CZECH TECHNICAL UNIVERSITY IN PRAGUE

FACULTY OF ELECTRICAL ENGINEERING

DEPARTMENT OF CYBERNETICS

BACHELOR'S THESIS

Algorithms for Minesweeper Game Grid Generation

JAN CICVÁREK

Bachelor Project Supervisor: MSc. Štěpán Kopřiva, MSc.

May, 2016

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

BACHELOR PROJECT ASSIGNMENT

Student: Jan C i c v á r e k

Study programme: Open Informatics

Specialisation: Computer and Information Science

Title of Bachelor Project: Algorithms for Minesweeper Game Grid Generation

Guidelines:

1. Study the game of minesweeper, problem definition and complexity.
2. Study the constraint satisfaction problem and other relevant techniques.
3. Formalize the problem of solving the game and generating the game grid.
4. Propose an algorithm for solving the game when solvable, with emphasis on CPU time.
5. Implement the algorithm described above.
6. Evaluate the algorithm on different game instances.
7. Adapt the algorithm for gradual generation of solvable grid.
8. Evaluate the algorithm on different mine densities and grid dimensions.

Bibliography/Sources:

[1] Stuart Russel, Peter Norvig – Artificial Intelligence: A modern approach, 2nd edition - 2003
[2] Richard Kaye - Infinite versions of minesweeper are Turing complete - Birmingham, 2007
[3] Ken Bayer, Josh Snyder and Berthe Y. Choueiry - An Interactive Constraint-Based Approach to Minesweeper . - University of Nebraska-Lincoln, 2006

Bachelor Project Supervisor: MSc. Štěpán Kopřiva, MSc.

Valid until: the end of the summer semester of academic year 2015/2016

L.S.

doc. Dr. Ing. Jan Kybic
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, January 14, 2015

Abstrakt

Minesweeper je videohra z roku 1990. Nalezení řešení jedné její instance nebo důkaz jeho neexistence je NP úplný problém. V této práci prozkoumám algoritmy, které tento problém řeší v polynomiálním nebo exponenciálním čase s různou úspěšností. Implementuji svůj vlastní algoritmus s důrazem na vysokou úspěšnost a využitelnost při generování pole. Nakonec také implementuji algoritmus, který je schopný generovat pole hry minesweeper, které je vždy řešitelné a zavedu nové hodnocení obtížnosti, které tento algoritmus využívá.

NP úplné a NP těžké problémy jsou velmi frekventované, lze se s nimi setkat při zajišťování kybernetické bezpečnosti, vývoji nových léků, alokaci zdrojů nebo například při obecném prohledávání stavového prostoru. Hodně NP problémů jde řešit pomocí algoritmů s polynomiální složitostí, které je řeší s vysokou úspěšností, ale nikomu se nepodařilo dokázat, že lze NP problémy v polynomiálním čase vyřešit deterministickým automatem nebo naopak možnost řešení deterministicky v polynomiálním čase vyloučit, proto je každé jejich studium přínosné.

Abstract

Minesweeper is a videogame, first introduced in the year 1990. To find a solution for one instance of this game, or prove that it does not exist, is an NP-Complete problem. In this thesis, I will introduce algorithms that can solve this problem in either polynomial or exponential time with varying success rates. I will implement my own solver, with emphasis on high success rate and its possible utility in gradual generation of the minefield. I will also implement an algorithm that can generate a minefield that is always solvable and introduce new difficulty rating system that can also be used as an input for this new minefield generator.

NP-Complete and NP-Hard problems are very common. We need to solve them in cyber-security, when developing new medicine, optimizing resource allocation, or just when searching a statespace. There are many algorithms that can search for solutions to the NP problems in polynomial time with high success rate, but the NP problems have never been proved to be solvable with deterministic automata in polynomial time or proven to be unsolvable in polynomial time with deterministic approach. For that reason, any study can help us understand these problems better.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principu při přípravě vysokoškolských závěrečných prací.

V Hradci Králové dne 24. 5. 2016 .....

Contents

1 What is minesweeper?

Minesweeper is a small puzzle-solving single-player video game originally developed by Robert Donner between years 1989 and 1990 [4]. Game was based on videogame Mined-Out developed by Ian Andrew and released in 1983 [4]. Minesweeper game is defined by grid, whose dimensions could be defined before the game starts. Dimensions don't have to be similar but minimum size of nine is applied for both, there is no theoretical upper bound, minesweeper played on infinite grid could be solved by Turing machine [1]. In this paper, I will only work with finite grid with no upper bounds. The grid is filled with squares, each containing two information. First is number of mines in vicinity, ranging from zero to eight. Second information tells us whether the square contains a mine. Player can make move step by asking whether a square contains a mine. If it does, the game ends and the player losses, if it doesn't, player obtains number of mines in vicinity of chosen square. At the beginning of the game, player knows the size of the grid, number of squares containing a mine and that square chosen in first move will not contain a mine. Goal of the game is to select all squares containing no mine and avoid all containing one. Solving this problem is NP-complete [2] and solvability without guessing is not guaranteed for every grid generated. This is big problem when instead of short time relaxation, player might get frustrated with problem he cannot possibly solve. Each successfully solved game is evaluated by two conditions. First is time from first move, second is 3BV [4], which counts the number of moves needed to solve the game. This allows minesweeper to be played competitively and to appear on various Tournaments [4].

1.1 Formal definition

Formal definition could prove more difficult than expected. First, we need to determine what is our goal. It could be locating all the squares with the mine, determining solvability of the grid or to find a next move. In this case, we will take the player perspective, to better explain the game in itself.

1.1.1 Game parameters

These parameters are not known to the player but are used to generate world with successor function.

$$n, m \in N$$

$$n, m > 0$$

$$S = \{s_{1,1}, .., s_{1,m}, .., s_{n,m}\}$$

$$s_{i,j} = \{0, 1\}$$

$$V = \{v_{1,1}, .., v_{1,m}, .., v_{n,m}\}$$

$$v_{i,j} = \sum\nolimits_{max(\|i-i_2\|,\|j-j_2\|)=1} s_{i_2,j_2}$$

$$3BV = \{1, .., m * n\}$$

Where $s_{i,j}$ tells whether square on the grid contains mine, 0 for clear square and 1 for mine. $v_{i,j}$ is sum of $s_{i,j}$ of the adjacent squares. 3BV is used to determine difficulty[4].

1.1.2 Initial state

$$X = \{x_{1,1}, .., x_{1,m}, .., x_{n,m}\}$$

$$x_{i,j} = \{-1, 0, 1\}$$

$$\forall i, j \; x_{i,j} = -1$$

$$M = \sum\nolimits_{i,j} s_{i,j}$$

Where $x_{i,j}$ after initiation, each square has value assigned -1, which signifies that it is undecided. Value 0 is for clear square and 1 for square with mine. M determines the quantity of mines on the grid.

1.1.3 Goal test

Victory

$$\sum\nolimits_{i,j} s_{i,j} = \sum\nolimits_{i,j} x_{i,j}$$

$$x_{i,j} = \{0, 1\}$$

Loss

$$\exists i, j \quad x_{i,j} = 1 \wedge s_{i,j} = 1$$

1.1.4 Successor function

$$succ(x_{i,j}, t) > x_{i,j} = t$$

$$t \in Z$$

$$t \in < -1, 1 >$$

Each change to the grid counts as one step with arc cost equal to 1.

2 Generating solvable grid

In this paper, I will be trying to find way to generate a grid that could be solved without guessing. This means, the after each move, when game isnt won, the player must have enough information to find one unrevealed square containing no mine. This problem has been approached before. Mines-Perfect offers multiple solutions [3]. When current knowledge doesn't allow to determine next safe move, the program either adds additional information (1), lets you choose mine and still continue in the game (2), rearranges the mine, so your move is correct (3) or solves part of the game for you (4). Each of this modes has it's disadvantages and deviate from the original rules. In options 1,2,3 the player must prove that there is no safe move, which increases complexity. In option 4, part of the game is solved automatically, not allowing player to play. All of these approaches require implementation of solver that identifies the standoff state. More advanced approach could be seen in Mines by Simon Tatham [5]. Random grid is generated and solver ensures no standoff state. Solver in here incorporates many approaches, greedy approaches with lower complexity that are not guaranteed to solve for all solvable grids and Depth-first search based algorithm when greedy approaches fail. Due to time complexity, after n tries (100 is basic), 3BV value is lowered. This means that in mean of 3BV of this generator and random generator that only excludes unsolvable games are different, leading to less complex games. It is apparent that solver is very important, when generating solvable grid.

3 Minesweeper solver

We already know that solving minesweeper game is an NP-complete problem [2]. In contradiction to most publicized solvers, we are not looking for solution with best success rate, we simply want to know if it is necessary to guess in order to solve the grid. For that reason, our solver doesn't have to work with Markov chains in state space, dealing with probabilities. We are only looking for deterministic state space since we are only interested in 100

3.1 Single point solvers

These algorithms may be perceived as constrain satisfaction problem without constrain propagation and if there is optimal solution, it is not guaranteed to find it. Each iteration of the algorithm consists of two steps.

- For every clear square, the algorithm compares number of mines in vicinity and number of marked mines in vicinity. If they equal and there are any undecided neighbours, algorithm marks undecided neighbours as clear.
- For every clear, the algorithm square compares the number of undecided neighbour squares and unrevealed neighbour mines. If the two numbers are equal, algorithm marks all undecided neighbours as mines.

For each step of this algorithm, we have to go thru all of the clear squares. But number of these squares is always below m*n and computing time needed to resolve

each of these squares within one step is constant. Thanks to this, we can make every step in polynomial time. This algorithm terminates when the grid doesn't change after one step. In this case, we have to apply more complex solver, but after one step of more complex solver, we can go back to single point solver and repeat this until solution is found or we can say that there is no safe solution. Since we will be using this algorithm in our solver, here is pseudocode of the implementation:

```
initialization;
while action made in last iteration do
    for i = 0 to Height do
        for j = 0 to Width do
            if clear square then
                if neighbour mines==marked neighbour mines &&
                undecided ≥ 0 then
                    reveal neighbours
                end
                if neighbour undecided==unmarked neighbour mines &&
                undecided ≥ 0 then
                    mark neighbours
                end
            end
        end
    end
end
```

Algorithm 1: Single point strategy

3.2 Depth-first search

Not very effective algorithm, but its concept makes it ideal for situation where we rely on number of total mines as a constrain. Also behaves very similar to constrain satisfaction problem in certain situation. When CSP needs to take more constrains in consideration to achieve consistency. This algorithm is more viable for collecting data. Mostly differences between iterations and procedures to solve fields that differ only in one mine.

3.3 Constrain satisfaction problem

There are many ways to implement this problem. Especially in respect to arc consistency and production of constrains. Ken Bayer, Josh Snyder and Berthe Y. Choueiry use increasingly complex arc consistency algorithms, but don't go exploring all the possible mine layouts in the entire field [8]. While this implementations is very efficient, it could fail to find deterministic solution. Chris Studholme, PhD builds constrains for each square in game and proceeds until viable layout is found [6]. Although, when layout in compliance with the constrains is found, it is not guaranteed to be the only one and since game accepts only one of the layouts, we have to find the other layouts that satisfy the constrains. Studholme to calculate Markov probabilities for these layouts,

which we don't have to do, since we only interest ourselves in 0 and 1 probabilities, where 0 is square that was clear in every layout and 1 is square that was mine in every layout. If we find at least one, we could proceed to another iteration, else we have just proved that there is more than one admissible solution, which is more than game accepts and grid is thus unsolvable.

3.4 Equation Strategy

Equation strategy was introduced by John D. Ramsdell in Programmers Minesweeper [7]. Ramsdell utilizes fact, that number of each clear square is sum of its neighbours, clear squares having 0 and mined ones having 1. Using this knowledge, he creates set of linear equations for each undecided square with any clear neighbouring squares. Similarly as with constrain satisfaction problem, we could run into underdetermined system which is in many ways similar to multiple layouts in compliance with generated constrains. It is not surprising, considering that equation strategy's approach resembles that of constrain satisfaction problem, if we look on the linear equations as on constrains.

Studholme's implementation of constrain satisfaction problem was able to achieve comparable, even slightly better results, with as much as twenty times less CPU time needed [6], which doesn't prove that equation strategy is inferior as it could be caused by implementation.

4 Technical background of my solution

Minesweeper is drastically different from most problems, where CSP is used.
When new game is initialized, the number of correct solutions based on available information is

$$PSs = \frac{(n*m)!}{(n*m-\sum_{i=0}^{n}\sum_{j=0}^{m} s_{i,j})! * (\sum_{i=0}^{n}\sum_{j=0}^{m} s_{i,j})!}$$

In advanced setting, where n = 16, m = 30 and $\sum_{i=0}^{n}\sum_{j=0}^{m} s_{i,j}$ = 99, we get approximately $5.6 * 10^{104}$ correct solutions.
This means, that while the problem definition written above is correct, an algorithm that would try to employ it would have little chance of executing in realistic time.
The reason why minesweeper is still solvable is that it will provide additional information almost every time the solver assigns 0 (not mine) to any field.
Solvability is compromised, when the current board only gives enough information to prove one field does not hold a mine and by assigning 0 to that field no additional information is gained.

4.1 Single point strategy and its variations

This approach offers very good computation times, but doesn't search the whole statespace, because it only uses simple constraints. I've decided to implement this algorithm

as a first stage of my solver for its speed, but later omitted it, as it wasn't faster than my the more complicated algorithm that was able to detect these easy passable successor functions during the process of building the sets I use in when no simple solution was find, rendering this algorithm redundant. It is still used in the Matlab GUI demonstration application, because of its simplicity.

4.2 Depth-first algorithms

Because of the number of passable solutions, this algorithm would need to cut branches that would lead to solution, in order to execute in realistic time. time. This algorithm will always find a solution, if there is one, but would often explore greater depths than what is required[11] increasing CPU time and preventing it to be used in a real-time application. It is very easy to implement and could be used to solve the beginner (9 by 9) field, but its low efficiency is not justifiable.

4.3 Breadth-first algorithms

This approach respects the nature of a Minesweeper, but is not sophisticated enough to allow for easy heuristics to be implemented. I haven't seen any implementation of this algorithm for Minesweeper and didn't attempt one myself since I didn't consider it optimal. But my final algorithm does use the basic idea of depth dependent analyse of the statespace.

4.4 Constraint satisfaction algorithms

Minesweeper gameplay feels very much like a constraint satisfaction problem. Fields with numbers provide information on adjacent fields and next move must be made so that it satisfies these constraining fields. But when solving an constraint satisfaction problem, we have an unchanging set if variables, which would be fields, set of domains, mine and safe in the case of Minesweeper and the constraints. Incidentally, in Minesweeper, they are provided to the solver based on every step he has done since the game initiation. This causes the game to change constantly and prevents general CSP algorithms to be utilized.

4.4.1 Multiple constraint satisfaction problems

While the constrain satisfaction algorithm seems to be one of the best methods of solving the minesweeper grid, my goal is to generate a solvable grid. This could be done through trial and error, but the efficiency would not be suitable for a real time use.

success rates with advanced grid, randomly generated

Algorithm	maps solved	success rate
Single point strategy	0	0%
Equation strategy	64	6.4%
CSP Studholme	71	7.1%
My CSP	76	7.6%

This test was done on a sample of 1000 advanced maps. Each had the width of 30, height of 16 and 99 mines. The maps were the same for all the algorithms. The algorithms started from a same corner and in case the first constraint field wasn't 0, the map was discarded and another one generated. Single point strategy and Equation strategy were capped at 10seconds, my CSP had cap of 2 seconds to make a move and 47 milliseconds to evaluate one step on current depth

success rates with intermediate grid, randomly generated

Algorithm	maps solved	success rate
Single point strategy	1	0.1%
Equation strategy	502	50.2%
CSP Studholme	502	50.2%
My CSP	637	63.7%

This test was done on a sample of 1000 intermediate maps. Each had the width of 16, height of 16 and 44 mines. The maps were the same for all the algorithms. The algorithms started from a same corner and in case the first constraint field wasn't 0, the map was discarded and another one generated. Single point strategy and Equation strategy was capped at 10seconds, my CSP had cap of 2 seconds to make a move and 47 milliseconds to evaluate one step on current depth

Based on these results, trial and error could be implemented for generating an intermediate minefield, but would not work in real-time application for advanced one. Because of these results, I've decided to use a gradual generation and placing the mines one by one, not all at once at random locations. This means that a lot of potential repetition, that should best be avoided. That is why my solver doesn't solve the whole minefield, but new CSP, with clearly set variables, domains and constraints, for every step. This will increase CPU time, but not complexity, as the number of steps grows linear with the number of fields.

4.4.2 Gradual CSP with MRV and FVC heuristics

I wanted to avoid inadmissible heuristics, to avoid discarding solvable fields. Minimum remaining values generally significantly lowers computation time[12] so I chose it as a method of selecting which variables to try first. Forward checking was needed, because the number of possible solutions in the 10^{100} realm would be impossible to evaluate and FCH shows good results in most use cases[10][12].

5 Implementation of the solver

5.1 Adapting the minefield

First step is cheaply removing all the fields adjacent to constraint nodes with value 0. The main algorithm can do this, but the Map is equipped with O(n*m) function necessary for the GUI client, and is cheaper than the CSP of depth 1, which is O($n^2 * m^2$).

The size of the statespace is dependent on a factorial of the number of elements, also variables in CS problems. For this very reason, the second and most important step is reducing the number of variables and the number of constraint fields.

```
initialization;
for each field in MineMap do
    if field X_{i,j} == MARKED then
        field(X_{i-1; x+1,j-1; j+1}) = field(X_{i-1; x+1,j-1; j+1}) - 1;
        remove(X_{i,j});
    end
    if field(X_{i,j}) == 0 then
        if reveal neighbours == success then
            continue(main loop);
        end
        remove(X_{i,j});
    end
    if field(X_{i,j}) ¿ 0 then
        addConstraint(X_{i,j});
        addNewNodesAround(X_{i,j});
    end
    if field X_{i,j} == UNPROBED && NodeList.contains(X_{i,j}) == false then
        field X_{i,j} = unconstrained variable;
    end
end
```

Algorithm 2: Adapting the minefield

With the use of this algorithm, we can cut everything except for the constraint fields and variables that are on the border of unvisited and probed fields. We also mark the location of unconstrained variables for later use.

5.2 Forming a cluster

With exponential complexity, every way that decreases the size of the set of variables is highly advantageous. Two clusters are sets of constraint fields and nodes, where no node from the first set is adjacent to any constraint from the second set, no constraint from the first set is adjacent to any node in the second set and vice versa.

```
0 | 0 | 0 | 2 | M | * | * | * |
0 | 0 | 0 | 2 | M | 3 | 2 | 1 |
0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
2 | 2 | 1 | 0 | 0 | 0 | 0 | 0 |
M | M | 1 | 0 | 0 | 0 | 0 | 0 |
* | 3 | 1 | 0 | 0 | 0 | 0 | 0 |
* | 2 | 0 | 0 | 0 | 0 | 0 | 0 |
* | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
```

Figure 1:*Minefield with two clusters after several steps*[1]

Minefield on Figure 1 contains two clusters, each with 3 constraint nodes and 3 variables. This problem can be solved at depth 1.

[1]Figure 1 can be passed onto any solver compliant with PGMS[7] as a minefield, see section 10 for more information

5.3 The main CSP

The algorithm iterates through all the depths starting with one and ending with global limit, that could be set as a variable when calling the play function.
The current depth signifies the number of guesses we make when trying to find a safe step. If the depth is higher We get a problem that can be defined as a CSP problem.

5.3.1 Theoretical part

```
    0     1     2     3
0   0  |  0  |  0  |  0  |
1   0  |  0  |  0  |  0  |
2   1  |  1  |  0  |  0  |
3   *  |  2  |  0  |  0  |
4   *  |  2  |  0  |  0  |
5   1  |  1  |  0  |  0  |
```

Figure 2:*Simple minefield*[2]

The minefield on Figure 2 can be defined as a CSP, with these parameters:

$$Variables : x_{0,3}, x_{0,4}$$

$$Domains : D_i = 0, 1$$

$$Constraints : x_{0,3} = 1, x_{0,4} = 1, x_{0,3} + x_{0,4} = 2$$

[2]Figure 2 can be, with slight modifications, passed onto any solver compliant with PGMS[7] as a minefield, see section 10 for more information

```
Data: Successor function:
for each variable in current cluster do
    add undecided most constrained variable x on activeList;
    permutations(activeList);
end
```

Algorithm 3: Pseudocode of a successor function

```
Data: Permutations with repetition function:
while permutation is max == false do
    if last var is max == false then
        Increase last variable;
        push the new permutation;
    end
    if last var is max == true then
        for every variable in permutation from back do
            if current variable is max == true then
                current variable = 0;
            end
            Ifcurrent variable is max == false increase current variable;
            push the new permutation;
            continue;
        end
    end
end
```

Algorithm 4: Pseudocode of a permutations with repetition function

But such an approach makes it very difficult to implement heuristics respecting the nature of minesweeper, for that reason, my algorithm generates the constraints dynamically from list of constraint fields.

5.3.2 Deciding if a move is the correct one

Instead, I make use of the small domain size in minesweeper. Every field is either a mine or safe field. For depth n, I choose n fields and go through all their permutations with repetition, which means 2^{depth} instances. I use SPS algorithm in them and throw out the instances that contradict themselves.
For example, this CS problem has 3 variables and 5 constraint fields.

```
   0   1   2   3
0  0 | 0 | 0 | 0 |
1  0 | 0 | 0 | 0 |
2  0 | 0 | 0 | 0 |
3  1 | 1 | 1 | 0 |
4  * | * | 1 | 0 |
5  * | * | 1 | 0 |
```

Figure 3:*Resolution step 1* [3]

Variable $x_{1,4}$ is picked first and gets assigned the value C (safe field). SPS is run to check consistency by cutting domain sets of active variables and find contradictions.

```
   0   1   2   3
0  0 | 0 | 0 | 0 |
1  0 | 0 | 0 | 0 |
2  0 | 0 | 0 | 0 |
3  1 | 1 | ! | 0 |
4  H | C | 1 | 0 |
5  * | H | 1 | 0 |
```

Figure 4:*Resolution step 2* [3]

Constraint fields with least active nodes are resolved first. For that reason $x_{0,4}$ and $x_{1,5}$ get assigned the H (unsafe) value.
When arc consistency reaches constraint field at (2,3), there is a contradiction and whole resolution gets discarded.

```
   0   1   2   3
0  0 | 0 | 0 | 0 |
1  0 | 0 | 0 | 0 |
2  0 | 0 | 0 | 0 |
3  1 | 1 | 1 | 0 |
4  C | H | 1 | 0 |
5  * | C | 1 | 0 |
```

Figure 5:*Resolution step 3* [3]

We resolve the next permutation with repetition. This time $x_{1,4}$ gets assigned the H value. Resolution then assigns C to $x_{0,4}$ and $x_{1,5}$, the arc consistency successfully finishes and the solution is added on the solutions set.

[3]Figures can be, with slight modifications, passed onto any solver compliant with PGMS[7] as a minefield, see section 10 for more information

Now our solution set of sets looks like this: $\{ \{ x_{1,4}, x_{0,4}, x_{1,5} \} \}$. Since we are at depth 1 and one set got discarded, we have only $2^1 - 1 = 1$ sets.
We can't really prove any single one of our guesses to be true, but since we have assigned all the possible permutations with repetition to the selected variable, we can say for sure, that we did resolve for a correct guess
Now we check, if there is any node that has been assigned the same value by all the undiscarded(retained) solution sets, if that is so, it also means it is a product of resolution based on a correct guess and it is safe to play implement all these domain cuts for all of the resolved variables. In Figure 5, variables $x_{1,4}$,$x_{0,4}$ and $x_{1,5}$ all had their domain set size reduced, so we play all these changes, ending up with this gamestate:

```
   0   1   2   3
0  0 | 0 | 0 | 0 |
1  0 | 0 | 0 | 0 |
2  0 | 0 | 0 | 0 |
3  1 | 1 | 1 | 0 |
4  1 | M | 1 | 0 |
5  * | 1 | 1 | 0 |
```

Figure 6:*Resolution is complete, we get new CSP*

With a new CSP, we start the whole process over.

5.3.3 Using number of mines when main algorithm fails

Upon initiation, solver is provided with a total number of mines on a minefield. Marking of mines is not a part of a win condition, but it is another constraint, I have been avoiding so far.
Since the aim of this algorithm is to ultimately provide a solvable minefield using gradual generation, number of mines needs to be clearly separated from the rest of the resolution process.
When gradually building a minefield, situation where some variables will never get into our active set, will be more common, than it is in the set of solvable minefields.
There are two main reasons for this, both of similar nature. To explain them, I need to first explain the two distinct situations, where the total number of mines could be used to advance a game minesweeper game. With both cases, the concept of clusters is unused and all the border variables are seen as active.

5.3.4 Advancing by counting total mines, all mines on border

First type of advancing by counting the total number of mines occurs when all the remaining mines lay on the border variables, those are the variables that have any constraints placed on them (apart from total mine count constraint).
There can be any number of clusters and any number of possible placement of mines, as long, as all the possible permutations with repetition place all the remaining mines on the border, we can safely assign 0 (safe) to all unconstrained variables. This although does not guarantee solvability, as the unconstrained mines are not guaranteed

to provide information that would contradict all but one permutation of border mine domain assignments.

5.3.5 Advancing by counting total mines, all inner variables are mines

The second solvable type occurs, when all the mines that are not constrained, are mines. Again, there can be any number of clusters. All the passable permutations of the border mines must contain (remaining mines - unconstrained variables). Any permutation where border fields have less than (remaining mines - unconstrained variables) contradicts the rules, as all minds could not be places. Any passable permutation where border fields have more than (remaining mines - unconstrained variables) but less then the situation in previous section (all mines on border) are unsolvable.

5.3.6 Advancing by counting total mines, examples

This is an example that cannot be solved without the use of counting the total number of mines:

```
0 | 0 | 0 | 0 | 0 | 2 | X | * | * | * | * |
2 | 3 | 2 | 1 | 0 | 2 | X | X | X | X | 2 |
X | X | X | 1 | 0 | 1 | 2 | 3 | 3 | 2 | 1 |
3 | * | 3 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
1 | * | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
1 | * | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
3 | * | 3 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
X | X | X | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
2 | 3 | 2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
```

Figure 7:*Unsolvable without minecount*

The count of remaining mines in figure 7 is in the range from 3 to 5, less than 3 would not satisfy the field constraints and more than 5 could not be fitted onto unvisited fields while still respecting the field constraints. The constrained variables hold 3 mines in every passable permutation, of which there are 4 (two permutations per each cluster). If the number of remaining mines is 3, we know that all the unconstrained variables are safe, which provides additional information and we can continue the resolution further, ending at the state:

```
0 | 0 | 0 | 0 | 0 | 2 | X | 4 | 4 | X | 2 |
2 | 3 | 2 | 1 | 0 | 2 | X | X | X | X | 2 |
X | X | X | 1 | 0 | 1 | 2 | 3 | 3 | 2 | 1 |
3 | * | 3 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
1 | * | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
1 | * | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
3 | * | 3 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
X | X | X | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
2 | 3 | 2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
```

Figure 8:*Unsolvable with minecount*

We have uncovered additional fields, but ultimately did not gain the information needed to win.

If the number of remaining mines is 5, we know that all the unconstrained variables are mines, which allows us to make a play and further the progress in a game, but does not provide additional information and we are unable to uncover any additional fields.

```
0 | 0 | 0 | 0 | 0 | 2 | X | X | X | * | * |
2 | 3 | 2 | 1 | 0 | 2 | X | X | X | X | 2 |
X | X | X | 1 | 0 | 1 | 2 | 3 | 3 | 2 | 1 |
3 | * | 3 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
1 | * | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
1 | * | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
3 | * | 3 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
X | X | X | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
2 | 3 | 2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
```

Figure 9:*Unsolvable with minecount*

If the number of mines is between 3 and 5 (only 4 in this example), we can do nothing, since there are mines among the unconstrained fields, but they can be in any configuration.
In cases where the number of mines on the border could not be determined even after cutting the configuration exceeding or not reaching the limit amounts, the problem is unsolvable, as there are at least two solutions compliant with the constraints and there is only one correct solution in minesweeper.

5.4 Admissible heuristics

5.4.1 Most constrained variable

This fail-first heuristic allows me to recognize contradiction quickly and not effectively decrease the size of the statespace.

```
0 | 0 | 0 | 0 | 0 |
0 | 0 | 0 | 0 | 0 |
0 | 0 | 1 | 2 | 2 |
0 | 0 | 2 | * | * |
0 | 0 | 2 | * | * |
```

Figure 10

In the example shown on figure 10, the corner variable ($x_{3,3}$) will be resolved for first. Since there is most constraints on it, the chance of a contradiction is higher, and indeed, assigning C (safe) value leads to a quick contradiction with constraint field ($x_{2,2}$), making the whole resolution more effective.

5.4.2 Least restrictive field

As I have said before, I don't use a set of constraints, although I have defined this set, but in practice, I generate it dynamically. The fewer variables limited by the constraint field, the bigger the chance that either resolution or contradiction will appear, allowing me to save computation time again.
When we go back to figure 10, thanks to this heuristic, the first constraint field after trying to assign a domain to $x_{3,3}$ is $x_{2,2}$, that also gives us contradiction right away.

5.4.3 Variables bound to other variables

To prove there is no solution on given depth in a single cluster, I iterate through all the combinations (size = depth) of the active variables in the actual cluster. For each of these combinations, I have to go through all the permutations with repetition. This process is costly, it is the reason why my algorithm has exponential complexity. But some of the combinations aren't perspective.

```
0 | 1 | M | 1 | 0 | 0 | 0 |
0 | 1 | 2 | 2 | 1 | 0 | 0 |
0 | 1 | 2 | M | 1 | 0 | 0 |
0 | 1 | M | 2 | 1 | 0 | 0 |
1 | 2 | 1 | 1 | 0 | 0 | 0 |
M | 1 | 0 | 0 | 0 | 0 | 0 |
1 | 1 | 0 | 0 | 0 | 0 | 0 |
0 | 0 | 0 | 0 | 0 | 0 | 0 |
0 | 0 | 0 | 0 | 0 | 0 | 0 |
0 | 0 | 0 | 0 | 0 | 0 | 0 |
```

Figure 11:*minefield, requires multiple guesses*

The minefield on figure 11 needs to go to depth 2 to be successfully resolved by my CSP implementation. On depth 2 there are $\frac{8!}{2!*(8-2)!} = 28$ combinations, each with 4 permutations with repetition. This number of combinations is unnecessary and can be cut down.
For example variables $x_{2,0}$ and $x_{2,1}$ are bound together. When one is assigned the domain C (safe) the other gets H (unsafe) and vice versa. I can detect it at the cost of n (8 in this case) when searching for solutions in depth 1. When I pick a variable and try to assign one and then the other domain, whenever there is a node that gets its domain size reduced both times, I remember it and in the higher depths, I check that my current combination doesn't have any variables that have each other blacklisted. When the blacklist of variable A contains variable B, blacklist of variable B contains variable A, which makes the checking easier. When I encounter this conflict, I can discard the whole combination, since I already know that this one will resolve the same like the combination one depth lower, that had all the same variables, except only one of the conflicting two. In the current depth, I will just have twice as much resolutions, but half will be discarded for contradiction and the other half will be identical to the lower depth combination.

5.5 Inadmissible heuristics

Since this solvers purpose is to detect solvable minesweeper grids, I have hoped to avoid any inadmissible heuristics, but I have been unable to achieve acceptable runtime without them.

5.5.1 Time constraints

The algorithm is designed to be used in a real time application on various devices. Being able to assure a realistic runtime is crucial, that is why the algorithm can present result even when it is ended prematurely. The constraints limit how much time can be spent iterating through one combination and how much time can be spent at one depth.

5.5.2 Depth constraints

Thanks to the similarities my algorithm shares with the way humans solve this problem, depth is an excellent difficulty rating. Higher depths often get extremely computation heavy, but occur very infrequently.

maximal and average depth of my CSP

minefield	maximal depth	average depth
9x9, 10 mines	2	1.02
16x16, 40 mines	9	1.8
30x16, 90 mines	9	1.24

My CSP was capped at the depth 9, had 200 milliseconds to solve a depth and 4.7 milliseconds to evaluate one combination current depth

6 Solver evaluation

In section 4.4.1, we have already seen that other algorithms don't always detect a solvable minefield. I have proven this by solving it with my algorithm. It is improbable that my CSP implementation can find all the passable solutions, as it also uses inadmissible constraints, but I have nothing to test it against.

6.1 Example of problematic minefield for Equation strategy

```
0 | 1 | X | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * |
0 | 2 | 3 | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * |
0 | 1 | X | 3 | * | 3 | 2 | 1 | * | * | * | * | * | * | * | * | * | * | * |
0 | 1 | 3 | X | * | * | 2 | * | * | * | * | * | * | * | * | * | * | * | * |
0 | 0 | 2 | X | 3 | 1 | 2 | X | * | * | * | * | * | * | * | * | * | * | * |
2 | 2 | 2 | 1 | 1 | 0 | 1 | 3 | * | * | * | * | * | * | * | * | * | * | * |
X | X | 3 | 1 | 1 | 0 | 1 | 3 | * | * | * | * | * | * | * | * | * | * | * |
3 | X | 3 | X | 1 | 0 | 2 | X | * | * | * | * | * | * | * | * | * | * | * |
1 | 2 | 4 | 3 | 2 | 0 | 3 | X | * | * | * | * | * | * | * | * | * | * | * |
0 | 1 | X | X | 2 | 1 | 3 | X | 4 | * | * | * | * | * | * | * | * | * | * |
0 | 1 | 3 | X | 2 | 1 | X | 3 | 3 | * | * | * | * | * | * | * | * | * | * |
0 | 0 | 1 | 1 | 1 | 1 | 1 | 2 | X | * | * | * | * | * | * | * | * | * | * |
1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 2 | * | * | * | * | * | * | * | * | * | * |
X | 2 | 1 | 1 | X | 2 | 2 | 1 | 2 | 2 | * | * | * | * | * | * | * | * | * |
3 | X | 2 | 2 | 3 | X | 2 | X | 2 | * | * | * | * | * | * | * | * | * | * |
X | 2 | 2 | X | 2 | 1 | 2 | 1 | 2 | * | * | * | * | * | * | * | * | * | * |
```

Figure 12:*EQS gives up prematurely*

```
0 | 1 | X | X | 2 | 2 | 1 | 1 | 0 | 1 | 3 | X | 3 | X | 1 | 0 | 0 | 0 |
0 | 2 | 3 | 4 | X | 2 | X | 1 | 0 | 2 | X | X | 4 | 2 | 1 | 0 | 0 | 0 |
0 | 1 | X | 3 | 3 | 3 | 2 | 1 | 0 | 2 | X | 4 | X | 1 | 0 | 1 | 1 | 1 |
0 | 1 | 3 | X | 3 | X | 2 | 1 | 1 | 1 | 1 | 2 | 1 | 2 | 2 | 3 | X | 2 |
0 | 0 | 2 | X | 3 | 1 | 2 | X | 3 | 2 | 1 | 0 | 0 | 2 | X | X | 4 | 3 |
2 | 2 | 2 | 1 | 1 | 0 | 1 | 3 | X | X | 2 | 1 | 0 | 2 | X | X | 3 | X |
X | X | 3 | 1 | 1 | 0 | 1 | 3 | X | 4 | X | 1 | 0 | 1 | 3 | 3 | 3 | 1 |
3 | X | 3 | X | 1 | 0 | 2 | X | 4 | 3 | 1 | 1 | 0 | 1 | 2 | X | 1 | 0 |
1 | 2 | 4 | 3 | 2 | 0 | 3 | X | X | 1 | 0 | 0 | 1 | 2 | X | 2 | 1 | 1 |
0 | 1 | X | X | 2 | 1 | 3 | X | 4 | 2 | 1 | 0 | 1 | X | 2 | 1 | 0 | 1 |
0 | 1 | 3 | X | 2 | 1 | X | 3 | 3 | X | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 2 |
0 | 0 | 1 | 1 | 1 | 1 | 1 | 2 | X | 3 | 2 | 0 | 1 | 1 | 2 | 1 | 1 | 1 |
1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 2 | X | 2 | 2 | 2 | X | 3 | X | 1 | 2 |
X | 2 | 1 | 1 | X | 2 | 2 | 1 | 2 | 2 | X | 3 | X | 3 | X | 2 | 1 | 1 |
3 | X | 2 | 2 | 3 | X | 2 | X | 2 | 2 | 3 | X | 2 | 3 | 2 | 3 | 2 | 4 |
X | 2 | 2 | X | 2 | 1 | 2 | 1 | 2 | X | 2 | 1 | 1 | 1 | X | 2 | X | X |
```

Figure 13:*my CSP can solve this problem*

The minefields aren't complete and are missing the few of the columns on the right, as I've run out of screen space, but EQS only has unknown variables there and my algorithm has solved the whole minefield with only ever getting to depth 2.
Depth 2 is something even fairly inexperienced player can solve and should not be omitted from the results.

6.2 Example of problematic minefield for other CSP

```
0 | 0 | 0 | 0 | 1 | X | 2 | X | 1 | 0 | 0 | 2 | X | * | * | * | * | * |
1 | 1 | 0 | 0 | 1 | 1 | 3 | 2 | 3 | 1 | 1 | 2 | X | * | * | * | * | * |
X | 1 | 0 | 1 | 1 | 1 | 2 | X | 4 | X | 1 | 2 | 3 | * | * | * | * | * |
3 | 3 | 1 | 1 | X | 1 | 2 | X | X | 2 | 1 | 2 | X | * | * | * | * | * |
X | X | 3 | 2 | 2 | 1 | 1 | 3 | 4 | 3 | 1 | 2 | X | 3 | * | * | * | * |
3 | X | 3 | X | 1 | 1 | 1 | 2 | X | X | 1 | 1 | 1 | 2 | * | * | * | * |
* | 2 | 2 | 1 | 1 | 2 | X | 4 | 3 | 3 | 1 | 1 | 1 | 3 | * | * | * | * |
* | 4 | 2 | 1 | 0 | 2 | X | 3 | X | 1 | 0 | 2 | X | 4 | * | * | * | * |
X | X | X | 1 | 0 | 1 | 1 | 2 | 1 | 1 | 0 | 2 | X | 4 | * | * | * | * |
X | X | 3 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 2 | * | * | * | * |
3 | 3 | 3 | 1 | 1 | 1 | 2 | 2 | 1 | 0 | 0 | 1 | 2 | 3 | * | * | * | * |
1 | X | 2 | X | 2 | 2 | X | X | 1 | 1 | 1 | 2 | X | X | * | * | * | * |
1 | 1 | 2 | 1 | 2 | X | 3 | 2 | 1 | 2 | X | 3 | 4 | X | * | * | * | * |
1 | 1 | 1 | 0 | 1 | 1 | 2 | 1 | 1 | 2 | X | 2 | 2 | X | 5 | * | * | * |
1 | X | 1 | 1 | 2 | 3 | 3 | X | 1 | 1 | 1 | 1 | 1 | 3 | X | * | * | * |
1 | 1 | 1 | 1 | X | X | X | 2 | 1 | 0 | 0 | 0 | 0 | 2 | X | * | * | * |
```

Figure 14:*CSP gives up prematurely*

```
0 | 0 | 0 | 0 | 1 | X | 2 | X | 1 | 0 | 0 | 2 | X | X | 1 | 0 | 0 | 1 |
1 | 1 | 0 | 0 | 1 | 1 | 3 | 2 | 3 | 1 | 1 | 2 | X | 4 | 2 | 1 | 0 | 1 |
X | 1 | 0 | 1 | 1 | 1 | 2 | X | 4 | X | 1 | 2 | 3 | 4 | X | 1 | 0 | 2 |
3 | 3 | 1 | 1 | X | 1 | 2 | X | X | 2 | 1 | 2 | X | X | 2 | 1 | 0 | 2 |
X | X | 3 | 2 | 2 | 1 | 1 | 3 | 4 | 3 | 1 | 2 | X | 3 | 1 | 0 | 0 | 2 |
3 | X | 3 | X | 1 | 1 | 1 | 2 | X | X | 1 | 1 | 1 | 2 | 1 | 1 | 0 | 1 |
2 | 2 | 2 | 1 | 1 | 2 | X | 4 | 3 | 3 | 1 | 1 | 1 | 3 | X | 3 | 1 | 0 |
X | 4 | 2 | 1 | 0 | 2 | X | 3 | X | 1 | 0 | 2 | X | 4 | X | X | 2 | 0 |
X | X | X | 1 | 0 | 1 | 1 | 2 | 1 | 1 | 0 | 2 | X | 4 | 4 | X | 3 | 1 |
X | X | 3 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 2 | X | 3 | X | 1 |
3 | 3 | 3 | 1 | 1 | 1 | 2 | 2 | 1 | 0 | 0 | 1 | 2 | 3 | 2 | 2 | 1 | 1 |
1 | X | 2 | X | 2 | 2 | X | X | 1 | 1 | 1 | 2 | X | X | 3 | 1 | 0 | 0 |
1 | 1 | 2 | 1 | 2 | X | 3 | 2 | 1 | 2 | X | 3 | 4 | X | X | 2 | 2 | 1 |
1 | 1 | 1 | 0 | 1 | 1 | 2 | 1 | 1 | 2 | X | 2 | 2 | X | 5 | X | 2 | X |
1 | X | 1 | 1 | 2 | 3 | 3 | X | 1 | 1 | 1 | 1 | 1 | 3 | X | 3 | 2 | 1 |
1 | 1 | 1 | 1 | X | X | X | 2 | 1 | 0 | 0 | 0 | 0 | 2 | X | 2 | 0 | 0 |
```

Figure 15:*my CSP can solve this problem*

The minefields aren't complete and are missing the few of the columns on the right, as I've run out of screen space, but the CSP by Dr. Studholme only has unknown variables there and my algorithm has solved the whole minefield with only ever getting to depth 2.
Depth 2 is something even fairly inexperienced player can solve and should not be omitted from the results.

6.3 Computation time

My algorithm doesn't have the time as the main focus, but it is still very important, unfortunately, it really isn't at all competitive when compared to Equation strategy by John D. Ramsdell and especially CSP by Dr. Studholme.

results for advanced game, 30x16 fields, 99 mines, 1000 tries

strategy	average runtime [milliseconds]	wins
Equation strategy	94.5	64
CSP Studholme	1.06	68
My CSP	206	70

The maps were the same for all the algorithms. The algorithms started from a same corner and in case the first constraint field wasn't 0, the map was discarded and another one generated. The Equation strategy had 10 seconds to make a step. My CSP was capped at the depth 9, had 200 milliseconds to solve one cluster at given depth and 4.7 milliseconds to evaluate one combination current depth

results for intermediate game, 16x16 fields, 40 mines, 1000 tries

strategy	average runtime [milliseconds]	wins
Equation strategy	4.8	508
CSP Studholme	0.04	510
My CSP	9.3	518

The maps were the same for all the algorithms. The algorithms started from a same corner and in case the first constraint field wasn't 0, the map was discarded and another one generated. The Equation strategy had 10 seconds to make a step. My CSP was capped at the depth 9, had 200 milliseconds to solve a depth and 4.7 milliseconds to evaluate one combination current depth

results for beginner game, 9x9 fields, 10 mines, 1000 tries

strategy	average runtime [milliseconds]	wins
Equation strategy	0.16	790
CSP Studholme	0.025	790
My CSP	1.47	790

The maps were the same for all the algorithms. The algorithms started from a same corner and in case the first constraint field wasn't 0, the map was discarded and another one generated. The Equation strategy had 10 seconds to make a step. My CSP was capped at the depth 9, had 200 milliseconds to solve a depth and 4.7 milliseconds to evaluate one combination current depth

The results of My CSP and EQS by John D. Ramsdell are stable in relation to each other, with my implementation taking twice as long to finish. This did not occur in the smallest 9x9 setting, but that would be to my initiation times, that set everything up for a gradual generation of solvable minefield.
CSP implementation by Dr. Stuholme seems to be a complete outliner, performing much better then the other two algorithms and scaling incredibly well with increase of variables from 9x9 to 16x16 field.
EQS by John D. Ramsdell is in its representation more of a CSP than the other two implementations, because it really works exactly with the constraints. I see that as a main reason for such a similar scaling. Once I generate the constraints using the constraint fields, we are solving almost identical problem.

7 Generator implementation

Since everything is already prepared, not much needs to be done. I start placing the mines at random, making sure I don't place them on other mines or on a blacklisted field. Whenever I place a new mine, I let my CSP algorithm check solvability. When I pass the problem, I also pass information about previous runs. Until the new mine is still among inactive variables, I only carry out the past steps. If I get the new mine into node or constrain list (it is possible to just pass it), I start updating the list of previous runs. When I get the new mine into resolved, I check whether my current constraints and nodes match any of the past ones. If they do, I reconnect the list there and return success, if not, I keep on resolving, updating and checking the list. If the resolution succeeds, I discard rest of the old list and return success, else I return failure and the new mine gets placed onto the blacklist. When I encounter situation, where I need the total sum of mines to succeed, I use rand function with 0.08 success rate, if it passes, all the remaining safe fields in those clusters get put on the black list, but the new mine stays, else, the new mine gets removed and its field gets blacklisted.

8 Generator evaluation

The results of my algorithm on the most common parameters of the field were:

results of my minefield generator, 1000 tries

field parameters	average runtime [milliseconds]	maximal runtime
Beginner 9x9 10mines	32.5	614
Intermediate 16x16 40mines	662	1589
Advanced 30x16 99mnines	7723	12990

Results for intermediate and beginner fields are very good. Although, waiting almost 8 second for an advanced game is not usable in real time.
That might not really be a problem though. Today, even mobile processors have multiple computing cores and my algorithm only utilizes one thread. But since every combination is resolved separately, it is not a problem have multiple threads resolving a different combination each, which would save a lot of time. Also, the first generated solution could be a simpler one, for example with the maximal depth of 2 and the more complex grid could be generated after the first one is available.

9 GUI demonstration

I have also developed a functioning GUI, it is not implemented in Java, but in Matlab. It does not feature the grid generating algorithm introduced above, but much simpler one. The whole application servers as a demonstration and a proof of concept.

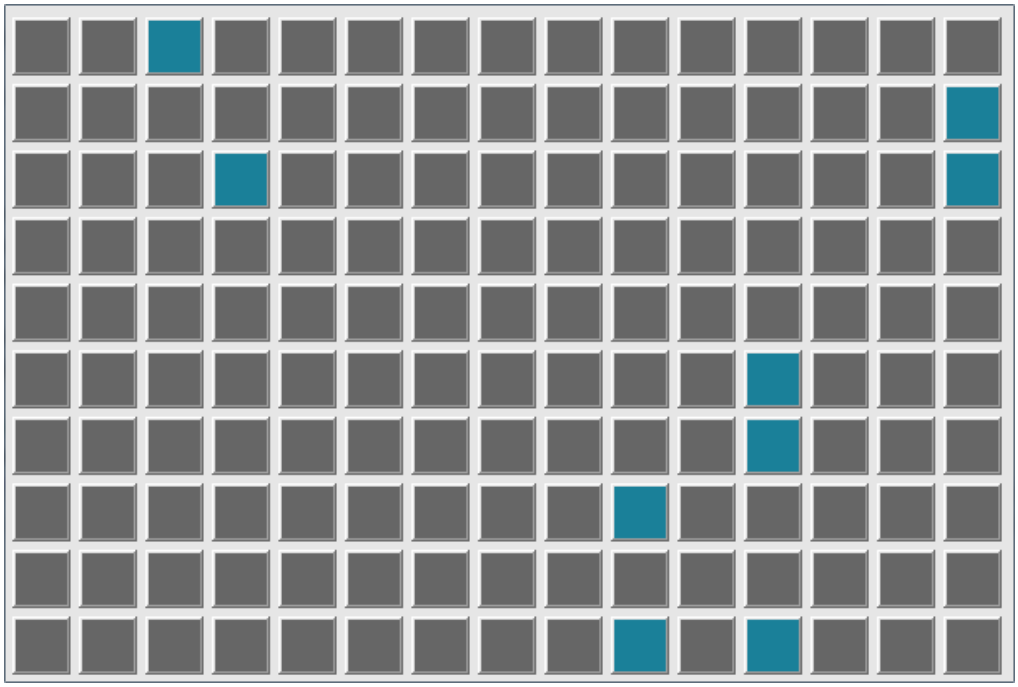

Figure 16:*Screen grab of game ready for initialization*

In order to deal with the slight inconsistency in time needed to generate a new field, separate solution is generated for every field, as they are tested and implemented, the tile turns from gray to blue. Both tiles are clickable, but the blue tiles guarantee solvability.
Thanks to this approach, it is acceptable for a grid to be generated, than if it had to react on user selecting a tile and immediately preparing a field, since solvability depends on a starting point.

10 Inputting text minefield examples into the application

All the minefield representations in this text were in form of basic ASCII graphics, and they can be imported directly into the application, either as they are, or with any number of mines added. Before pasting, make sure that each line ends with a space, not a |symbol and starts with a fields, not a space or index of the row.
The numbers on constraint fields will correct itself, when the mine setup changes, but any difference in spacing will result in a skewed input. All that is needed is pass a string with the minefield, and % symbol, instead of new line, to the stringToField(String s, int x, int y) function, found in any instance of MineMap class and use the Field[][] it returns in a constructor of a new MineMap instance. Any Strategy implementing the Strategy.java provided with PGMS[7] will work with this map.

11 Conclusion

My first goal was to find an efficient solver, that would also be suited for use in a solvable minefield generator.
While the solver did very well in detecting a solvable field, its computation time was underwhelming. Taking two times longer than Equation strategy is acceptable, considering my algorithm isn't aimed mainly at speed, but compatibility and success rate. The CSP by Dr. Studholme on the other hand is clearly superior and the slightly better success rate does not justify my diametrically worse computation time.

The game generator doesn't have a direct comparison. Christian Czepluch's implementation[3] does something closely related, but it a very different way. Instead of finding a solvable minefield, the application only looks for situation, where player is forced to guess and it changes the minefield so that the guess would be correct. Unfortunately, as the tests of solvers revealed, proving that no safe step could be made, is much more tedious than finding one, so the dynamic of the whole game is completely changed. Another solution was provided by Simon Tatham in his portable puzzle collection[5]. While he also uses a solver, it is a very low tier one, that doesn't register a more complicated solutions and discards them. My generator has a chance to be the closest representation of standard random games, with just the unsolvable one discarded. There are certainly differences, but they are well hidden a plus, there is the addition of the maximum depth setting, which does really well to represent the actual complexity the player will have to consider, if he or she wants to win the game.

References

[1] Richard Kaye, *Infinite versions of minesweeper are Turing complete*. School of Mathematics, The University of Birmingham, Birmingham, 2007

[2] Richard Kaye, *Minesweeper is NP-complete*. Mathematical Intelligencer, Birmingham, 2000

[3] Christian Czepluch, *Mines-Perfect 1.4.0*. URL `http://www.czeppi.de/english/index.html`, 2001

[4] MineSweeper, *Authoritative Minesweeper*. URL `http://www.minesweeper.info/index.html`, 2014

[5] Simon Tatham, *Simon Tatham's Portable Puzzle Collection*. URL `http://www.chiark.greenend.org.uk/~sgtatham/puzzles/doc/mines.html#mines`, 2014

[6] Chris Studholme, *Minesweeper as a constraint satisfaction problem*. 2000

[7] John D. Ramsdell. *Programmers Minesweeper* $PGMS$. URL `http://www.ccs.neu.edu/home/ramsdell/pgms/`

[8] Ken Bayer, Josh Snyder and Berthe Y. Choueiry, *An Interactive Constraint-Based Approach to Minesweeper*. Constraint Systems Laboratory, Department of Computer Science and Engineering, University of Nebraska-Lincoln, 2006

[9] Dmitry Kamenetsky and Choon Hui Teo, *Graphical Models for Minesweeper*. Project Report, 2007

[10] T Schiex, H Fargier, G Verfaillie, *Valued constraint satisfaction problems: Hard and easy problems*. IJCAI, Toulouse Cedex France, 1995

[11] R Tarjan *Depth-first search and linear graph algorithms* SIAM journal on computing, 1972

[12] D Frost, R Dechter *In search of the best constraint satisfaction search*. Dept. of Information and Computer Science University of California, Irvine, AAAI, 1994

Appendices

Enclosed CD
There is an enclosed CD-ROM with every print of this thesis.
The contents of the CD-ROM are:
PDF version of this document
Netbeans project No.Guessing.Minesweeper, with the source files of the map generator ($MineMap.java$) and solver ($CSP\_FCH\_MRV.java$), also all the support classes and other code
Matlab files, $mainWindow.m$ will start the GUI demo