 0 0 1 > > > > 1 0 0 |
6 | 0 0 1 2 4 4 3 1 0 0 |
7 | 0 0 0 0 1 > 2 1 1 0 |
8 | 0 0 0 0 1 1 2 > 1 0 |
9 | 0 0 0 0 0 0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



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

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

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



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

Advanced Patterns

- Some patterns can only be solved via examining multiple tiles.
- My first idea for this was to hard code as many of these patterns as I could think of, but this approach is infeasible.
- In my research I found a way to solve these with the use of a constraint satisfaction problem.
- As of now this is not finished.

Stats

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

Reflections

- As the project currently is, it is not finished.
- I should have done more research and planning earlier as it was very late when I realized my initial ideas would not work.
- A future addition which could be added to improve the player could be to add a rule that calculates the probability of a tile being a mine for every tile to choose the tile least likely to be a mine instead of a random guess.