Rutgers University.

CS 440: Artificial Intelligence **Rutgers: Spring 2021**

Artificial Inteligence

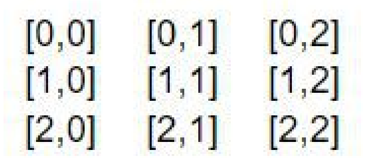
Homework: Minesweeper

By: Wilson Peralta, Krit Bhattarai *NetId: wrp29, kkb62*

Abstract

This tesis explores the how the representation and manipulation of knowledge regarding the mine field of a minesweeper game.

Representation
We represented the board as a matrix of size dim*dim. For instance, for dim = 3, the locations in the board would appear as:



[0,0]	[0,1]	[0,2]
[1,0]	[1,1]	[1,2]
[2,0]	[2,1]	[2,2]

Figure 1: 3x3 matrix example.

Now, the environment would be represented by several other matrices which hold the relevant information. The matrices are all dim*dim matrices and store the following information:

-Whether or not the location is a mine.
-If it is not a mine, the total number of mines in its surroundings/neighborhood (clue).
-Total number of neighbors for the location (we had to keep track of this as well because all the locations do not have exactly 8 neighbors, it can be either 3, 5 or 8).

For our knowledge base, we have other dim*dim matrices, that store the following information for a location:
- The clue , how many mines are in the surrounding neighbors.
- Total number of neighbors.
- Total number of identified neighbors.
- Total number of unexplored neighbors.
- Total number of safe neighbors.
- If a location has been identified or not. If it has been identified, the element of this matrix at said location will be set to IDENTIFIED.(unidentified set by default.)
- If it has been identified, Whether or not our location is safe, if it is safe the element of this matrix at said location will be set to SAFE. (not safe set by default.)

Once we discover this information, it will be added to our algorithm's knowledge base. The inference would be represented by algebraic equations as the matrices in our knowledge base just store numbers representing the information. If a constraint is satisfied, the equation would be true, false otherwise.

We start with an initial knowledge base, we feed it with current information: number of neighbors, number of unidentified neighbors, number of neighbors marked safe and number of mines identified in the neighborhood. we can directly access the clue from the environment by getting the number of surrounding mines at a location, but we do not have direct access to all this information. So, we will traverse to each neighbor, check their status at that point and increase or decrease values accordingly.
After updating information for a location, we will backtrack and update information on each possible location which can be updated by new relevant information.
Then, we will check constraints for all updated locations in the following manner:

- If the number of identified neighbors minus the clue equals the number of unexplored neighbors, all unexplored neighbors are classified as mines and the knowledge base is updated accordingly.
- If the total number of safe neighbors, minus the number of revealed safe neighbors equals the number of hidden neighbors, all hidden neighbors are classified safe and the knowledge base is updated.

After doing this constraint checking process, if we were able to obtain new information, we will backtrack and update information on each possible location which can be updated by newly obtained information. We will then repeat this process by backtracking to the locations where information has been updated, until and unless no new information can be inferred.

Our program will deduce everything these constraints are able to deduce for the entire board. This can be guaranteed as our program backtraces to update information for the entire environment/board each time something new is discovered and also does constraint checking and inferring with every updated data.
However, the information revealed by this process might not be all the inferences we could have necessarily done with the available knowledge. As an improvement, we implemented the following:

-We will explore each of the corners first due to the lesser probability of finding neighboring mines. Also, facilitating the constraints checked for when the corner location equals zero or clue equaling 3(number of neighbors).

-Instead of randomly checking for new locations where contraints were not able to identify all neighbors, we will check neighbors of a given location if the probabiliy of a mine being on one of those locations is less than 50%, 50% being the probabilty of exploring a random location. Therefore, if it is less than 50% then explore neighbors.
That is by:
N probabilty = (clue/total number of hidden neighbors.)
The information is updated every time we query a cell and calculate its neighbors. Then, we proceed to choose one of the neighbors marked as safe and proceed to compute its neighbors, using the same method described above.
For a given state of the board, the program tries to make inferences, updates information for all possible cells if it was successful, and repeats the same process with the updated state until there is nothing more to infer. So, the decision of which cell to search next is based upon where the constraint was satisfied, and where the information was updated, that is, if a constraint is satisfied at a location, its neighbors are explored and updated recursively to the point where no further data is available. Then, the process is repeated for all updated locations.
Now, if no more inferences can be successfully made, a new location is explored based on wether we have any neighbors that meet 50% probability criteria. Otherwise locations will be explored randomly until criteria for any location and its neighbors is meet.

The program selects the next location based on whether it was marked safe by one of the constraints mentioned above, if no location was identified. Then, by selecting which cell has a lower probability of being a Mine, by our formula described above.

Rutgers University.

Based on a board of size $30 * 30$ on the basic agent, we found that:

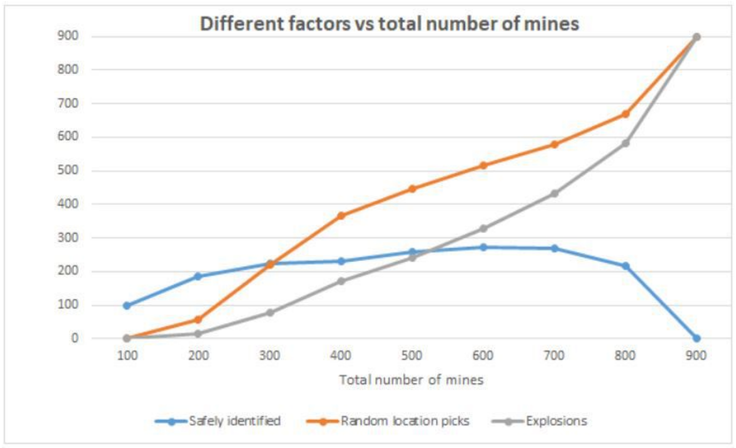
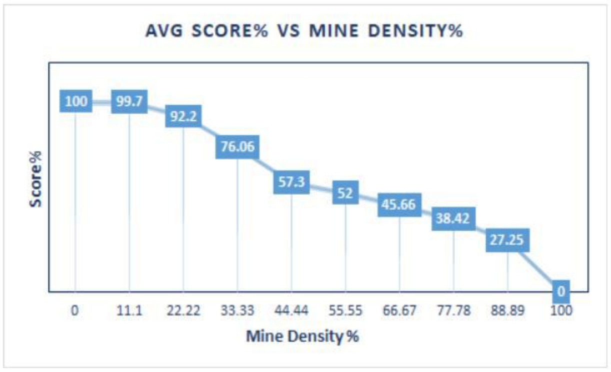


Figure 2: function of mine density the average final score (safely identified mines / total mines)

For the basic agent the number of random location picks increases, increasing the number of explosions due to the 50% probability of a random location to be a mine. Hence, the more random picks the higher the chance of exploding.
Also, for the mine density we calculated the average of runs without explotions in a game:



As one could deduce, the higher the mine density the less likehood of finishing a game without exploding.

Rutgers University.

For our improved agent making more “educated guesses” based on our constraints. Random location picks dropped significantly by 10.03% which in turn improved the safe identification of bombs by up to 11.75% in our best escenarios.

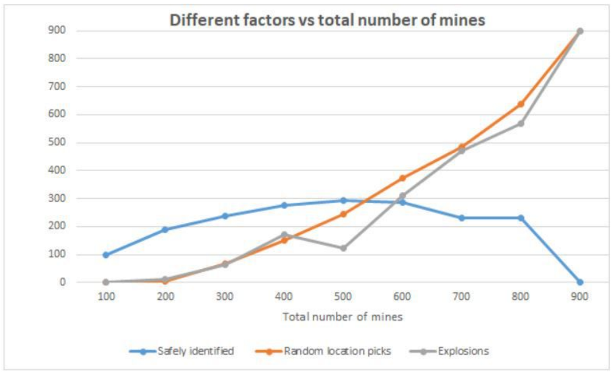
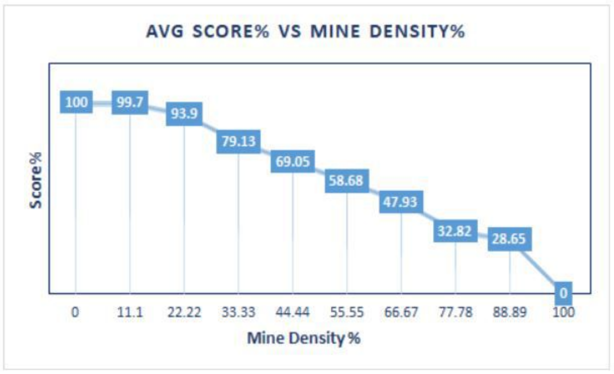


Figure 3: function of mine density the average final score (safely identified mines / total mines for improved agent)



We integrated in our program a step-by-step printing system that will show all moves taken by the program, we ran it many times and for our improved agent, we were fascinated by the effectiveness of it. We did not notice any significant decisions that we could’ve handled differently for an improvement worth mentioning in our results.
For the efficiency we found that: Querying a cell O(1), updating info on current cell O(1) checking its neighbors O(n), calculating information obtained from neighbors O(n), updating info on current cell and neighbors O(1). most of the calculations is done recursively, Therefore; $O(n^C)$ for some constant c.