CS 440
Spring 2021

Project 2: Minesweeper
Author: Alden Lu, Haoran Wen.

1 Preface: How to Run the Program

To start the program: python3 main.py [dimension] [number of bomb]
After starting the program use the following commands to run the agents.

CL CMD	Action
SPACEBAR	Start game
a	Run Improved Agent
b	Run Basic Agent
RETURN	Reset Board

The acronyms for the improved agent is IBM Agent which stands for Inferencing Bombs Minesweeper Agent.

2 Problem 1: Representation

The minesweeper board is represented in a 2d array of cell/node/tile objects. Each cell object would keep track of its clue, its neighbors, and its state (bomb, safe, flagged as bomb, flagged as safe). The cells and its clue are visualized into a grid board using pygame.

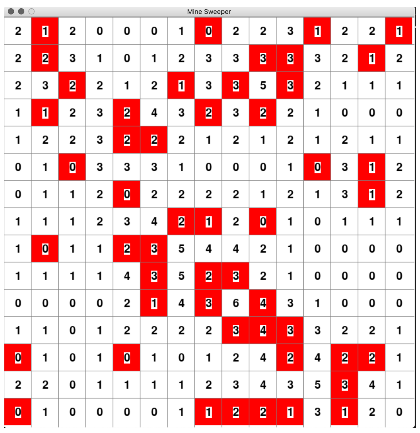

Figure 1: Actual Board

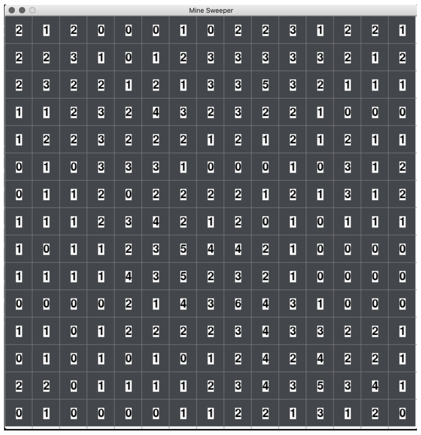

Figure 2: Start Game Board

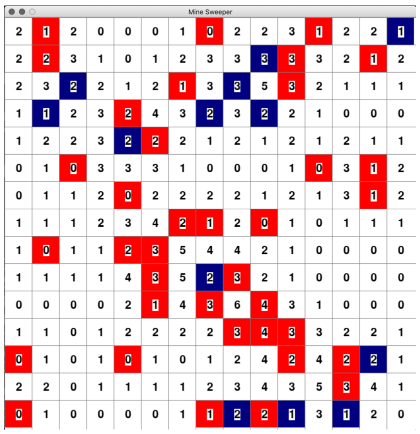

Figure 3: Solved with Basic Agent

Use use the color feature of pygame to help us represent the state of the each cell and the board as a whole when solving the board.

Cell Color	Representation
White	Safely flagged a cell as safe
Red	Safely flagged a cell as bomb
Blue	Clicked on a bomb

For knowledge representation we use with dictionaries to model our knowledge base of information, with the key value pair as satisfaction variable and variable value respectively. We also use a list of unexplored and revealed cells, in conjunction with the knowledge dictionary for the agents to play through the game.

When a cell is clicked in would reveal if the cell is safe or a bomb, provide the knowledge of the clue of the cell, and construct the constraint equation ($i.e. A+B+C+E+D+F+G+H+I = clue$). From the clue, the equation and the lists and dictionary mentioned above, we are able to deduce the number of neighboring bombs and safe cells, number of revealed neighboring mines, number of revealed neighboring safe cells and number of unrevealed neighbors. If the clue is equal to the number of neighbors then we can infer that all neighbors are bombs and if clue is 0 then all neighbors are safe.

Using the dictionary of variables we can create equations that the improved agent to inference more information.

3 Problem 2: Inference

Basic Agent
The basic agent keep track of two lists, a list of current hidden cells and a list of unclicked/unopened cells currently on the board. The agent also keep track of a dictionary of revealed and flagged cells on the board. Using the 2 lists and dictionary the agent solves the board by clicking on cells and updating the lists and dictionary appropriately.
When the agent is presented with a cell and its clue, it extract all the information it can out of the current cells: its neighbors, its hidden neighbors, its revealed bomb neighbors, and its revealed safe neighbors. Using the extracted information about the current cell along with the dictionary and 2 list mentioned above we are bale to infer about the hidden cells around the current cell.
If the clue is equal to the number of neighbors the cell has then we Can infer that all surrounding cells are bombs.
if the clue is 0 then we can infer that all surrounding bombs are safe. The basic agent only attempt to deduce the state of cells that are neighbors to the currently clicked cell. At each query the agent updates the dictionary of facts, the list of clicked/opened cells and the list of revealed cells and will all cells in the board has been revealed then the program returns the score.

Improved Agent: Inferencing Bombs Minesweeper (IBM) Agent
The inferencing preformed by the IBM agent is essentially proof by contradiction. Given a set of equations that contain only consist of variables that are still hidden to the agent, it will pick a least involved variable or in other words a variable that shows up the least in all of the equations. Then the program will assume it as a bomb and add this "knowledge" in a dictionary. Because we assume that this least involved variable is a bomb, we iterate the all of the equations and update its value. For example, given our least involved variable is Cell ID: 4 and given we have the equation $4 + 1 + 2 = 1$(Cell 4 + Cell 1 + Cell 2 = 1 bomb), we will update our and thus the equation becomes, $1 + 2 = 0$. Essentially, it will find the corresponding value

from the dictionary and subtract it. Because of this equation, we see that 1 is safe and so is 2. Thus we add this to the dictionary and update the equations again. We know there will be a contradiction when we get an equation whose numbers of variables are less than the value of the equation (e.g. $1 + 2 = 4$) or when a value of the equation will be less than 0. Once we know that there is a contradiction, we can then assume that it was not a bomb and add it to a list for which contain only safe cells we can query.

4 Problem 3: Decision

Given any state of the board, if there is a safe cell that we have not query or marked as safe yet, we will query it to find more safe cells. If there are more safe cells, we will keep querying them. If there are no more safe cells, we will try to inference for one, and if we do not find a contradiction, we will finally pick at random of the list of hidden cells to the agent. If we do find a contradiction, we will throw that cell that we assumed wrong into the safe cell list for querying

5 Problem 4: Performance

According to the Minesweeper Board Museum typical board configurations are 8x8 with 10 bombs, 16x16 with 40 bombs, and 16x30 with 90 bombs. As this project ask for a square board we choose to use a the second configuration option of board size of 16x16 with 40 bombs. The figure below shows an example of a 16x16 board with 40 bombs. In the figures below the basic agent and IBM agent are playing two different boards.

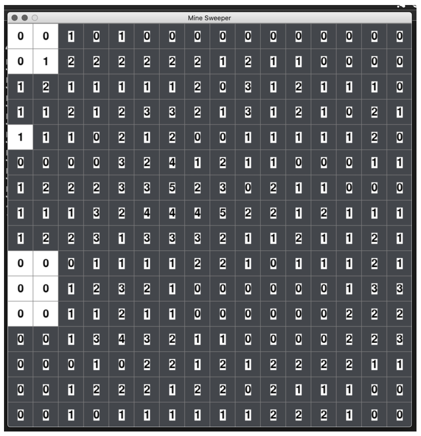

Figure 4: Basic Agent: Early Game State

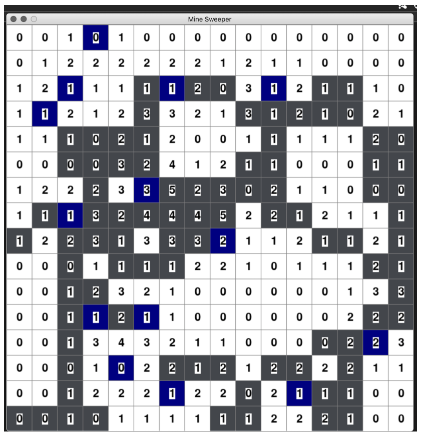

Figure 5: Basic Agent: Mid Game State

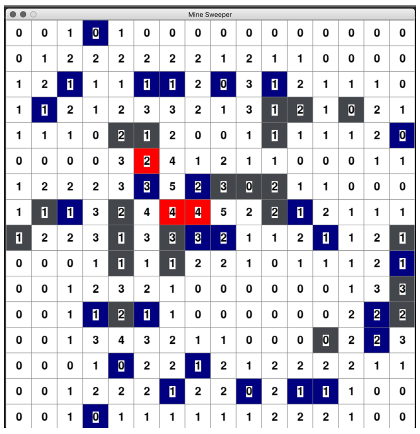

Figure 6: Basic Agent: End Game State

There are points in the game where both agents make sub optimum moves. These moves occur when the agent can no longer deduces anymore information and randomly choose an available cell to click on. Because the cell is chosen at random there is a chance that it clicks on an hidden bomb. The IBM agent is able to deduce and flag the locations for bombs a lot more effectively per query compared to the basic agent, and thus the IBM agent steps on less

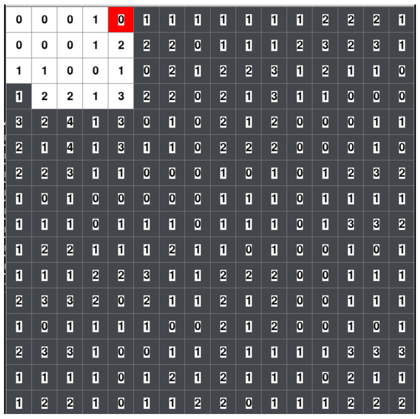
Figure 7: IBM Agent: Early Game State

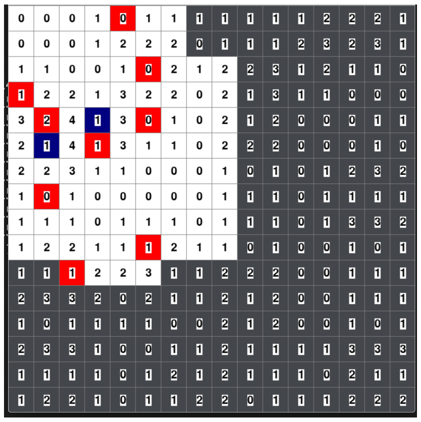
Figure 8: IBM Agent: Mid Game State

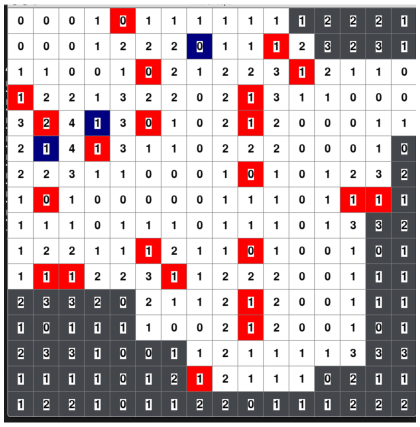
Figure 9: IBM Agent: End Game State

bombs than the basic agent.
While the each agent might query a bomb cell at random, it does so as randomly choosing an available cell to query is how we programmed the agent to function, so the agents decisions are not surprising as it is preforming as we expected it to.

A video of the program running can be found here: https://drive.google.com/file/d/1g1d2xlPXqY1AzowKIEW

6 Problem 5: Performance

The figure below shows a comparison of how the basic agent algorithm runs compared to the improved agent algorithm The results from the figure above does reflect our expectations of

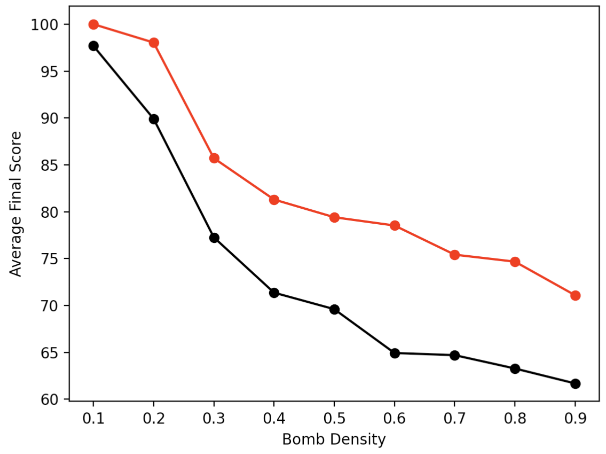

Figure 10: Dim: 16, Black: Basic Agent, Red: Improved Agent

how the two agents compared to each other. The improved agent consistently beat out the basic agent across all bomb density. From the graph the point were the minesweeper game gets hard at around 0.3 bomb density as the success rate drops below 90 percent.

7 Problem 6: Efficiency

As we are implementing this program on our personal computers. Hardware limits plays a role in our how fast our project runs. This was evident when comparing the runtime of the basic agent to the improved agent. As the improved agent preforms far more computations then the basic agent it takes far longer to run compared to the basic agent. While the compare hardware constraint is a problem specific constraint, we also had some implementational constraints that could have affected the space and time efficiency of our program. Our program utilizes multiple instances of lists and the iterating through these lists. Due to how many time the program iterates through list while preforming computations, especially when attempting to inference for new information, this would affect the runtime of our program. Some improvements we could have invested in would be to figure out a way to smartly prone the list of equations when inferencing or some sort of heuristic in picking which cell to assume for a contradiction so that we only need to iterate over the smallest amount of necessary equations to solve to arrive at a terminal conclusion. This we decrease the number of equations we are solving at each step of the inference. And since the equations are stared in lists, this would decrease the number of resources used at each step.

As data collection is a part of the project we were restricted on how large of a board we can use when gathering the data.