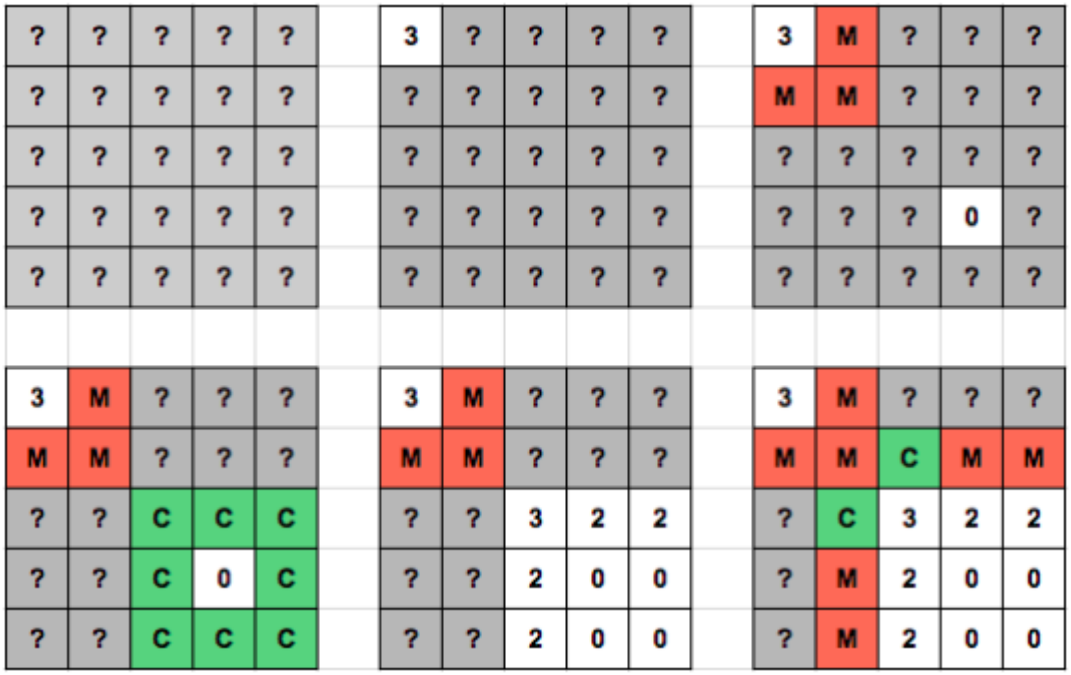
# MineSweeper Report

## CS 440




*Cesar Herrera, Jae Weon Kim, Sunehar Sandhu*

*Spring 2020*

# List of Figures

# Introduction

The goal of this project is to write a program to play Minesweeper- that is, a program capable of sequentially deciding what cells to check, and using the resulting information to direct future action.

# Program Specification

There should effectively be two parts to the program, the **environment** representing the board and where the mines are located, and the **agent**. When the agent queries a location in the environment, the environment reports whether or not there was a mine there, and if not, how many of the surrounding cells are mines. The agent reports whether or not there was a mine there, and if not, how many of the surrounding cells are mines. The agent should maintain a knowledge base containing the information gained querying the environment, and should not only be able to update its knowledge base based on new information, bus also be able to perform inferences on that information and generate new information.

- The environment should take a dimension d and a number of mines n and generate a random $d$ x $d$ boards containing n mines. The agent will not have direct access to this location information, but will know the size of the board. *Note: It may be useful to have a version of the agent that allows for manual input, that can accecpt clues and feed you directions as you play an actual game of minesweeper in a seperate window.*

- In every round, the agent should assess its knowledge base, and decide what cell in the environment to query.

- In responding to a query, the environment should specify whether or not there was a mine there, and if not, how many surrounding cells have mines.

- The agent should take this clue, add it to its knowledge base, and perform any relevant inference or deductions to learn more about the environment. If the agen is able to determine that a cell has a mine, it should flag or mark it, and never query that cell. If the agent can determine a cell is safe, it's reasonable to query that cell in the next round.

- Traditionally, the game ends whenever the agent queries a cell with a mine in it - a final score being assessed in terms of number of mines safely identified.
- However, extend your agent in the following way: if it queries a mine cell, the mine goes off, but the agent can continue, using the fact that a mine was discovered there to update its knowledge base (but not receiving a clue about surrounding cells). In this way the game can continue until the entire board is revealed - a final score being assessed in terms of mines safely identified out of the total number of mines.

The last modification allows the game to 'keep going' and avoids the situation where you accidentally find a mine early in the game and terminate before the game gets interesting.

You may either do a GUI or text based ineterface - the important thing for the purpose of the project is the representation and manipulation of knowledge about the mine field. This will draw on the material discussed in a) Search, b) Constraint-Satisfaction Problems, and c) Logic and Satisfiablility. You must implement a basic MineSweeper agent, described in the next section, and your own agent as an improvement on the basic one.

# A Basic Agent Algorithm for Comparison

Implements the following simple agent as a baseline strategy to compare against your own:

- For each cell, keep track of
  - whether or not it is a mine or safe (or currently covered)
  - if safe, the number of mines surrounding it indicated by the clue
  - the number of safe squares identified around it
  - the number of mines identified around it
  - the number of hidden squares around it.
- If, for a given cell, the total number of mines (the clue) minus the number of revealed mines is the number of hidden neighbors, every hidden neighbor is a mine

- If, for a given cell, the total number of safe neighbors (8-clue) minus the number of revealed safe neighbors is the number of hidden neighbors, every hidden neighbor is safe.
- If a cell is identified as safe reveal it and update your information
- If a cell is identified as a mine, mark it and update your information
- If no hidden cell can be conclusively identified as a mine or safe, pick a cell to reveal at random.

# An Improved Agent

The algorithm described for the basic agent is a weak inference algorithm based entirely on local data and comparisons - it is effectively looking at a single clue at a time and determining what can be conclusively said about the state of the board. This is useful, and should be quite effective in a lot of situations. But not every situation - frequently multiple clues will interact in such a way to reveal more information when taken together.

Your improved agent should model the knowledge available, and use methods of inference to combine multiple clues to draw conclusions when possible or necessary. Not that 'knowledge' to model includes potentially: a) whether or not the square has been revealed, b) whether or not a revealed cell is a mine or safe, c) the clue number for a revealed safe cell, and d) inferred relationships between cells.

# Questions and Writeup

Answer the following questions about the design choices you made in implementing this program, both from a representational and algorithmic perspective.

- Representation: How did you represent the board in your program, and how did you represent the information/knowledge that the clue cells reveal? How could you represent inferred relationships between cells?
- Inference: When you collect a new clue, how do you model / process / compute the information you gain from it? In other words, 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 we be sure of this, and if not, how could you consider improving it?

- Decisions: Given a current state of the board, and state of knowledge about the board, how does your program decide which cell to search next? Aside from always opening cells that are known to be safe, you could either a) open cells with the lowest probability of being a mine (*be careful - how would you compute this probability?*) or b) open cells that provide the most information about the remaining board (*what could this mean mathematically?*). Be clear and precise about your decision mechanism and how you implemented it. Are there any risks you face here, and how do you account for them?

- 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?

- 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?

- Effeciency: 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 impove on?

- 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.

# Questions and Writeup Answers

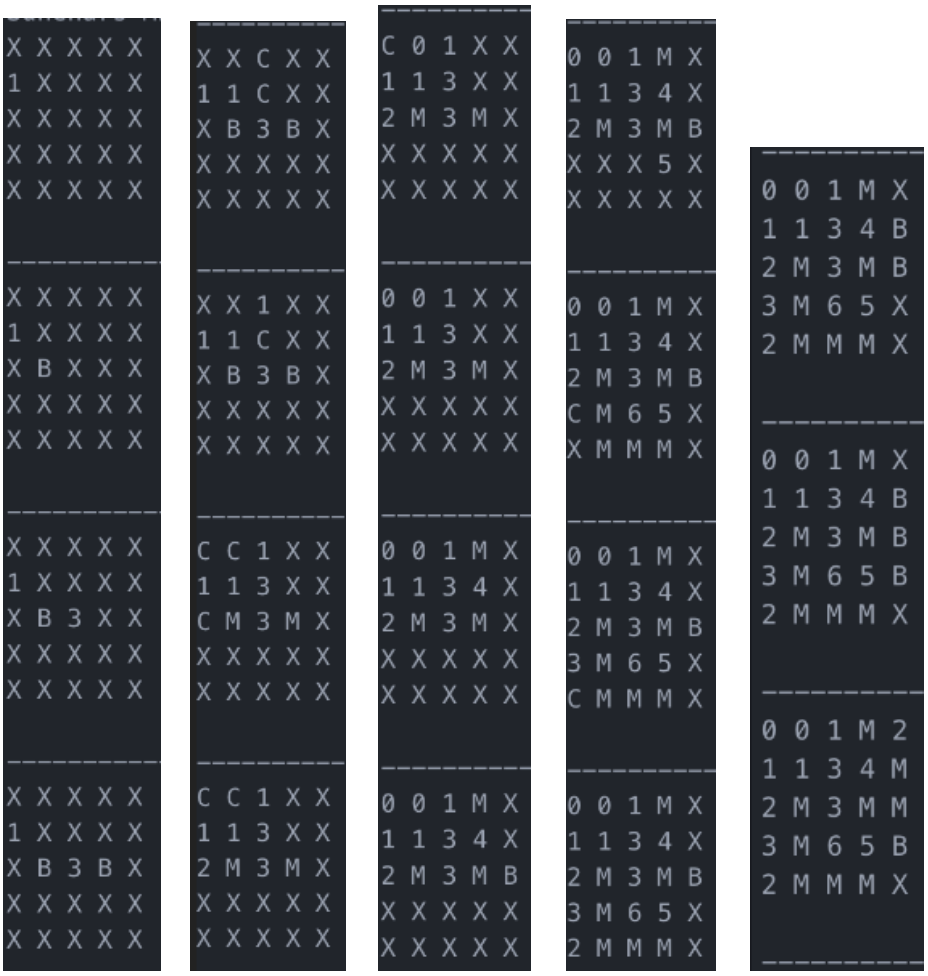
1. **Representation**:

   **Basic Agent** (*File: MinesweeperAgent.py*)
   ***For visualization see Figure(s): 1*

   (a) The board is represented and constructed by a 2D matrix and is visualized using the terminal checking each tile's status by iterating every clue that we found and declared as a mine, cleared tile, or bomb that has exploded. (MinesweeperAgent.py)

   (b) The information is represented as a list(fringe) which checks if the current tile has a mine. We enumerate respectively: true = 1, false = 2, hidden = 3. We keep track of potential mines and previous mines in lists and use them to update our knowledge base. It uses previous clues to make more informed decisions for picking future tiles.

   (c) For the basic agent, the inference starts with the random tile from the minesweeper grid, and we explore the adjacent tiles to figure out the clues one by one. According to the clues that are found, we add them to the list and keep declaring what is possible by flagging a cell as either a mine or revealing it as a clear tile.




(a) 1 (b) 2 (c) 3 (d) 4 (e) 5

(f) 6

Figure 1: Basic Agent Terminal Visualization [*Grid: 5 x 5, Mines = 10*]
*The above figure is read from top to bottom*
***KEY**:*
***C** = Flagged 'Clear' by Basic Agent*
***B** = Mine detonated by Basic Agent*
***M** = Flagged as Mine before/after detonating*

**Improved Agent** (*File: ImprovedVisualization.py*)
***For visualization see Figure(s): 2 & 3*

(a) For the Improved Agent, we implemented a graphic plot , showing the average percentage of the each score of the game that we play, and implemented a GUI visualization of minesweeper. (*see Figure(s): 2 & 3*)

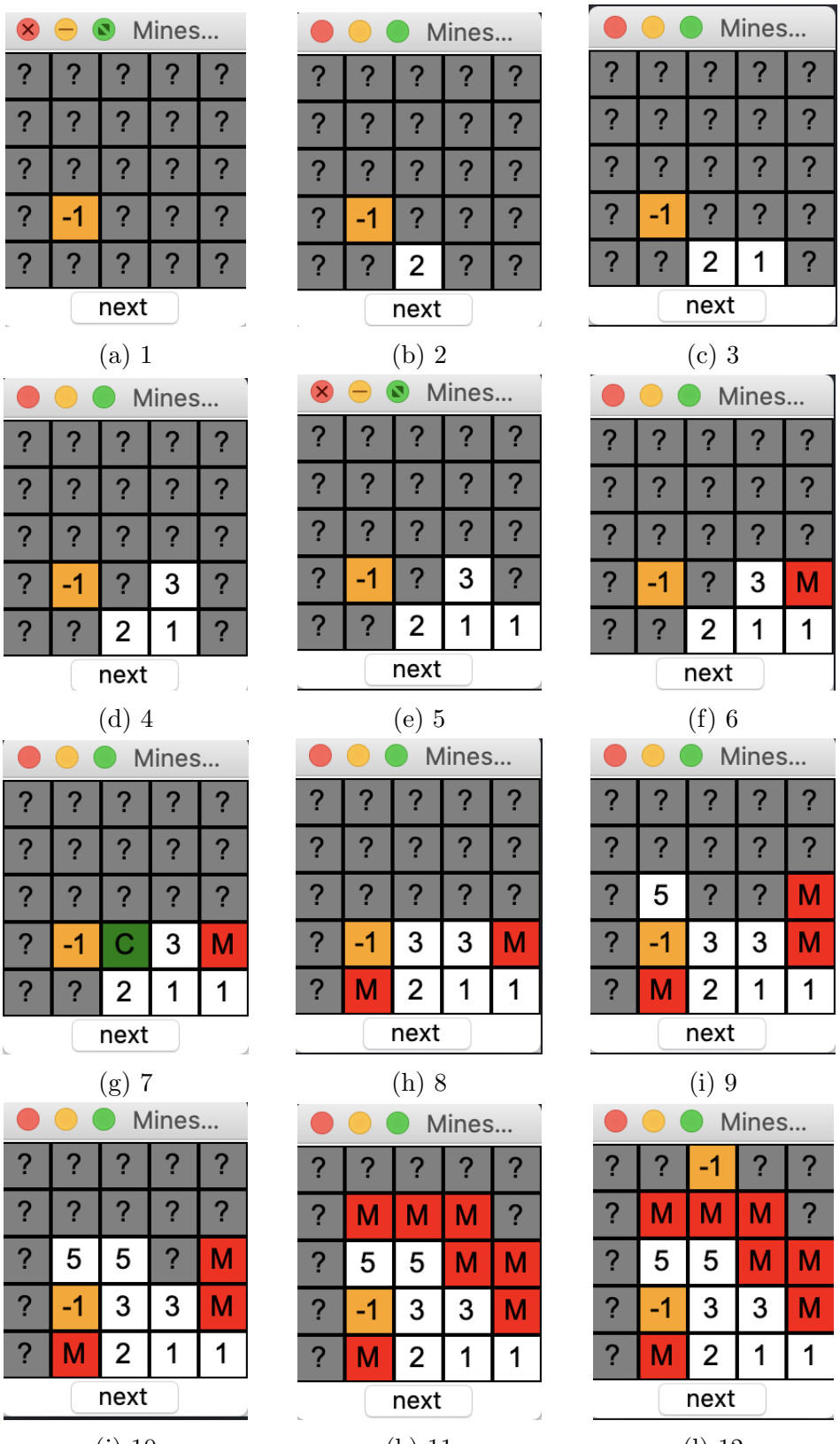
(b) The information is represented as a list(fringe) which checks if the current tile has a mine. The improved agent performs probability checks if there is no tile to safely visit. The agent takes a guess if there is nothing clear and probability checks every viable option.

(c) According to the baseline inference, which is a function called inferenceStart, the agent requires satisfiability of several conditions to check before revealing the tiles without guessing the random tile.

   i. First, there should be two random tiles that are selected near enough, but they should not invade each other's adjacent fringe for the later computation of the possible probability.

   ii. Second, when there is a high enough probability to declare a tile as mine, then the agent declares it as a mine. Otherwise, we sequentially have to reveal another outer random tile that is near to the two other randomly selected tiles.

   iii. Third, when the agent deduces more than one clue at a time, then it can start inferencing multiple clues and generally chooses the adjacent random tiles **only** when it's necessary to reveal the hidden tile by guessing.




(a) 1 (b) 2 (c) 3

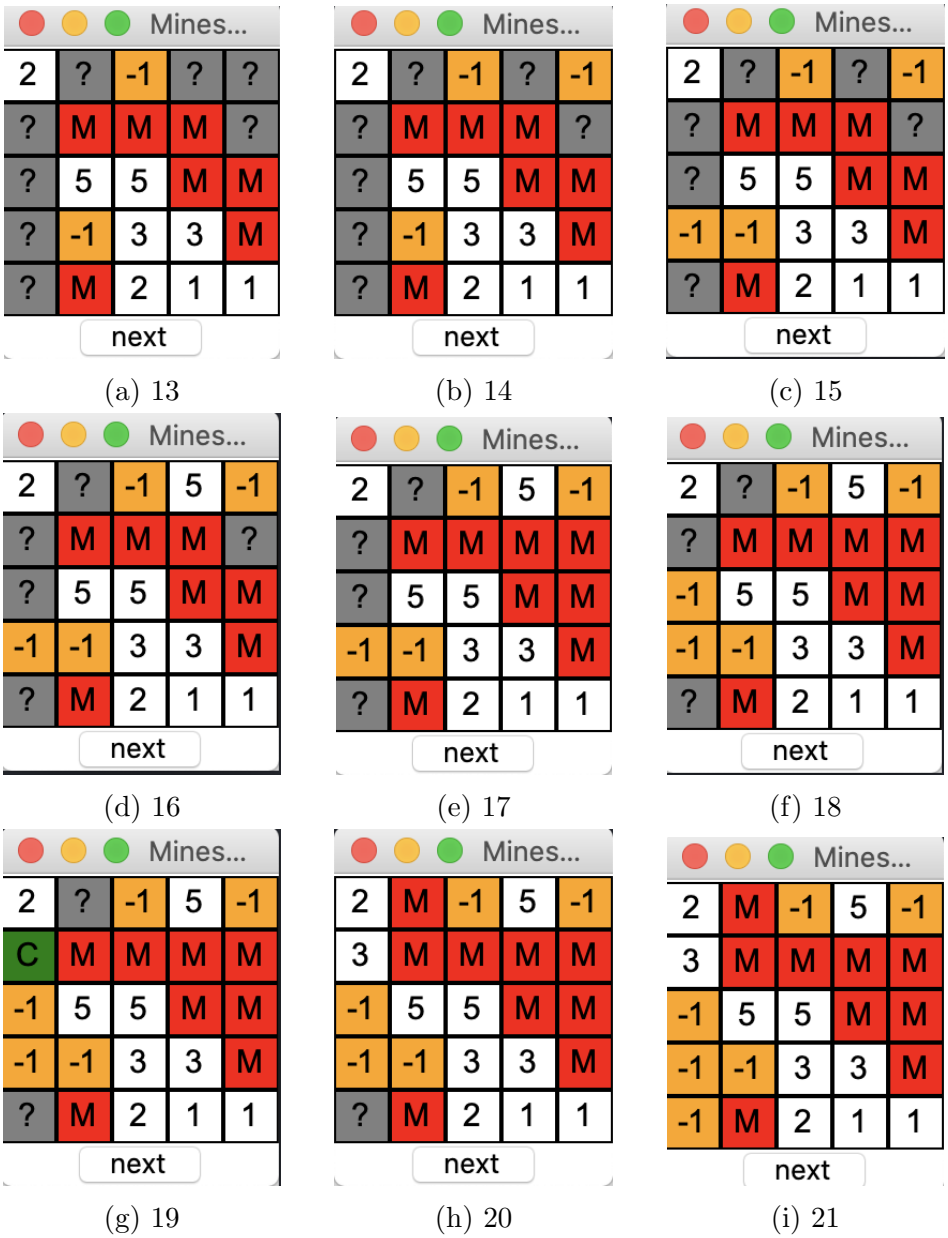
(d) 4 (e) 5 (f) 6

(g) 7 (h) 8 (i) 9

(j) 10 (k) 11 (l) 12



Figure 2: Improved Agent Terminal Visualization *[Grid: 5 x 5, Mines = 15]*
*The above figure is read from left to right*
***KEY**:*
***C** = Flagged as 'Clear', **-1** = Mine hit, **M** = Flagged as 'Mine'*




(a) 13 (b) 14 (c) 15

(d) 16 (e) 17 (f) 18

(g) 19 (h) 20 (i) 21

Figure 3: Improved Agent Terminal Visualization Continued...

2. **Inference**:

**Basic Agent**

(a) After collecting a new clue, our knowledge base implements the use of a fringe by checking all the adjacent neighbors

(b) There are 2 things we need to do:
   i. Compute value of the block we have just explored
   ii. When we check a cell at random, we return -1 if there is no mine or a number if there are neighboring mines. Then we choose an adjacent neighbor, however, we do not check it. We check the adjacent neighbor's neighbors and choose a cell. We compute the probability of those neighboring cells and when we go back to the original cell we chose, we add the probability of that number's cells. Based on the computed probabilities, the ones with the higher probabilities are flagged as potential mines. We continue this process and see if our agent makes the right choices. However, one thing to note is that since combining clues can be time-consuming without a proper direction or heuristic, it's best to stop combining with an already known clue. For a given knowledge base, even if we add more hints to it, the statements we can deduce are still the same. It is not economical to spend so much time combing clues when we can explore different deductions from somewhere else because new hints might lead to the same exact statements.

(c) There are still ways to combine clues together to improve the program. The structure of some hints, for example, value relationship while geometric relationships tend to be more solvable. We can improve our proof by contradiction by combining a large number of hints at one time, however, in testing this makes it more time-consuming.

**Improved Agent**

(a) The model for the improved agent finds multiple clues at once as long as the agent sufficiently finds the clues based on the probability that we compute. To do that, our random sequence of tiles requires sufficient enough tiles (usually more than one or two) which they prefer to not engage next to each other but rather explore the outer part from the adjacent tiles of the tile that was initially picked.
   i. We added the inference, called probability_inference, which will compute neighbor tiles to declare which one can be the possible mine or possible hidden cleared tiles.

(b) Once the condition is satisfied, the improved agent starts exploring what can be the possible mine by comparing the probability of potential mine location. Also, the agent can declare the revealed or hidden clear tiles by finding the lowest probability around the tiles that are revealed. Revealed clear tiles can be visited by the agent and the agent declare the visited tile's number of the mines that are surrounded.

(c) Not necessarily deducing everything, but, as long as the agent satisfies the condition for revealing multiple tiles that are safe enough to reveal, the agent continuously deduces the tiles and keep adding the clues into the queue.

3. **Performance - first part**:

**Basic Agent**

(a) When we check the play by play progression of the game, we find some uncertainties.. This might happen erratically from the basic agent because the inferences found from the basic agent were not enough to declare the next visiting tile whether it is safe enough to reveal or not. Also, there were some uncertain moves for visiting the random or guessing tile although there were other safe enough tiles that could have been visited.

(b) The primary reason for this is that the agent has a difficult time handling complicated conditions in order to make a decision for revealing the tile.

‘

**Improved Agent**

(a) Since the improved agent requires more inferences and concrete enough conditions before revealing the tiles, there wasn't necessarily uncertainty. The part that was quite surprising was the time complexity that the agent required to solve the minesweeper-which was much faster than the basic agent. This might happened because the inferences were all stored in the queue and used to find multiple clues at once.

4. **Performance - second part**:

**Basic Agent**

(a) The result of the graph plot makes sense due to the limited inferences and insufficient clues that the agent can consider for revealing the tiles. Essentially, minesweeper becomes harder when the density of mine increases.

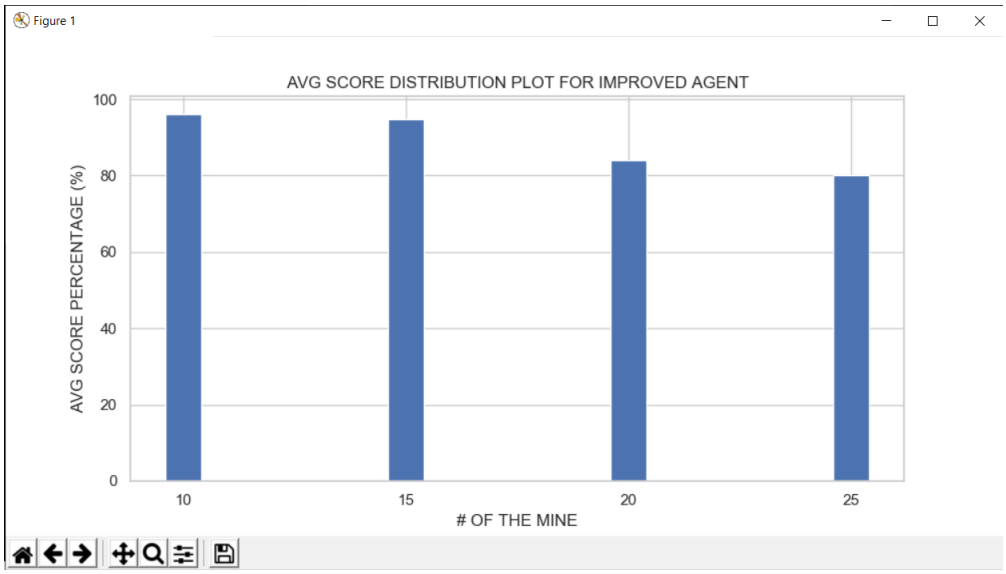
**Improved Agent**

(a) The result of the graph plot makes sense due to more inferences and clues that the agent can take thus it is able to make a concrete decision for revealing the tile. However, minesweeper still becomes 'harder' as the density of mine increases due to more information that has to be taken into consideration.

**Performance Comparison between Basic Agent and Improved Agent**

(a) Occasionally, the simple algorithms from Basic Agent were slightly better by comparing the average score for solving the minesweeper, generally, it happens when the number of mines was between 10 and 15. However, algorithms from the improved agent are better when it comes to solving minesweeper with more than 20 mines. This happens because inferences with more constraints from the improved agent are safer and more accurate to compute at a time. *See Figure:4 below for comparison*

(a) Basic Agent




(b) Impoved Agent

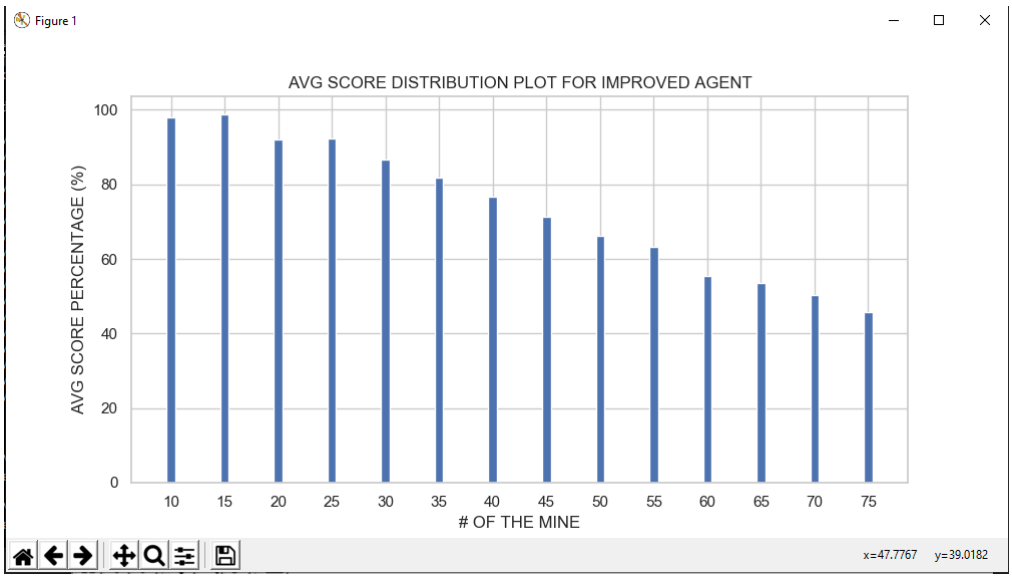



Figure 4: Basic Agent vs. Improved Agent
*- In order to generate/replicate results for Figure: 4 above you can go to line* ***225*** *in* ***MinesweeperAgent.py*** *and uncomment code. Additionally, you can change other variables as well: number of mines, number of games / iterations, and the dimension size.*

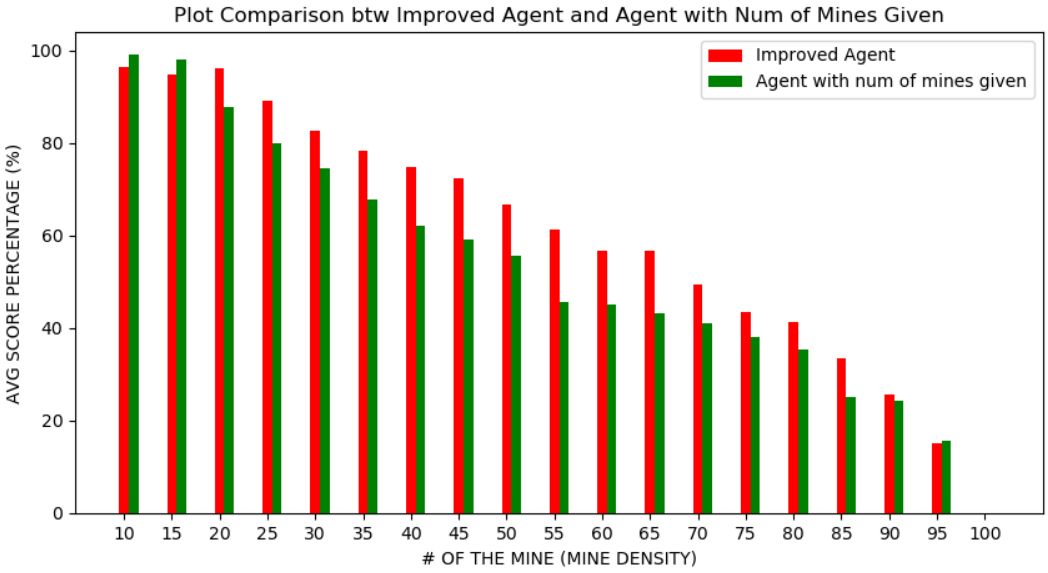



Figure 5: Plot Comparison: Improved Agent vs. Agent w/ # of Mines Given.
*- In order to generate/replicate results for Figure: 5 above you can go to line **555** in **ImprovedAgent.py** and adjust code. Additionally, you can change other variables as well: number of mines, number of games / iterations, and the dimension size.*
***As you can see from the plot above, the agent with the number of mines given [**green**] outperfomed the improved agent when the mine density was **10** and **15** respectively. However, the improved agent performed better overall when the mine density was greater than **20**.*

5. **Efficiency**:

**Basic Agent**

***Time Complexity***: $O(n^2)$ *to* $O(n^3)$
***Space Complexity***: $O(n^3)$

(a) If the environment has a bigger dimension size with a higher density of mines, it is possible to run into a longer time to solve minesweeper. Also, the baseline inferences were updated by the basic agent one at a time. Hence, a higher density of mines took quite a lot of time to retrieve the plot. The blueprint of the basic agent was to iterate a global dimension to check whenever the

possible mine or safety square is speculated and update the global dimension with inferences. This process is essential for adding each clue with inferences into the list, and this list will be used in the further steps or move that the basic agent tries to define. However, that is not to say that it is all due to the capabilities of the basic agent. It can also be implementation-specific, as there is possibly a more "efficient" implementation we did not consider. For example, the knowledge base database has to be thoroughly computed and stored whenever the basic agent decides the next move based on the probabilities that we found from current inferences. The problem was occurring when it takes a massive amount of time for iterating and computing all the probabilities for each square. Therefore, we concluded that the design for the basic agent could be inefficient due to the time complexity for considering the optimal decision to make without trying step into the random query. Therefore, we define the efficiency of the basic agent constraints as implementation-specific constraints.

**Improved Agent**

***Time Complexity**: O($n^2$) Occasionally, O($n^6$)*
*Sidenote: Generally it is O($n^2$), but there is a small instance where it is briefly O($n^6$) when the agent is speculating adjacent tiles.*
***Space Complexity**: O($n^2$)*

(a) For the improved agent, the issue of time was not at all the same as the basic agent. Testing the results for the improved agent was many times faster. So, space or time constraints were merely the same with the basic agent's inferences that were computing the global mine squares and adjacent squares to compute the probabilities for every square, which were close enough to the revealed squares. This process was very efficient because the improved agent is not entirely selecting the random square but rather selecting the one nearby the other two randomly selected squares that aren't revealed as mines.

The main difference could occur from problem-specific constraints because there are more specific requirements to use multiple clues at a time with computing all the possible probabilities for each square. This can be deduced to the fact that the improved agent

takes "more" info to make decisions. And, the decision comes up with declaring both safe enough to mark the square as "cleared" and risky enough to mark the square as "flag."

6. **Improvements**:

   (a) Given that if the board knows the number of bombs, we can keep a count of how many mines are left.

   (b) Our knowledgebase can utilize the facts that it knows how many mines are left and once it computes the probability values for the adjacent cells, it would be able to pick the best option, given all of the information.

   (c) Then we would decrement the counter and every time the bomb count becomes less, the knowledge base updates and knows which cells are safer than others more effectively.

   (d) In any instance, if we pick a cell in the area of the explored region or remaining part of the board, we can estimate those probabilities for the cells around that area. If the minimum of those probabilities is greater than the probability of finding a mine in the rest of the region, then exploring the rest of the region is an optimal move. While exploring the next region we assign priority based on the connectivity of the cell to the variables in the constraint list

# Contributions

**Jae: All code for following files: *GameSetting.py, ImprovedAgent.py, ImprovedGamesetting.py, ImprovedVisualization.py, MinesweeperAgent.py*, and Revised Questions and Writeup)**

**Cesar: Creation of LaTex doc**

**Sunehar: Questions and Writeup (*Original*)**