Intro to AI: Project 2 - Minesweeper

Ashwin Haridas (ah1058), Ritin Nair (rrn32)

March 2021

1 Introduction

In this assignment, we are tasked with designing an agent that plays the classic game "Minesweeper". The goal of this game is to safely identify all possible mines without accidentally "tripping" them. We implement two agents (basic and advanced) to play the game. Specific details on the implementation of each agent are provided below.

2 Basic Agent

The logic behind the basic agent was taken from the project description, and is fairly intuitive. The basic agent identifies if all hidden neighbors of a cell are mines if the clue of the cell minus the number of revealed mines is the number of hidden neighbors. All of the hidden neighbors of a cell are identified as safe if the total number of safe neighbors minus the total number of revealed safe neighbors is the number of hidden neighbors.

3 G.E.M. (Advanced Agent)

The basic agent is a weak inference algorithm that only looks at a single clue at a time to make conclusions. This is effective in many situations, but we can do better. Our improved agent is called "G.E.M", which stands for **G**aussian **E**limination in **M**inesweeper. This is a stronger inference algorithm, as we use multiple clues to reveal information when taken together. This algorithm actually uses the basic agent's algorithm and then adds on to it. Essentially, we first create systems of linear equations based off of these clues. Then, we combine these equations into one matrix and perform Gaussian Elimination on the matrix in an attempt to reduce it. Using a system of linear equations allows us to look at multiple clues at once. After the reduction, we look at the matrix and see if we can obtain any useful information (i.e. a cell is safe or a mine). This agent is an improvement from the basic agent and is analyzed through a series of questions that are answered below.

4 Questions

4.1 Representation

How do you represent the board in your program, and how did you represent the information / knowledge that clue cells reveal? How could you represent inferred relationships between cells?

In this program we represent the board using a 2D matrix that stores integer values at each cell, which each hold a specific meaning depending on the value of the integer. If a cell has a value of -1 it means that the status of the cell is unknown. If a cell has a value between 0 and 8 it is identified as a clue cell. These cells are known to be safe and the clue value on them refers to how many of their neighbors are mines. However, it does not reveal exactly which of its neighbors are mines. Furthermore, the maximum clue value for corner cells is 3, the maximum clue value for border cells is 5, and the maximum clue value for regular cells is 8. A cell that contains a mine is represented by marking it with the value 9.

This program uses a dedicated knowledge base (represented by a one-dimensional list) to store the information and knowledge that clue cells reveal. The coordinates of safe clue cells that have been revealed are added to the knowledge base, and then information regarding the current state of these cells is calculated. One thing to note is that we decided to not have a dedicated structure to store specific information about the cell. Instead, we have functions (that return this information) that we call as we look through our knowledge base of cells whenever we need said information. This allows us to be have better space efficiency. The information pertaining to a certain cell is recomputed at each iteration of the advanced agent algorithm to ensure that the information is up to date with the current state of the board. Choosing to represent the information in the knowledge base this way comes at the cost of some time complexity for computing the amount of safe neighbors, mine neighbors, and hidden neighbors around each cell in the knowledge base. We choose to use this method for representing information in the knowledge base regardless of this because it provides the highest guarantee that the information will be reflective of the current state of the board.

To represent different clues in the knowledge base interacting to reveal relationships between cells that we can infer to be true we use another 2D matrix. This matrix is used to represent a system of linear equations. Each row in this matrix acts as an equation that can model all the information we know about a particular cell in the knowledge base, and each column represents the coordinate that corresponds to the hidden neighbors of a cell. The columns of this matrix include every hidden neighbor of all the cells in the knowledge base, except for the last column which represents the total number of mines a cell has. The rest of the values of the matrix are either a 0 or a 1, with a 0 meaning that a cell does not have a specific hidden neighbor corresponding to the column and a 1 meaning that it does possess this hidden neighbor. Values of the matrix with the value 1 (except for ones in the rightmost column) essentially act as a variable representing a hidden neighbor. If this variable is solved to have a value of 0 it means it is safe, and if it has a value of 1 it means it is a mine.

4.2 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 you be sure of this, and if not, how could you consider improving it?

When a new clue is collected it is first added to the knowledge base. More specifically, the coordinates of the cell with the clue value are added to the knowledge base, and information about this cell is retrieved later when necessary. The necessary information needed about the cell includes the number of mines already found around the cell, the number of safe cells around the cell, and the number of hidden neighbors. This information is first used in the basic inference algorithm which was provided to us. If a basic inference is not able to be made with an individual clue alone, then it is beneficial to combine several clues within our knowledge base together to figure out if the interactions between various clues allows us to make any further inferences. Our algorithm combines every clue in the knowledge base together in the form of a 2D matrix, where each row represents a clue and each column represents a hidden variable (except for the last column which represents the total number of mines the cell a row represents has left in its hidden neighbors). Every time a new clue is added to the knowledge base, it is then also added to this 2D matrix as an extra row. The algorithm uses row reductions and Gaussian Elimination to simplify the matrix into reduced row echelon form with the purpose of seeing if all the clues in the knowledge base can reveal additional information due to the way different clues could potentially work together in deducing new information about neighboring cells.

Every time a clue is added to the knowledge base this Gaussian Elimination procedure is repeated to update the current state of knowledge that the algorithm has available based on the new clues that have been revealed on the board. Adding a new clue to the knowledge base also does not necessarily mean that everything related to that specific clue that was added will immediately be figured out, but rather that it could potentially allow the algorithm to figure out additional information about previous clues in the knowledge base too. Therefore, this approach figures out everything possible about all the clues in the knowledge base given the current state of the board and all the other additional clues in the knowledge base. While this does not guarantee that we are able to make inferences about every clue in the knowledge base for sure, it is a very good approach to do so given the current information the algorithm has available. The use of a matrix with every clue in the knowledge base also allows each clue to interact with every other clue in the knowledge base at one time, rather than having to check every possible combination of clues individually.

Using a system of linear equations and reducing it to reveal information is generally a good approach, and the actual results are summarized / analyzed in a later question. However, we could improve our inference model by considering adding different types of inference. For example, we can consider adding an algorithm for constraint satisfying search, which would essentially find satisfying assignments for the variables we create from our knowledge base. We can also consider adding an algorithm for expert querying / proof by contradiction. We could use the system of linear equations that was created before and propose an assignment for a variable and see if that

would violate any conditions. If it did violate anything, we would have a contradiction which could tell us legitimate information about that variable.

4.3 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?

Our program uses the current state of the board and a state of knowledge about the board represented by a 2D matrix of all the clues in the knowledge base to make inferences about cells and decide which cells to reveal next. After the matrix of clues in the knowledge base is simplified to reduced row echelon form, each row is checked to see if certain conditions are met. Depending on which conditions are met it can either be inferred that a cell is safe, that a cell is a mine, or that there is no meaningful information that we can extract from the knowledge base about a cell. Using this approach to make inferences about hidden cells on the board before choosing them helps reduce the odds of randomly picking a mine.

One way the program checks the rows of the matrix produced by the knowledge base to make inferences is by checking if a row is all 0s, 1s, and -1s (except for the last column). If this is the case and the last column is equal to the clue value of the row minus the number of mines that cell has already had revealed on the agent board, then the algorithm is able to make an inference about the hidden cells involved in the equation that the row represents. It is safe to say that all hidden cells represented by a column with the value 1 in this row must be a mine, and all hidden cells represented by a column with the value -1 in this row must be safe. This stems from the idea that the variables in the equation each row represents can only be assigned the value 0 if they are safe and 1 if they are a mine. Limiting the domain of the variables in this situation allows us to make inferences based on these equations in cases where they would normally yield infinite solutions.

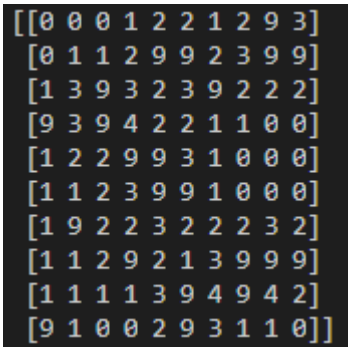
Another way the program checks the rows of the matrix to make inferences is by checking if a row has only values of 0 and 1 (except for the last column), and the last column of the row has a value of 0. This implies that the hidden cells represented by the columns in the row that have a value of 1 must be safe. We are able to make this inference because the domain of the variable values in the equations the rows represent are limited to only being either equal to 0 or 1, depending on if the cell is safe or if the cell is a mine respectively.

Once we are able to determine if a certain set of cells are mines or safe cells, we can make a decision on what to do next. If, either from basic inference or advanced inference, we find that a certain number of cells are safe, we can go ahead and search all of those cells immediately. This is because we know for a fact that these cells have to be safe based off of our inference algorithms. If we find that some cells are mine cells, we can mark them as mines immediately so that we are sure to never search those cells in the future.

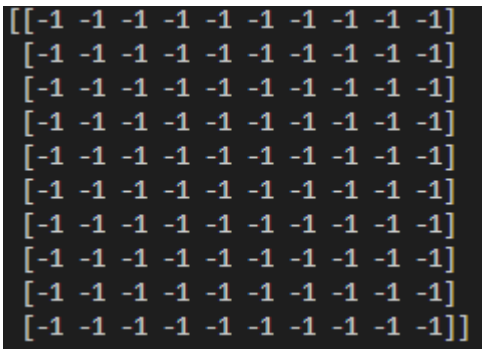
4.4 Performance

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?

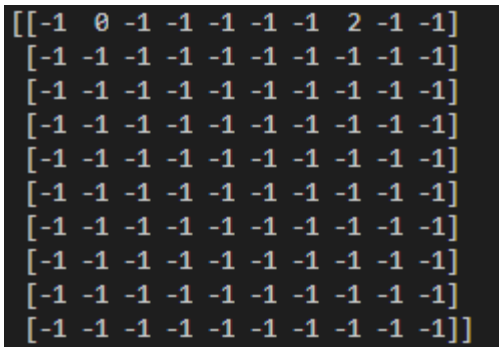
To answer this question, we use a 10 x 10 sized board with 22 mines. Here is what our board looks like:



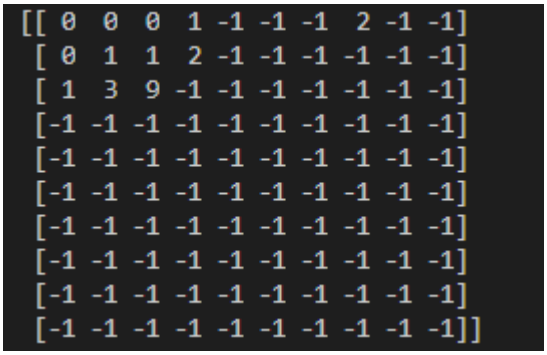
A cell with a number between 0 and 8 is a safe cell with a clue value of that number. A cell with the number 9 is a mine. Initially, before any cell is searched, our agent board is filled with values of -1, which represent an non-visited cell.



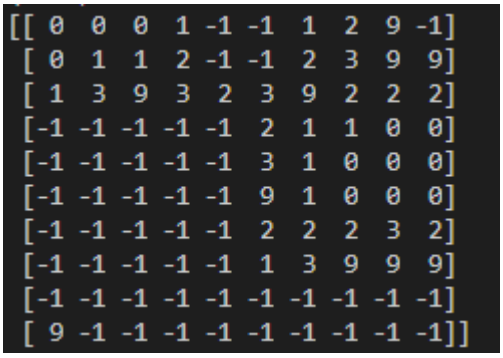
First, our agent makes a few random moves. This is because it is impossible to make inference since there is nothing to make inference of in the beginning. After a couple of random moves, our agent looks like this:



At this point, we notice that at "row 0, column 1" we have a clue value of 0. The basic inference tells us then that all of the neighbors must be safe, so we can visit those immediately. After visiting those cells, the agent was able to use basic inference again and find out that it can mark even more cells as safe and one cell as a mine.



After this, the agent continues to do basic inference and at times a random move, until the following stage:



Here is where our agent can perform advanced inference. The algorithm we designed creates a matrix based off of the clues in the current knowledge base and the unknown neighbors as variables. Row reduction is performed on the matrix to deduce information.

```
[[ 0  0  0  1 -1  2  1  2  9 -1]
 [ 0  1  1  2 -1 -1  2  3  9  9]
 [ 1  3  9  3  2  3  9  2  2  2]
 [-1 -1 -1  4 -1  2  1  1  0  0]
 [-1 -1 -1 -1 -1  3  1  0  0  0]
 [-1 -1 -1 -1  9  9  1  0  0  0]
 [-1 -1 -1 -1 -1  2  2  2  3  2]
 [-1 -1 -1 -1 -1  1  3  9  9  9]
 [-1 -1 -1 -1 -1 -1 -1 -1 -1 -1]
 [ 9 -1 -1 -1 -1 -1 -1 -1 -1 -1]]
```

It appears that from our advanced inference, the agent was able to detect that "row 1, column 5" is a mine and "row 6, column 4" and "row 7, column 4" are safe. Our agent was able to do this by combing multiple clues from the knowledge base. This decision would not be possible with the basic inference, which shows the improved intelligence our advanced agent has.

The agent continues to play the game and makes multiple inferences (both basic and advanced). For example, here's a time stamp where are about to make another advanced inference:

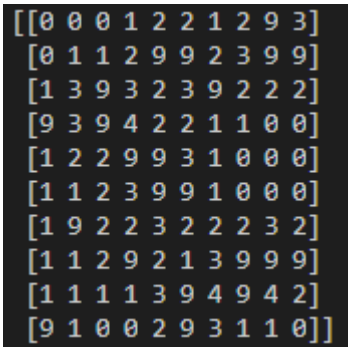
```
[[ 0  0  0  1 -1  2  1  2  9 -1]
 [ 0  1  1  2 -1  9  2  3  9  9]
 [ 1  3  9  3  2  3  9  2  2  2]
 [-1 -1  9  4 -1  2  1  1  0  0]
 [-1  2 -1 -1 -1  3  1  0  0  0]
 [-1  1 -1 -1  9  9  1  0  0  0]
 [-1 -1 -1 -1  3  2  2  2  3  2]
 [-1 -1 -1 -1  2  1  3  9  9  9]
 [-1 -1 -1 -1 -1 -1 -1 -1  4 -1]
 [ 9 -1 -1 -1 -1 -1 -1 -1 -1 -1]]
```

At this point, the agent is able to use advanced inference and mark "row 8, column 9", "row 9, column 9", "row 9, column 7", "row 9, column 8", and "row 8, column 4" as safe.

```
[[ 0  0  0  1 -1  2  1  2  9 -1]
 [ 0  1  1  2 -1  9  2  3  9  9]
 [ 1  3  9  3  2  3  9  2  2  2]
 [-1 -1  9  4 -1  2  1  1  0  0]
 [-1  2 -1 -1 -1  3  1  0  0  0]
 [-1  1 -1 -1  9  9  1  0  0  0]
 [-1 -1 -1 -1  3  2  2  2  3  2]
 [-1 -1 -1 -1  2  1  3  9  9  9]
 [-1 -1 -1 -1  3 -1 -1 -1  4  2]
 [ 9 -1 -1 -1 -1 -1 -1  1  1  0]]
```

Again, this would not be possible with the basic inference, so clearly our agent is an improvement over the basic agent.

Eventually, after more moves, the agent clears out the entire board. This is the final result:

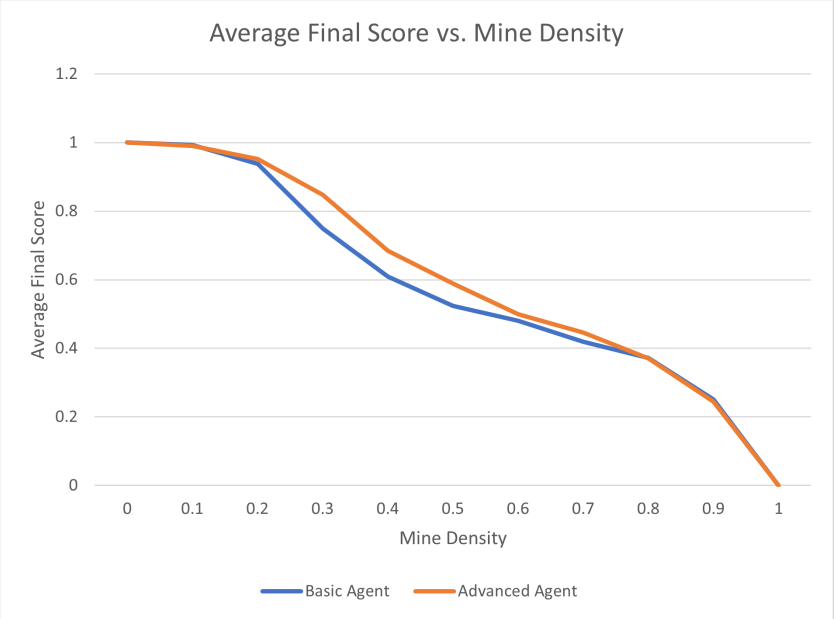


Our agent was able to find 20 / 22 mines, which is pretty good. In general, there were no points where the agent made a decision that we don't agree with. We were also not surprised when the agent was able to make the advanced inference moves (since this is the expected process of our agent), but it was interesting as those moves aren't something we would do if we were playing Minesweeper on our own. Our agent was able to make these decisions because of the inference model it follows, as looking at multiple clues to make decisions is a strong method.

4.5 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? How frequently is your algorithm able to work out things that the basic agent cannot?

Before we present our plot and discuss our findings, let us speak on our initial intuition on what we think will happen when comparing the final score between the basic agent and advanced agent. We think that our advanced agent will outperform the basic agent for all mine densities after 0.1. The basic agent and advanced agent will probably perform relatively the same for densities less than 0.1. Now, let's take a look at the plot:



The above plot was generated by running 50 trials at each density with randomly generated boards of size 25 x 25. From looking at the plot, it appears that we are somewhat right about our intuition. It seems that both agents perform relatively the same in the interval [0, 0.1). Then, in the interval (0.1, 0.8), the advanced agent performs better than the basic agent. Finally, in the interval (0.8, 1], both agents perform relatively the same again.

The graph makes sense. At a low density, both agents tend to perform the same. The reasoning for this is because at such a low density, there will be a very small amount of mines. This means that the basic agent will be able to identify the safe cells very quickly, because most of the cells are safe. Since the advanced agent makes use of the basic agent's inference model, it will also be able to identify the same safe cells. Also, at this density, the advanced agent does not have enough information to begin making advanced inferences, so in the end it behaves exactly like the basic agent. The reason an advanced inference is less likely is because we can only combine clues when there are overlapping hidden neighbors that are mines. At the low density, there is a smaller chance of doing this because of the lack of mines. This causes the behavior of both agents to be the same, and it explains why at low densities the performance is approximately the same.

In the interval (0.1, 0.8), it makes sense for the advanced agent to perform better than the basic agent. During this interval, the amount of mines is evidently more present compared to the previous interval. Therefore, the advanced agent is more likely to have enough information to perform its advanced inference. Because the advanced agent can do this, it will be able to detect more mines than the basic agent (and the basic agent will likely trip these mines from random selection).

In the final interval, (0.8, 1], there is an extreme amount of mines present in the board. In both agents, we will likely continuously trip mines due to constant random selection. This is because there are few safe cells that exist, and when we try to do inference it will likely be difficult to make informed decisions. We can not do inference because there is a lack of safe cells that we

can make inference on. In other words, there are not enough clue values (our knowledge base is small) that can assist us in doing any type of inference. Therefore, we can not make any new decisions which causes us to make more random selections (which could trip mines).

Minesweeper becomes hard after a density of 0.6. This is where both agents start to become close to each other and converge to an average score of 0.

Our algorithm seems to always be better than the simple algorithm, except for very low and very high densities in which both perform about the same. However, we can say that the simple algorithm is better at those densities because it is much more time and space efficient compared to the advanced agent. Since both agents have the same average score at those densities, it may be better to choose the basic agent then.

It seems that our agent is able to work things out that the basic agent cannot at a lower frequency at low and high mine densities, but is able to work things out at a much higher frequency (around 10% and sometimes better) in the middle densities. As mentioned earlier, the algorithm works things out as it has more useful information in its knowledge base, and this happens at those densities.

4.6 Efficiency

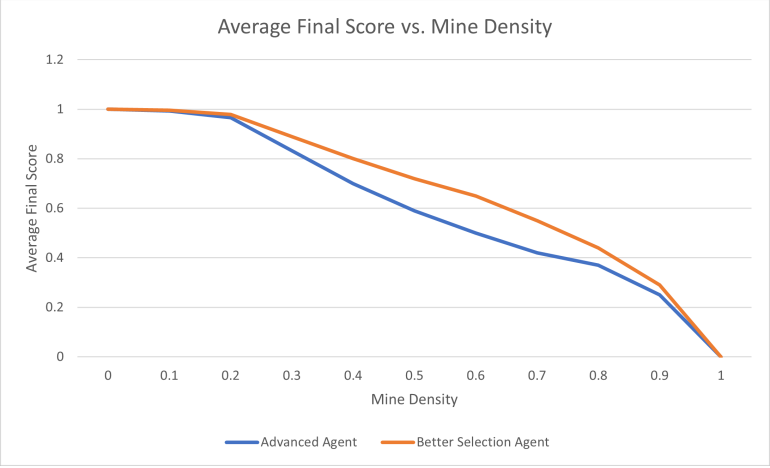
What are some of the space or time constraints you run into in implementing this program? Are these problem specific constraints, or implementation specific constraints? In the case of implementation constraints, what could you improve on?

One space constraint that limits the performance of this program is the way the matrix based on the knowledge base is produced. At each step of the algorithm this knowledge matrix is re-built to account for any updates that the knowledge base experiences. This guarantees that the information that the advanced inferences are based on are all up to date, and that we are not accidentally inferring wrong facts about the board due to out of date information. While this does guarantee that the knowledge matrix will be accurately built with the newest available information that the algorithm has access to at each current step, it does come at the cost of having to create a new matrix at each step of the algorithm where a basic inference was not able to be made. This is not generally a problem at the beginning of the algorithm when the amount of cells in the knowledge base is smaller, but it has a significant effect later on when more cells get added to the knowledge base. A larger knowledge base in turn results in the matrix produced also being bigger as well and taking up more space. This specific constraint is implementation specific rather than being problem specific, so it would be possible to slightly improve on it. Instead of just making a new matrix at each step we could instead have one matrix and update it with the information in the updated knowledge base. However, this would come at the cost of some time complexity to perform options to search for what rows and specific indices of the matrix to update depending on the efficiency of the matrix update process. Therefore, it is beneficial for us to avoid any of these complications and simply remake the matrix at each step to guarantee that it stores the most accurate and up to date information.

One time constraint that limits the performance of this program is the way our advanced agent uses Gaussian Elimination to make more advanced inferences. This algorithm has a big O time complexity of O(m^2n) for an n x m matrix which makes it significantly slower for larger input size boards. These boards tend to have larger knowledge bases, which in turn increases the input size of the knowledge matrix which the Gaussian elimination is performed on. This issue is a problem specific constraint which stems from the general complexity of row reduction operations and row swapping. Therefore, there are not any significant implementation changes that can be made to fix this time constraint besides resorting to using an entirely new strategy which does not involve Gaussian Elimination. This could however come at the cost of a new strategy potentially not being as effective as our current one at making advanced inferences by combining clues, which could overall reduce the success rate of defusing mines for the advanced agent.

5 Bonus

The original advanced agent we implemented used random selection for the next cell to be revealed if an inference was not able to be made. We can improve on the agent even further by using a more informed method for selecting the next cell to be revealed rather than simply picking a random one. This better decision making also takes into account the total number of mines on the board, and how many mines have been revealed so far. While less than 5% of the total mines on the board have been revealed, the algorithm sticks to just randomly selecting a cell to be revealed if no inferences could be made. This is because the chance of a hidden neighbor of a clue cell with even a value as low as 1 being a mine would be higher than a randomly selected cell being a mine. The main reason for this is due to the amount of mines left on the board and the high number of potential cells left to be chosen in the moves list. After some experimentation, we were able to decide that after 5% of the mines on the board are revealed, then it is more optimal to use a priority queue of cells that prioritizes cells with lower clue values, and randomly select from the hidden neighbors of the chosen cell. Choosing one of these hidden neighbors is less risky than randomly selecting any remaining cell on the board when the total amount of mines left to be revealed on the board is lower. The following graph compares the average success rates of the advanced agent and the better selection agent across various mine densities.



From the mine density interval of (0.2, 0.9) the better selection agent shows a significant performance increase over the advanced agent. In this interval the results are around 10 to 15% better for the better decision agent than the advanced agent. It seems that implementing a new selection mechanism greatly improves the performance of the agent.

6 Contributions

In this project we tried our best to divide up the workload evenly. In general, we discussed the algorithms and approaches we could take for the problem together. The actual implementation of the code and the writing of the report is what we divided. Ashwin did the majority of part 1 of the project which involved generating the board and making the basic agent which is able to make weak inferences to tell which cells on the board are mines. After this, we went over his implementation for part 1 of the project together so Ritin could get a better understanding of what Ashwin worked on. Then, Ritin did the majority of part 2 of the project, which was mainly implementing a way to combine multiple clues together to make more advanced inferences. This involved modeling the system of linear equations, writing the code for Gaussian Elimination, and writing the code for analyzing the reduced matrix to see what additional inferences could be made. For the report Ashwin did questions 4 and 5 and Ritin did questions 1, 2, 3, and 6. Afterwards, we discussed what to do for the bonus together and Ashwin worked on a solution for the bonus part of the project.

7 Honor Code

This assignment was done on our own. None of the code or the report was copied or taken from online sources or any other student's work.