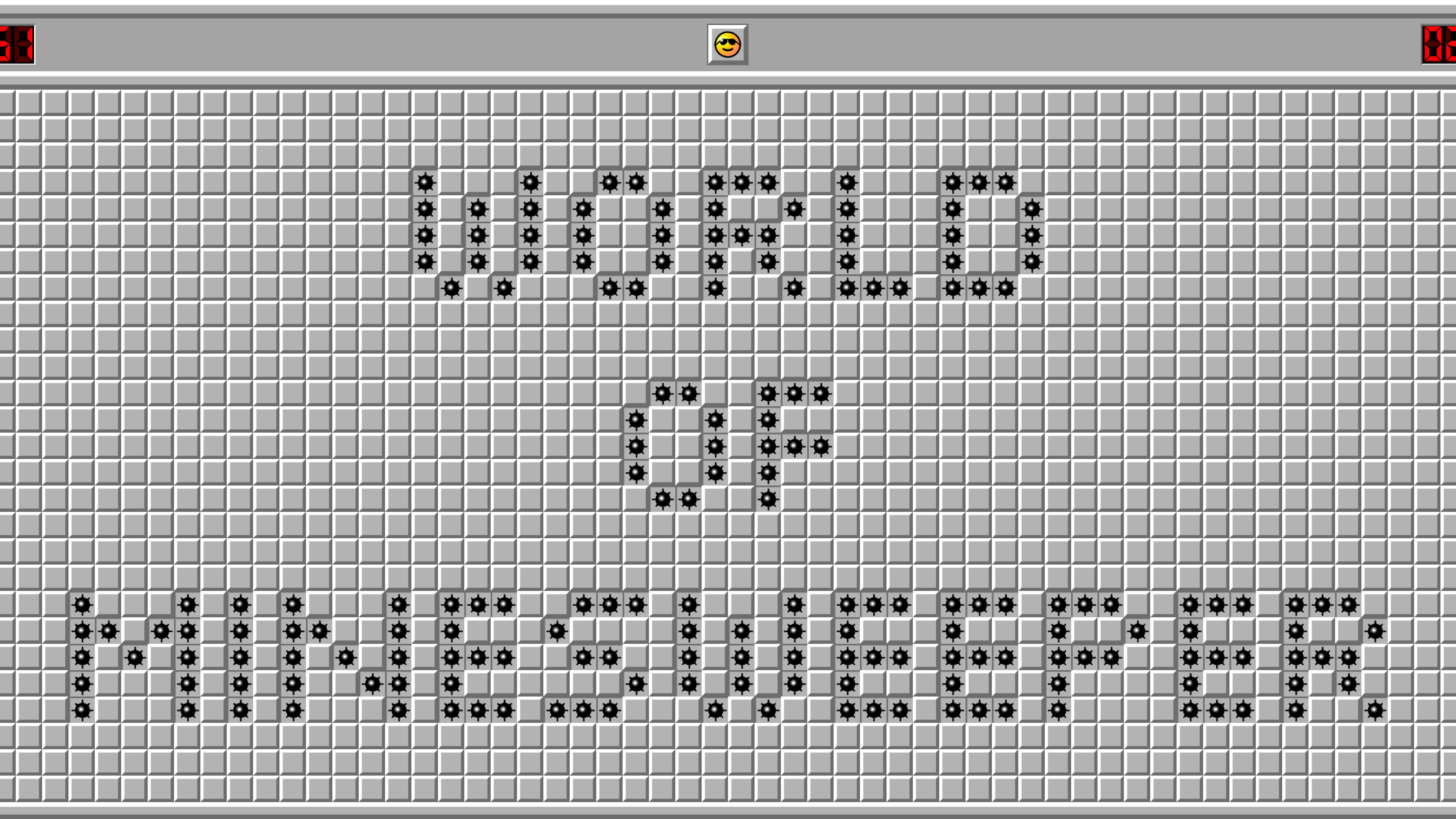
American University of Armenia

Minesweeper

Team:
Diana Sargsyan: diana_sargsyan@edu.aua.am
Mher Movsisyan: mher_movsisyan@edu.aua.am
Lilit Beglaryan: lilit_beglaryan@edu.aua.am

Supervisor
Prof. Monika Stepanyan


WORLD
OF
MINESWEEPER

LITERATURE REVIEW

Existing approaches to Minesweeper game

- RANDOM SEARCH (RS)
- HILL CLIMBING (HC)
- SIMULATED ANNEALING (SA)
- GENETIC ALGORITHM (GA)

Comparison of Algorithms

RANDOM SEARCH	Randomly generates successor configurations and picks the best of them. Brings Minesweeper into an Optimization problem with a penalty function (sum of the squared differences between opened mines and the indicators).	• Outputs a good solution locally but not for the whole board. • When encounters a bad configuration, discards it and starts a new iteration(missing the potential of that bad move to yield a good solution for the whole board).
HILL CLIMBING	1st step-randomly generates a solution. Makes small local changes with the hope of optimizing the penalty function.	• uses a penalty function • randomly places mines to generate new states • fast compared to RS
SIMULATED ANNEALING	Starts with a random configuration, maintains temperature t, assigns probablity values to states -c/t.	• keeps potentially good moves that seem to be bad locally • when t is high, similar to RS • when t is lower, similar to HC
GENETIC ALGORITHM	Starts with a population of potential solutions, assigns fitness value(inverse of penalty) for two fittest configurations A and B,takes one part of A, and then another part of B, plus a potential mutation = new successor	• uses random mutations and partitions • penalty function decreases as it moves always to fitter states

Constraint Satisfaction Problem

CSP is a factored representation of a mathematical problem, consisting of states whose solutions must satisfy a set of constraints

CSP consists of :

- set of domains
- set of domains
- set of constraints that specify allowable combinations of values

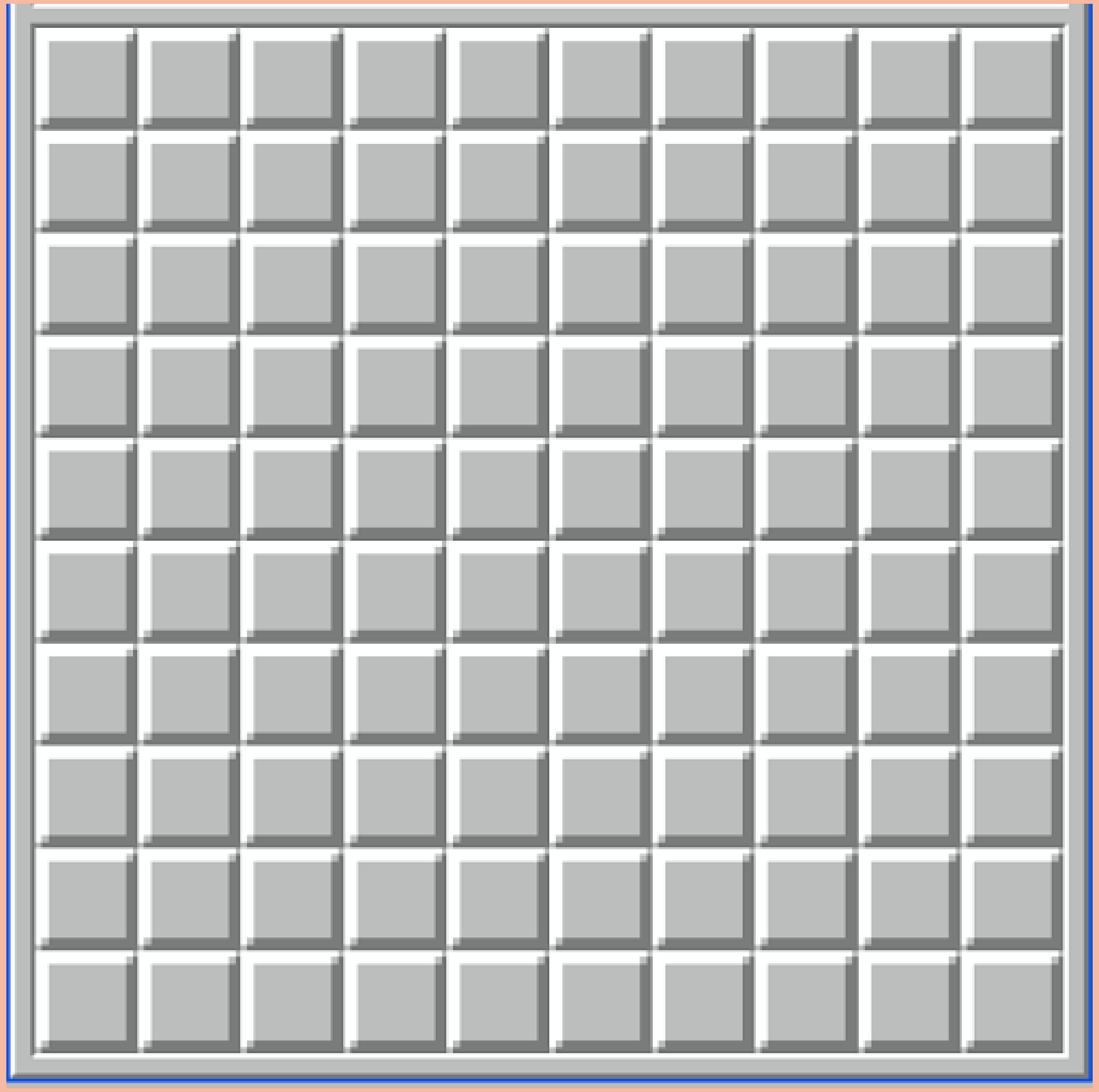
Minesweeper as a CSP

- the variables are squares of our 2D board
- the values of the domains are -1(mine) and 0(safe)
- each cell is in a constraint with its neighbours

such that the sum of abs. values of the neighbors is equal to the indicator assigned to that constraint

OUR APPROACH TO MINESWEEPER

Algorithm

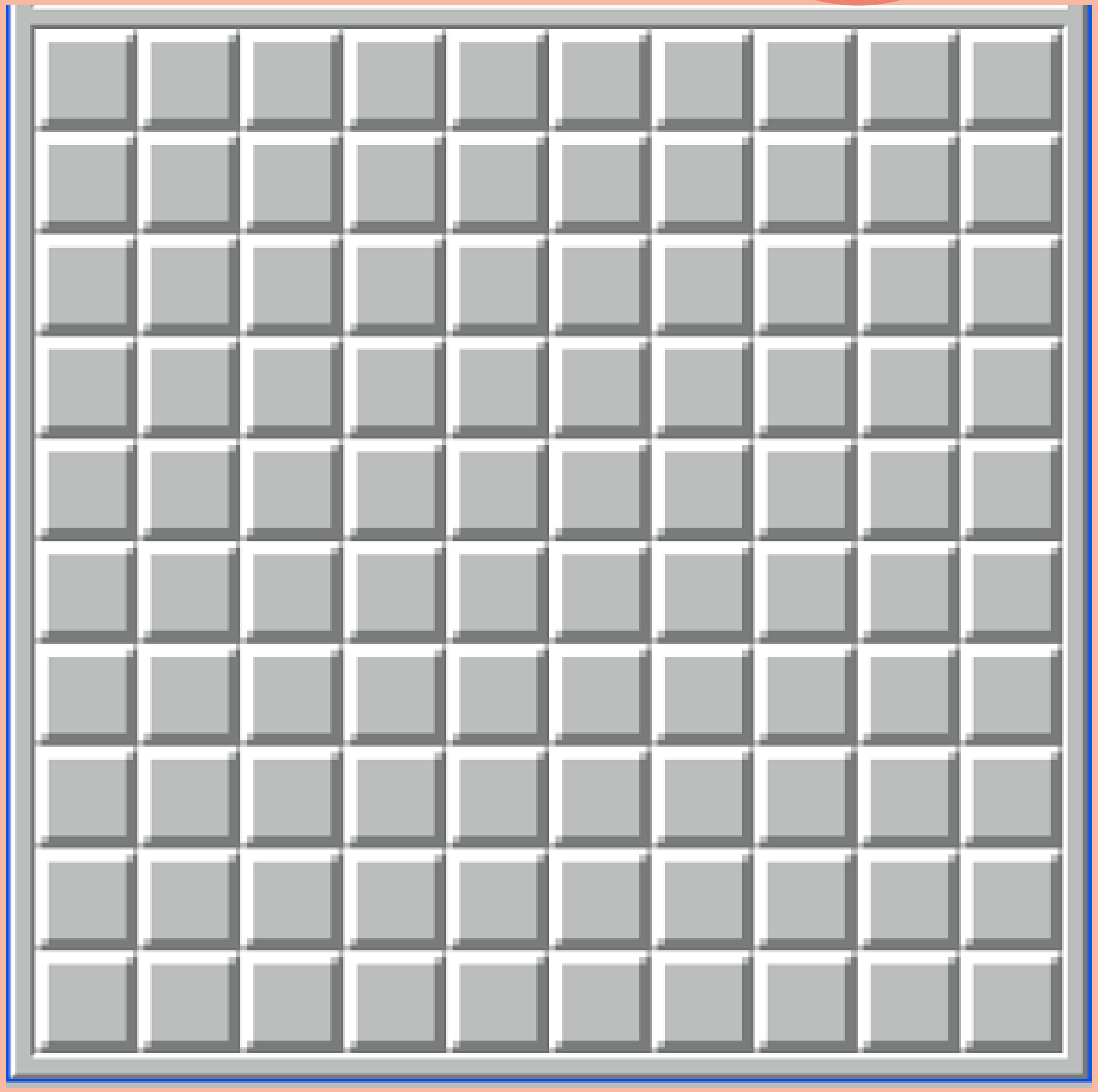


- **board** is **10 x 10** with **10** randomly placed mines
- **variables**- 100 squares
- **Domains** are **{-1,0}** : "mine" =-1, "safe"=0
- each square has an **indicator**(from **{0,...,8}**)
- multiple components
- possible to encounter a mine at the first click
- Start at (1,1) (if indicator != 0 => click(n,n), again indicat. !=0 => uncover corners of the board
- **Component 1**(till 12th iter.)- uncover corners when not having insight about "safe" squares
- **Component 2**- if indicat. ==0 => uncover all neighbors
- **Component 3-** if indicat. == #of covered neighb. => Flag them

OUR APPROACH TO MINESWEEPER

Algorithm

- **Component 4** (for optimal and fast search)-
find supersets-subsets among the constraints (of squares with Manh. dist. at most 2),
new constr' = superset-subset,
indic.(superset')=indic(superset)-indic(subset)
remove subset from superset
- **Component 5** Exhaustive search of the remaining constraints. All possible universes (permutations) are accounted for, the square with least # of universes where it contiains a mine is selected



NOW SHOWING

Demonstration of the game:
https://youtu.be/MN8mYPZc9Wg

Time Complexity

O(2^r) because of the exhaustive search, where r is the #of covered squares left

O(n^4) where n is the length of the board

Accuracy

10/10 generated game instances were solved correctly

Thank you!