Assignment 2
Introduction to Artificial Intelligence

BY

Kartik Rattan, 187001624
Mareesh Kumar Issar, 186000297

Date: October 22, 2019

3. Questions and Writeup

Ques 3.1. Representation: How did you represent the board in your program, and how did you represent the information / knowledge that clue cells reveal?

Solution. Board and all its associated functions are handled by the board class present in board_formation.h file. We represented the board in our program as a 2D vector of integers. We first created the actual board for the game (boardArray) and initialized all its values with 0. To create minesweeper, we then placed **n** mines randomly on a **D X D** matrix using shuffling. We initiate the first N (Number of mines) values in a linear DXD array with mines. Then for each of the first N values with probability p, we either keep the mine there or swap it with one of the elements from N+1 to D X D. After placing the mines, we called the function GenerateValuesAroundMines(), which increased the values of the cells surrounding the mine by 1 every time it encountered a mine on the board.

We then created the board (userBoardArray) that will be displayed to the user as a separate 2D Matrix of integers initialized with value -1.

We represented the information/knowledge in the user board (our KB) as follows:
-1: If a cell is not visited
0-8: Clue value representing the number of mines surrounding that particular cell
10: To represent exploded mine
20: Mine that was flagged successfully by the user

For visual display of the user board we used:
***:** If a cell is not visited
0-8: if the value represented by the cell is 0-8
M: if the value represented by the cell is 10
F: if the value represented by the cell is 20

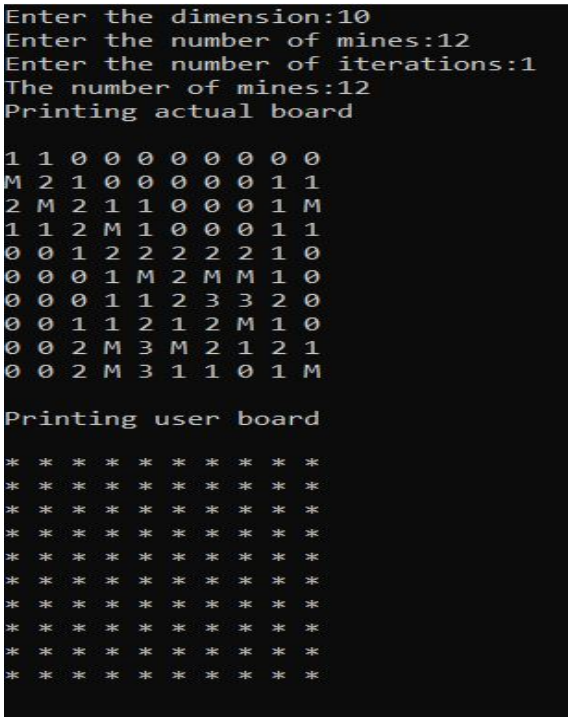


Fig1. Actual board and initial user board for d = 10 and n=12

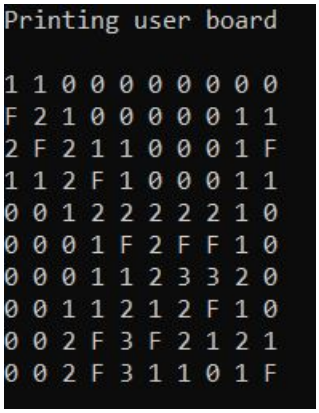


Fig2. User board for d = 10 and n=12 (after solving)

Ques 3.2. **Inference: When you collect a new clue, how do you model / process / compute the information you gain from it? i.e., how do you update your current state of knowledge based on that clue? Does your program deduce everything it can from a given clue before continuing? If so, how can you be sure of this, and if not, how could you consider improving it?**

Solution.

Our algorithm begins by querying a random cell. It then passes through three levels of inference termed as

Level1: Baseline
Level2: Inference_Queue
Level3: CSP

The following is the pseudo code for the first two levels:

Query a random cell
current cell = random cell

Define a queue inference_queue, add the current cell to it.
Define a queue unexplored cells, add the current cell to it.

While (! unexploredcells.empty())
Take the cell and find all its neighbors.
Query each neighbor to find number of hidden cells, number of revealed safe, revealed mines and the total number of mines (Clue).
If (Clue – revealed mines == Hidden Neighbors) ***//Rule 1***
All the hidden cells are mine
Flag the mines, increase avg score.

If (Neighbor Count – Clue – Revealed safe == Hidden Neighbors) ***//Rule 2***
Then all are safe.
Add the cells to a queue: unexplored cells
Add the cells to the inference queue
Query these cells first.

For (each cell in the inference queue) //this ensures we exhaust all the options before opening a random cell again
{
If (Clue – revealed mines == Hidden Neighbors)
All the hidden cells are mine

Flag the mines, increase avg score.

If (Neighbor Count – Clue – Revealed safe == Hidden Neighbors)
Then all are safe.
Add the cells to the inference queue
Query these cells first.
}

At each level, our algorithm ensures that we do not open a random cell until it is unavoidable. Opening a random point might lead to mine explosion and we ensure that it does not happen if our board has a safe cell left with us. In some cases, when a new safe position is opened at the end of the inference queue and is also not evaluated by CSP, a new random point is opened, which might lead to explosion of mine.

After all the deductions are made by baseline and inference_queue, we try to flag any location on the board that wasn't flagged by the above-mentioned methods by using CSP. CSP combines more than one clue to improve our inference from the current state of the board. Using CSP without baseline doesn't really give us much information as the CSP works best when majority of clues are revealed because then it can easily combine nearby clues to infer more about their neighbors.

In order to apply CSP we take a window of 5X5 and make equations related to each cell in the internal 3X3 board that has a clue

For the 3X3 inside the 5X5 window we follow the following steps:

1. Find all the neighbors with clue (0-8) present inside the 3X3
2. Apply Rule 1 and Rule 2 to all of them, to clear/flag any of their surrounding sure hidden cells
3. Find all the neighbors with clue (neighboursWithClue) present inside the 3X3
4. For each neighbor with clue create its equation(neighboursCSP) by calling GetNeighborsCSP() function. The GetNeighborsCSP() function creates a vector consisting of hidden points around a particular clue
5. If(neighboursCSP.size() != 0 && neighboursCSP.size() != (clueValue - numberOfRevealedMines) && (clueValue - numberOfRevealedMines) != 0).
   Then add the equation to a 2D vector of equations (windowCSP). Append the clue point for which the equation was generated at the end of the vector. windowCSP consists of all the valid equations generated by clues present inside the 3X3.

Solving the equations present in windowCSP:

For every possible set considering two equations at a time, we solve them using the Gaussian Elimination method as follows:

1. Define terms for each hidden cell in the 3X3 matrix. Find the equation with larger number of terms (longerEquation) and shorter number of terms(shorterEquation). The shorter equation will be the subset of the longer one and can thus be subtracted from it to deduce information regarding hidden cells. This is achieved as follows:

   For all the terms present in longerEquation try to find and eliminate the common terms that are present in the shorterEquation. This steps essentially finds out if the shorter equation is a subset of the larger.

2. If (total count of terms remaining in longerEquation == difference of their clue values) && (difference of their clue values > 0) && (total count of terms remaining in the shorterEquation == 0) then the shorter equation is a subset of the longer equation and the cells that are still hidden in the longerEquation are definitely a mine and we flag them.

   For example, consider the following arrangement

	0	1	2	3	4
0	**SAFE**	**SAFE**	**SAFE**	**SAFE**	**SAFE**
1	**SAFE**	**1**	**2**	**1**	**SAFE**
2	**SAFE**	**A**	**B**	**C**	**SAFE**
3	**SAFE**	**1**	**2**	**1**	**SAFE**
4	**SAFE**	**SAFE**	**SAFE**	**SAFE**	**SAFE**

Here A, B, and C represent the hidden cells.

Using rule 1 and rule 2 and their combination with inference_queue, we cannot deduce anything about A, B, and C.

But if we run CSP on this 5x5 (around central cell B) for and generating equations for all the clues present in central 3X3 (denoted in red) we get, the following equations:

A+B=1 (for clue value 1 at location (1,1))
A+B+C=2 (for clue value 2 at location (1,2))
B+C=1 (for clue value 1 at location (1,3))
A+B=1 (for clue value 1 at location (3,1))
A+B+C=2 (for clue value 2 at location (3,2))
B+C=1 (for clue value 1 at location (3,3))

Solving the above set of equations two at a time i.e.
A+B=1 (shorter equation as it has 2 terms A and B)
A+B+C=2 (longer equation as it has 3 terms A, B and C)

we deduce that C is definitely a mine and we flag it, similarly from equations

B+C=1 (shorter equation as it has 2 terms B and C)
A+B+C=2 (longer equation as it has 3 terms A, B and C)

we deduce that A is definitely a mine and we flag it, so that the agent does not open these cells randomly during the next iteration.

Our program first tries to infer whatever it can by applying baseline. Then our program applies CSP on a 5X5 window across the board and flags all the potential mines that were not flagged by baseline thereby reducing the chance that when a cell is opened at random the next time, that cell is a mine.

No, our program cannot deduce everything it can before continuing because after the mines were flagged by CSP we did not run inference again on the board, to deduce other possible information that those flagged mines revealed (or cells that are opened).

We could improve it further by running inference on the cells around the positions flagged as mines or opened as clues by CSP. For example, consider the following case:

1	*****	**2**
*****	*****	*****
*****	**3**	*****

Here the baseline, inference_queue and even the CSP fail to deduce anything about the hidden cells (*). But if we try filling the hidden cells with all the possible valid combinations of mines (**using variable assignment tree**) then we get the following results:

1	**M**	**2**
2	**4**	**M**
M	**3**	**M**

1	**2**	**2**
2	**M**	**M**
M	**3**	**2**

1	**2**	**2**
2	**M**	**M**
1	**3**	**M**

In all the 3 cases we find that second cell in the first column is always safe, and thus we can open it. Using this approach has another added advantage of giving a probability to each cell of being safe/mine, as we have calculated all the possible valid assignments.

Ques 3.3. **Decisions: Given a current state of the board, and a state of knowledge about the board, how does your program decide which cell to search next? Are there any risks, and how do you face them?**

Solution. After running the baseline algorithm along with inference_queue we try to reduce our risk to a large extent of opening a random cell that might be a mine. We then apply CSP, which further flags more mines and remove the potential mines that could have been opened randomly in the subsequent steps.

A risk in CSP step is that we are solving pairs of equations and trying to deduce if hidden cell is a mine, but we are not propagating this to solve the remaining equations that are present in the current window. We can mitigate this by breaking from the CSP and rebuilding all the equations again. This would be an expensive procedure as we have to compute and solve all the equations again.

After these steps the agent opens a random hidden cell. This step has the risk of opening up a mine randomly. To mitigate this, we could assign probability (that a cell is safe) using the variable assignment tree

discussed in the previous question. We also discuss an improvement in the last part when the total number of mines are known.

NOTE: Here we cannot just compute the probability by just using the formula (clue value / number of hidden cells around it) as the neighbors of the given clue are also neighbors of some other nearby clue. There are also missing equations generated from cells present at the corners because we look for 5X5 windows enclosing a 3X3 matrix that builds the equations. Equation formation is also limited by the size of the window.

Ques 3.4. **Performance: For a reasonably-sized board and a reasonable number of mines, include a play-by-play progression to completion or loss. Are there any points where your program makes a decision that you don't agree with? Are there any points where your program made a decision that surprised you? Why was your program able to make that decision?**

Solution. For board of dimensions 15X15 with 35 mines, the play by play progression is as follows:

STEP 1: We print out the actual board (for user reference) along with the initial user board.

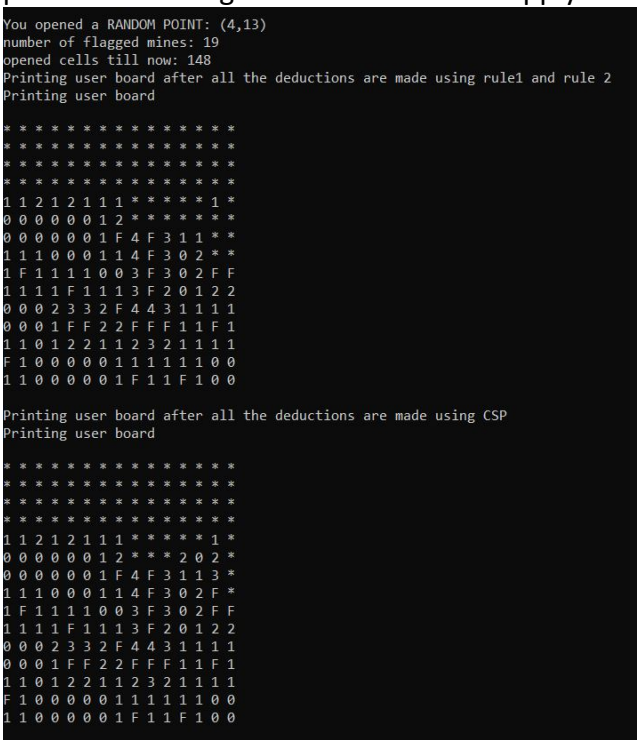
STEP 2: We open the random point (8,11) which reveals the clue value of 0. We then apply baseline, followed with inference_queue and display the resulting board. After that we apply CSP and print the resulting board.

```
For iteration 1 total safe places initially- 190
You opened a RANDOM POINT: (8,11)
number of flagged mines: 4
opened cells till now: 20
Printing user board after all the deductions are made using rule1 and rule 2
Printing user board

* * * * * * * * * * * * * *
* * * * * * * * * * * * * *
* * * * * * * * * * * * * *
* * * * * * * * * * * * * *
* * * * * * * * * * * * * *
* * * * * * * * * * * * * *
* * * * * * * * * F 3 1 1 * *
* * * * * * * * * F 3 0 2 * *
* * * * * * * * * F 3 0 2 * *
* * * * * * * * * F 2 0 1 * *
* * * * * * * * * 4 3 1 1 * *
* * * * * * * * * * * * * * *
* * * * * * * * * * * * * * *
* * * * * * * * * * * * * * *
* * * * * * * * * * * * * * *

Printing user board after all the deductions are made using CSP
Printing user board

* * * * * * * * * * * * * *
* * * * * * * * * * * * * *
* * * * * * * * * * * * * *
* * * * * * * * * * * * * *
* * * * * * * * * * * * * *
* * * * * * * * * * * * * *
* * * * * * * * * F 3 1 1 * *
* * * * * * * * * F 3 0 2 * *
* * * * * * * * * F 3 0 2 * *
* * * * * * * * * F 2 0 1 * *
* * * * * * * * * 4 3 1 1 * *
* * * * * * * * * * * * * * *
* * * * * * * * * * * * * * *
* * * * * * * * * * * * * * *
* * * * * * * * * * * * * * *
```

STEP 3: We open the random point (10,0) which reveals the clue value of 0, we then apply baseline and inference_queue and display print the resulting board. After that we apply CSP and print the resulting board.

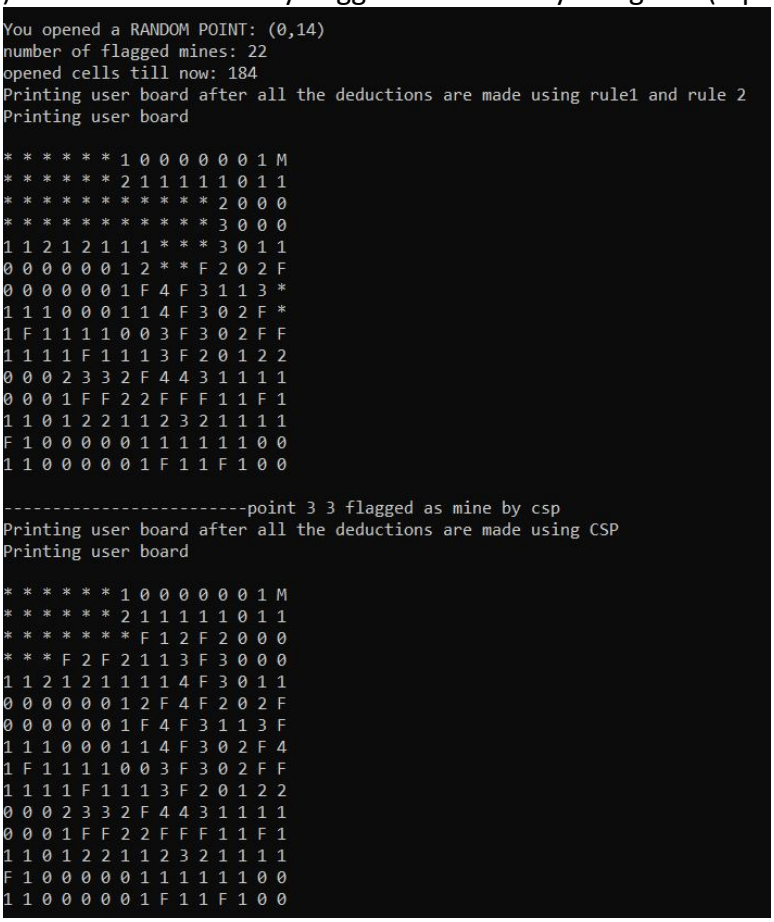
```
You opened a RANDOM POINT: (10,0)
number of flagged mines: 7
opened cells till now: 60
Printing user board after all the deductions are made using rule1 and rule 2
Printing user board

* * * * * * * * * * * * * * *
* * * * * * * * * * * * * * *
* * * * * * * * * * * * * * *
* * * * * * * * * * * * * * *
* * * * * * * * * * * * * * *
* * * * * * * * * * * * * * *
* * * * * * * * * F 3 1 1 * *
* * * * * * * * * F 3 0 2 * *
* * * * * * * * * F 3 0 2 * *
1 1 1 1 * * * * * F 2 0 1 * *
0 0 0 2 * * * * * 4 3 1 1 * *
0 0 0 1 F F 2 2 * * * * * * *
1 1 0 1 2 2 1 1 * * * * * * *
F 1 0 0 0 0 0 1 * * * * * * *
1 1 0 0 0 0 0 1 * * * * * * *

Printing user board after all the deductions are made using CSP
Printing user board

* * * * * * * * * * * * * * *
* * * * * * * * * * * * * * *
* * * * * * * * * * * * * * *
* * * * * * * * * * * * * * *
* * * * * * * * * * * * * * *
* * * * * * * * * * * * * * *
* * * * * * * * * F 3 1 1 * *
* * * * * * * * * F 3 0 2 * *
* * 1 1 1 1 0 0 3 F 3 0 2 * *
1 1 1 1 F 1 1 1 3 F 2 0 1 * *
0 0 0 2 3 3 2 F 4 4 3 1 1 * *
0 0 0 1 F F 2 2 F F F 1 1 F 1
1 1 0 1 2 2 1 1 2 3 2 1 1 1 1
F 1 0 0 0 0 0 1 1 1 1 1 1 0 0
1 1 0 0 0 0 0 1 F 1 1 F 1 0 0
```

STEP 4: We open the random point (4,13) which reveals the clue value of 1, we then apply baseline + inference_queue and display print the resulting board. After that we apply CSP and print the resulting board.

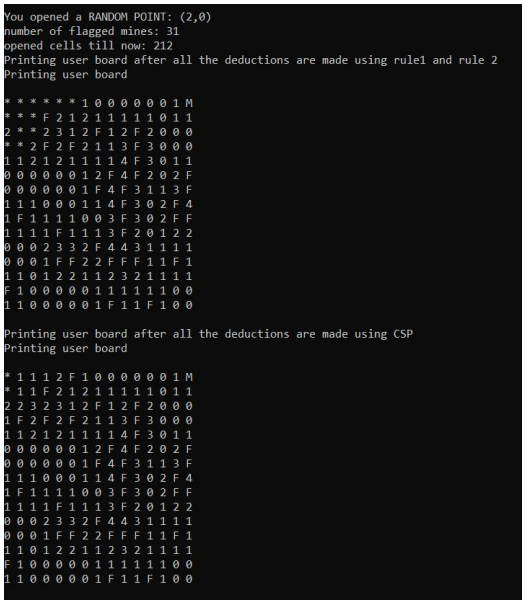


STEP 5: We open the random point (0,14) which reveals the value of M (i.e. we accidently clicked on a mine), we then apply baseline + inference_queue and display print the resulting board. After that we apply CSP and print the resulting board.

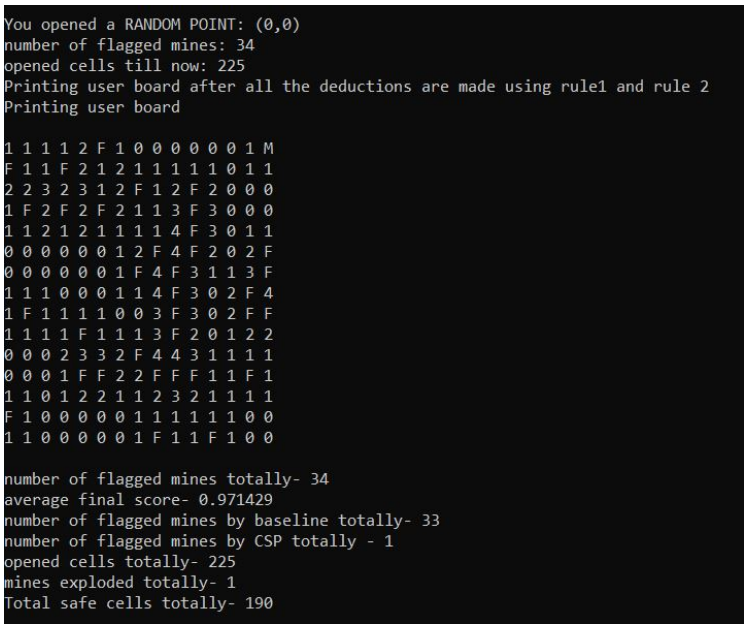
NOTE: Here the cell (3,3) has been successfully flagged as a mine by using CSP (equation solving).



STEP 6: We open the random point (2,0) which reveals the value of 2, we then apply baseline + inference_queue and display print the resulting board. After that we applied CSP and displayed the resulting board.



STEP 7: We open the random point (0,0) which reveals the value of 1, we then apply baseline + inference_queue and display print the resulting board. After that we did not apply CSP as all the cells were already opened.



We see the following unusual step taken by our agent. When the mine density is large, CSP tends to flag some safe cells as mines. Usually these safe cells are surrounded by correct mines. For example: for the original board shown below:

```
Printing actual board

M M M 3 2 M M M 3 M
M 7 M M 4 3 6 M 5 2
M M 4 M 4 M 4 M 6 M
M 5 3 4 M 5 5 M M M
M 5 M 3 M M M 6 M 4
M M 4 4 5 M M M 4 M
M M M 2 M M 6 3 4 M
M 5 3 3 4 M 3 M 3 2
M 3 3 M 3 2 3 2 3 M
1 2 M M 2 1 M 1 2 M
```

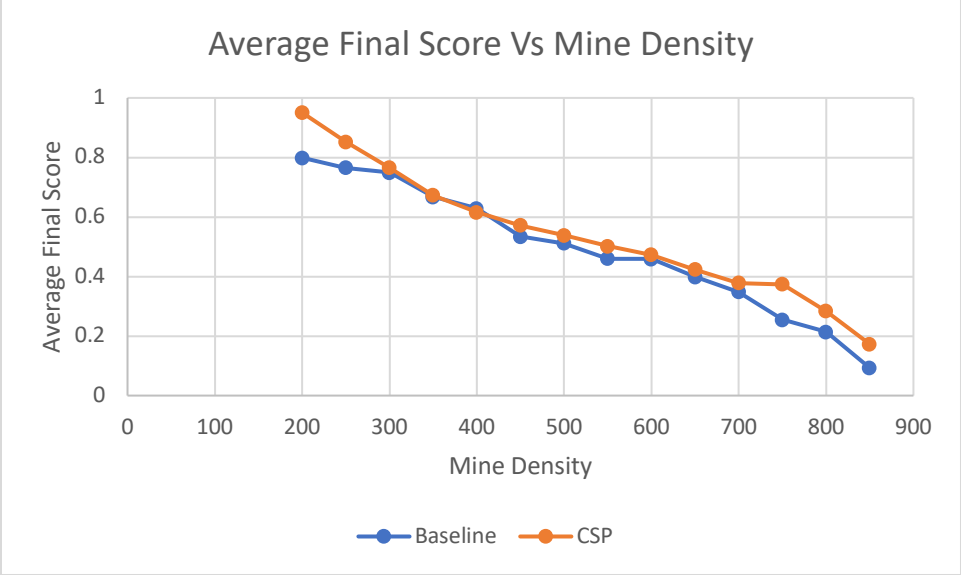
After a certain number of steps into solving, the CSP accidently flags a safe cell as mine. This happens because we do not dynamically change the latter equations if two former equations produce some deterministic results. Thus, if two equations infer that a hidden position is a mine or is safe, this is not propagated to latter equations in real time. Hence, it may make a mistake and flag a safe position as mine because it has no knowledge that a mine has already been flagged. This can be avoided by either dynamically changing the equations, or as discussed earlier: breaking the loop and generating the equations again.

```
for central cell 2 , 5
Printing user board

M F M 3 2 M * * * *
F 7 F F 4 3 6 M * 2
F F 4 M * * * M * *
M 5 3 4 M * 5 M F F
F * * 3 * * F 6 M 4
* M * * 5 M F M 4 M
M M * * * * * 3 4 M
* * 3 * * M 3 * * 2
* * 3 M 3 * 3 * * *
* 2 * * * * * * 2 *

-------------------------point 2 4 flagged as mine by csp
```

Here central cell represents the center of the 5X5 window.

Ques 3.5. **Performance: For a fixed, reasonable size of board, plot as a function of mine density the average final score (safely identified mines / total mines) for the simple baseline algorithm and your algorithm for comparison. This will require solving multiple random boards at a given density of mines to get good average score results. Does the graph make sense / agree with your intuition? When does minesweeper become `hard'? When does your algorithm beat the simple algorithm, and when is the simple algorithm better? Why?**

Solution. For the board dimensions **30X30** having different mine densities, we compare applying baseline and inference_queue (represented in the graph as baseline) with applying baseline, inference_queue, and Constraint Satisfaction Problem (represented in the graph as CSP) to get the following results. We run each of the iteration 20 times:



Yes, the graph makes sense and agrees with our intuition as CSP gives us a better performance as compared to the baseline implementation. Some of the cells in the inference queue remain unopened even when the cells surrounding it can infer its outcome. This happens because the cell is present higher up in the queue, is probed and we move on to the next clue. Further down the queue, some latter cell can make predictions about a former cell. So, these cells even though they can be correctly queried remained unopened. These cells are caught by the CSP and thus query them. Hence, we have more clues which results to flagged mines by CSP and thus our performance betters.

Both the baseline and CSP algorithm have comparable performance when the number of mines is less than (D*D)/2. After this we see that CSP provides better results than the baseline. This is due to the fact that CSP is able to flag several mines that could have been opened at random by the agent and thus we see an improvement in our average final score. Thus, after (D*D)/2 the minesweeper becomes hard. (Avg Final score drops below 0.5)

When the dimensions of the board are smaller, CSP does not help drastically. Thus, we can avoid the space and time computations of running CSP and hence our baseline performs better than CSP in terms of computation. Also, in the beginning with few opened cells, CSP does not infer much and most of the deductions are done by baseline and inference queue.

Ques. Efficiency: What are some of the space or time constraints you run into in implementing this program? Are these problem specific constraints, or implementation specific constraints? In the case of implementation constraints, what could you improve on?

Solution. We have the following space and time constraints, where D is the dimension:

	SPACE	TIME
Baseline	O(1)	O(1)
Inferenece_queue	$(O(D^2) - \text{total mines}) \sim O(D^2)$	$O(D^2)$
CSP	$O(D^2)$	$O(D^3)$

Our algorithm is mainly based on Level 1 and 2 of inference. The inference queue underneath work on the two baseline rules. Thus, mostly, we have a constant time implementation of the rules. We run our algorithm until we open all the hidden cells present in the maze. Thus, the number of times our algorithm runs is DXD

as there are D^2 cells in the maze. The CSP part work few times only when the above two levels fail. Thus, the $O(D^3)$ is amortized growth rate. We thus conclude that the runtime of our algorithm is $O(D^2)$. This is mostly implementation based, depending upon our choice of data structures. Also, since the aim is to uncover all the cells, the runtime is $O(D^2)$. The following questions helps us in improving our algorithm computationally with early termination, only because the number of total mines is known.

Ques. Improvements: Consider augmenting your program's knowledge in the following way - tell the agent in advance how many mines there are in the environment. How can this information be modeled and included in your program, and used to inform action? How can you use this information to effectively improve the performance of your program, particularly in terms of the number of mines it can effectively solve? Re-generate the plot of mine density vs expected final score for your algorithm, when utilizing this extra information.

Solution. As the agent now knows about the number of mines remaining in the board, we gain advantage both in terms of increasing our efficiency in finding mines (hence increasing our average final score) as well as computationally. Let us discuss both of them individually:

1. Let's begin with the easier of the two explanation- Computational Advantage. Since we now know the number of mines present in the board, this will lead to early termination. Hence, if we have detected all the mines present initially (either by exploding or flagging), there is no need to go through the remaining cells. Also, probing less about of cells also leads to less space requirements, as some cells never enter the data structures for analysis.
2. During our last two levels, they search and exhaust all the possibilities locally. They look for safe cells using the two baseline rules and CSP, but do not open a cell until its sure that it's a safe mine. Adding to this we can make another rule where which can be run locally for a (N' X N' window, where N' < N, the dimension of the board) or Globally:

   if (total number mines – revealed mines till now == 0)
   ALL REMAINING UNOPENED CELLS ARE SAFE CELLS
   END THE ALGORITHM

If they do not have any other safe cells to open, they query for a random mine. This is where the knowledge for total number of mines is helpful. As explained by Dr. Cowan using the example

2	?	?
?	?	?
?	?	?

a) Number of mines = 3
b) Number of mines = 7

Briefly explaining here, Dr. Cowan acknowledges how the information about number of mines allows us to select far away cells in case a) and closer cells in case b).

Now when we query for a new random unopened cell, and if it has an opened neighbor, the probability is defined by the clue and not the total number of mines. For instance, in the below example we see that the probability for the highlighted cells change from 1/5 to 1/3 with the presence of the clue. The total number of mines did not affect it.

2	?	?

?	?	?
?	?	1

Our new implementation is based upon the following property:

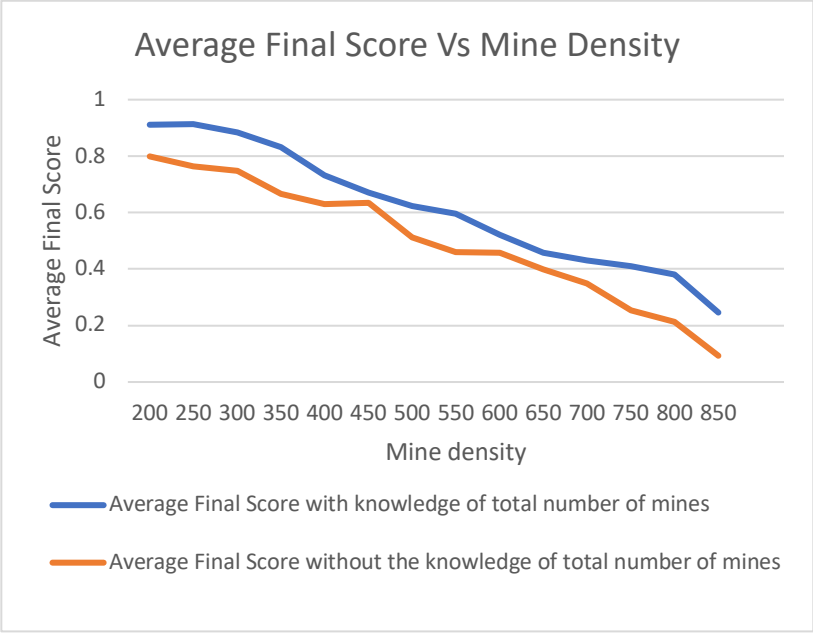
K = [number of mines remaining] / [Safe Places]

We make one assumption here. The ratio K remains largely unchanged since if our agent is not terrible, it identifies a clue which lead to flagging a mine. Hence, the number of safe places decreases, but it also leads to flagging and thus number of mines remaining also decreases.

1. Number of mines are large: When we query for the new cell, if K is large, we query cells close to clues. We begin by querying cells with clue 1, then 2, 3 and so on. Since the total number of mines are high, any new random point might turn out to be a mine. For instance, let us build a 100X100 with 5000 mines. The value of K is 5000/10000 = 0.5. if we open a new cell with clue value 1, the probability that a neighboring cell has a mine is 1/8 = 0.125. Thus, we would prefer to open a cell close to it rather than any random clue.

2. Number of mines are small: When we query a new cell, if K is small, we query cells far away from the clues, i.e. a random point. For instance, let us build a 100X100 with 50 mines. The value of K is 50/10000 = 0.005. If we open a new cell with clue value 1, the probability that a neighboring cell has a mine is 1/8 = 0.125. Thus, we would prefer to open a random cell far away as we expect that we most often than not query a safe position.

For our implementation, if K > 0.2, we do the first step. If K is < 0.2, we do the second step. Though we saw some improvement in the results for a 30 X 30, the improvement is not that significant. This is also because probability gives an estimate, a belief and not the exact value. Thus, it directs us to search locally or globally, but its direction may or may not be true.



Dealing with Uncertainty

Ques. When a cell is selected to be uncovered, if the cell is 'clear' you only reveal a clue about the surrounding cells with some probability. In this case, the information you receive is accurate, but it is uncertain when you will receive the information.

Sol. We realized there are two ways to look at uncertainty in this case, one where we play the minesweeper until we blow a mine (REAL GAME) and the tweaked version we implemented where we continue to uncover all the cells until there is not cell left unopened (TWEAKED GAME).

One thing that is common for both the games is that we are losing our opportunity to win the game if with some probability **p** we do not open the clear cell. This takes us further away from winning the game. Also, in this case, if we identify a safe cell and fail to open it then we look for either another safe cell in our safe cells queue or open a random mine. So, from 100% sure of opening a safe mine at an instance, we might end up opening a mine in the next step.

In the REAL GAME, this will definitely hurt our chances to win. The probability of winning the game will decrease. Let's imagine our base probability to reveal a clue is 0.5. Therefore, 50% of the times we identify and open a clear cell, we achieve nothing at all. Moreover, 50% of the times we have to open another cell where we might end up blowing up a mine and lose. Thus, this uncertainty will impact us in a negative way.

In the TWEAKED GAME, our termination occurs when we uncover all the cells available. The objective function that relates to us winning the game is the average final score. Since we end up giving away clues with probability **p**, we end up opening another cell with probability **(1-p)**. The clue could have improved our knowledge base and helped us in identifying the mines. Thus, this impacts our final score and the efficiency of our AI decreases.

Since our focus is on the second version of game, we came up with two adapted implementations related to the TWEAKED GAME.

1. Now the major loss occurs when we ignore the clue/safe mine and have to query a random cell which ends up being a mine. We can remove this possibility by continuously querying the current cell until it is opened. This increases the computation time but makes sure that we update our knowledge board and gather useful information from the clue we identified. Hence, this did not impact our final score for an instance of the game but increased our runtime for playing the game.

2. If we have a constraint of querying a cell just once, the it surely decreases our average final score. We end up not updating our knowledge board and the identified clue does not impact the game at that point. The amount of decrease also depends on the probability **p.** If **p is high,** then we ignore the clues with high probability, the KB is not updated regularly and to add to it we might end up blowing the mine in the next step. On the other hand**, if p is small,** we query the clue identified more often and update our KB. It is better than not using the clue at all, but still lead to decrease in the average final score.

Ques. When a cell is selected to be uncovered, the revealed clue is less than or equal to the true number of surrounding mines (chosen uniformly at random). In this case, the clue has some probability of underestimating the number of surrounding mines. Clues are always optimistic.

Sol. Mathematically, we represent the baseline solution as:

For 3X3 Window,
1. If (Clue – revealed mines == Hidden Neighbors)

All the hidden cells are mine
Flag the mines, increase avg score.

2. If (Neighbor Count – Clue – Revealed safe == Hidden Neighbors)
   Then all are safe.
   Query these cells first.

If the clue is optimistic (or the clue reveals a number $\leq$ to the actual number mines around the cell), it impacts our equations above as well as inference done with Constraint Satisfaction Problems algorithm.

We can see a direct impact of the optimistic clues because of their impact on the equations above. Thus, we will ow focus on the CSP implementation with optimistic clues using the following example of 3X3 minesweeper.

1	2	1
A	B	C
1	2	1

The two equations implemented with the above table using optimistic clues are:

$A + B + C \leq 2$, _____1
$A + B \leq 1$, and _____2
$B + C \leq 1$ _____3

We use Gaussian Elimination to Solve the Systems of Equations for CSP. Now, if we subtract 3 and 2 from 1 individually, we are left with:

$A \leq 1$,
$C \leq 1$, and
$B \leq 0$

Since the number of mines cannot be negative, we deduce that B is not a mine and is a safe cell which can be probed further. For the first two equations, we can deduce the following:

1) If there is no external Oracle.
It becomes extremely difficult to effectively deduce from the 6 equations about whether A and C are indeed mines. This resonates from the fact that equation 1 does not tell us confidently that there are 2 mines within A, B and C. There might be 2, 1 or 0 mines. Now this tells us both of them are mines, one of them is or neither of them is a mine. Also, if one of them is a mine, then we cannot definitely tell whether it is A or C. This will require to solve a lot of equations of the type variable ≤ 0 and infer something from it.

2) If there is an Oracle.
Now if an external Oracle telling us that we have N number of mines, we can infer from the remaining free cells, number of total revealed mines and total remaining hidden neighbors. For instance:
If we knew that the board has 10 mines. We also know that before stumbling upon this 3X3 window, we have revealed 8 mines. Thus, the remining number of mines are 2. Also, we have 2 hidden neighbors left (we keep check of remaining hidden neighbors in the board). Thus, this will help us deduce that the two remining cells A and C are definitely mines.

Thus, either ways to successfully solve the board, we have to evaluate more equations consistently to reach equations of the type: variable ≤ 0 to successfully deduce information and update our knowledge board. This will lead to an increase in our number of computations made, making the maze harder to solve as well decreasing the average final score. We might reach a point where the equations are not solvable with the current KB. This leads us to opening a random cell which increases are chances of hitting a mine.

Ques. When a cell is selected to be uncovered, the revealed clue is greater than or equal to the true number of surrounding mines (chosen uniformly at random). In this case, the clue has some probability of overestimating the number of surrounding mines. Clues are always cautious.

Sol. Taking the same example as the above and also mentioning explicitly that a cell can contain just mines, we have the following scenario:

$A + B + C \geq 2$, _____1
$A + B \geq 1$, and _____2
$B + C \geq 1$ _____3

We use Gaussian Elimination to Solve the Systems of Equations for CSP. Now, if we subtract 3 and 2 from 1 individually, we are left with:

$A \geq 1$,
$C \geq 1$, and
$B \geq 0$

Since one cell can have just one mine, we can with certainty say that both A and C are mines. But what do we deduce about B?

1) If there is no external Oracle.

   It becomes extremely difficult to effectively deduce from the 6 equations about whether B is a mine or not. This resonates from the fact that equation 1 does not tell us confidently that there are 2 mines within A, B and C. There might be 2 or 3 mines between them. Now this tells us that B might or might not be a mine.

2) If there is an Oracle.

Now if an external Oracle telling us that we have N number of mines, we can infer from the remaining free cells, number of total revealed mines and total remaining hidden neighbors. For instance:
If we knew that the board has 10 mines. We also know that before stumbling upon this 3X3 window, we have revealed 8 mines. Thus, the remining number of mines are 2. Thus, since both A and C are mines, this leads us to believe that the remaining mines in the board equals 0 and B is not a mine anymore.

In both the cases, we remain cautious so that we don't flag a mine as safe. We always lack confidence in explicitly saying a cell is free, stating it might or might not be a mine.