AI Project 2

Arpan Gupta and Fawaz Tahir

March 2021

1 Questions and Writeup

1.1 Representation

Our program consisted of two boards that each contain a table of cells. Each cell contains a coordinate point and a state. The various states consist of a mine that has been revealed, which is represented by '-1', a non-mine cell that has been revealed, which is represented by an integer from 0 to 8, a closed cell, which is presented by the 'X', and a marked closed cell, which is represented by 'M'. The first board is an internal map that stores knowledge that can be used for inference. Primarily, the internal map shows how many mines surround each cell and which cells are mines. The second board is the board that the user sees. In the user board, each cell starts off in a closed state and when opened, it reveals the knowledge stored in that specific cell from the internal map. When looking at the inferred relationships between cells, we look at direct neighbors of opened cells. However, the user is only able to use the knowledge of direct neighbors that have already been revealed, safe or not. With this information, the cell that has been selected can make educated decisions and solve the program to the best of its ability. With the advanced agent, we decided to have the computer access more knowledge so that it could make even better decisions when solving the game. Rather than only looking at the current cell's direct neighbors, we had the computer look at the each of the neighbors' direct neighbors. This information allowed the user to make their next decision based off of a larger range of cells. Additionally, this allowed for a stronger inferred relationship between cells. After adding this information, we saw a dramatic increase in performance when compared to before adding this information.

1.2 Inference

We model the collection of clues using a combination of recursion and queues. Let's take a situation when we are about half way through a minesweeper game. The agent chooses a random cell. It then stores that random cell, and applies some basic inferences which check if all the neighbors are safe or if all the neighbors are bombs. If we can figure out that all the neighbors are safe, we can reveal them to the agent. The agent will then begin to check each of these safe neighbors. The safe neighbors are stored in a queue, and we made a recursive call to a method that checks if any new inferences can be made on these neighbors. Since this method is recursive, it will keep checking neighbors, and neighbors of neighbors, until it finally can not make any inferences. At this point, it will pick a random point and execute the method for that point. The goal of using this method is to try and ensure that the agent deduces everything before it resorts to picking a random point. According to our traces, we believe that this is done correctly, and the agent is able to deduce all information regarding the entire maze, not just the one clue.

Now that we have discussed how the program runs, we can discuss what inferences each execution checks for before it moves on. The first thing we want to check is if all the neighbors of a cell ares mines. This would occur when the condition:

clue value - count of neighboring mines = number of hidden neighbors

This logically makes sense because it is saying if I have a clue of 4 and have 2 neighboring mines, and 8 total neighbors with 6 being open, we know that the two hidden neighbors are both mines. In this instance, we would mark all the neighbors as mines, and then, recursively call and check if any of the neighbors of the

mines can be inferences. This proved to be very useful since in the cases of a neighboring one to a mine, we were able to open all the neighbors and continue checking each neighbor from there.

The next inference that we checked was if all the neighbors are safe, meaning we can open them and use their clue values to help us deduce more of the map. This would occur when:

number of total neighbors - clue value - number of safe neighbors = number of hidden neighbors

This would mean that all the neighbors are safe and none of them are mines. Our program will open all the neighbors, add them to a queue, and then recursively call the method to check for inferences on each of these neighbors. Using these two main inferences, combined with checking neighbors of neighbors and constantly calling recursively to ensure that after once clue is updated, we can make sure that all related clues are accounted for, we were able to make our agent "smart". This was very indicative in the performance of the agent.

1.3 Decisions

The improved agent goes through a variety of steps when deciding which cell to search next. First, it takes a look at what information the current cell has and it determines its next step depending on what that information reveals. If the current cell is a mine, the improved agent takes a look at its direct neighbors that have already been revealed. The reason it takes a look at these neighbors is because the knowledge base has increased and those neighbors can now make new educated decisions that it may not have been able to before. It then makes those decisions depending on if certain conditions were met. If none of the neighbors were able to use that information to make a new decision, then it picks a new cell at random.

Another situation that could happen is if the current cell contains a positive integer. Here there could be 3 situations. Either all of its hidden neighbors can be marked as mines, all of its hidden neighbors can be marked as safe, or no inference can be made. If no inference can be made, it would follow the same process as it would have if the current cell contained a mine. Looking at the first of these three scenarios, the program would be able to determine if the cell's hidden neighbors can be marked as mines if the total number of mines surrounding that cell minus the number of revealed mines surrounding that cell is the total number of hidden neighbors. All of this information is given when a cell is revealed and contains a positive integer. Once determined that all of its neighbors can be marked as mines, the program then checks the neighbors of the mines. This creates a sort of modified Beam Search algorithm. The program then re-evaluates the neighbors of the cells that have just been marked. The reason for this is because the the neighbors of the marked cells now have new information that they can use to re-evaluate and make a new educated decision. All of these neighbors get stored into a queue and can potentially set off a chain reaction of new informed decisions. It is inserted into the queue in no specific order since the order should not have an effect on the results of the outcome. However, a specific queue order could improve time and space complexity and if we had to rewrite our program to be improved, this would be a good place to start. This recursive logic allows the program to possibly completely uncover the board efficiently and effectively. Another scenario would that the program could determine if the current cell's hidden neighbors can all be marked as safe if the total number of safe neighbors minus the number of revealed safe neighbors is the number of hidden neighbors. Once again, all of this information is revealed when the current cell is uncovered. Once all of the neighbors are marked safe, they are inserted into a queue. In this queue, the safe neighbors are revealed one by one and its neighbors then go through the evaluation process. If the second-order neighbors can not make an inference, then the program resorts to the worst case situation which is selecting a cell at random.

The program makes sure to emphasize re-evaluation at every step so that the improved agent can make more informed decisions at every step. The basic agent only looks at the information given directly to it from the current cell. The improved agent, rather, looks at a lot more information. Not only does it look at the information given directly to it from the current cell, it also takes a look at information that was revealed in the past by looking at the current cell's neighbors and looking at the those neighbors' neighbors. This increases the amount of information when making a decision dramatically and increases performance when compared to the basic agent.

1.4 Performance

We can trace a 4x4 map with 5 mines inside to help us see how the advanced agent progresses through the map
The map:

-1	1	0	0
1	1	1	1
1	2	4	-1
1	-1	-1	-1

Agent Play by Play:

X	X	X	X
X	X	X	X
X	X	X	X
X	X	X	X

Initial Display of Map

X	X	X	X
X	X	X	X
X	X	4	X
X	X	X	X

1. Randomly picked 4 (row:3, col:3)

X	X	X	X
X	X	X	1
X	X	4	X
X	X	X	X

2. Randomly picked 1 (row:2, col:4) since no inference from 4 could be made. Checked 4 (row:3, col:3) since 4 is a neighbor of 1 (row:2, col:4).

X	X	X	X
1	X	X	1
X	X	4	X
X	X	X	X

3. Randomly picked 1 (row:2, col:1) since no inferences could be made.

X	X	X	X
1	X	X	1
X	X	4	X
X	X	-1	X

4. Randomly picked -1 (row:4, col:3) since no inferences could be made. Checked 4 (row:3, col:3) since 4 is a neighbor of -1 (row:4, col:3). No inferences could be made. Checked 1 (row:2, col:4) since 1 is a neighbor of 4 (row:3, col:3). No inferences could be made.

X	X	X	X
1	X	1	1
X	X	4	X
X	X	-1	X

5. Randomly picked 1 (row:2, col:3) since no inferences could be made. Checked 4 (row:3, col:3) since 4 is a neighbor of 1. No inferences could be made. Checked 1 (row:2, col:4) again since 1 is a neighbor of 1 (row:2, col:3). No inferences could be made.

X	X	X	X
1	X	1	1
X	2	4	X
X	X	-1	X

6. Randomly picked 2 (row:3, col:2) since no inferences could be made. Checks 4 (row:3, col:3) again since 4 is a neighbor of 2 (row:3, col:2). No inferences could be made. Checks 1 (row:2, col:3) since 1 is a neighbor of 2 (row:3, col:2). No inferences could be made. Checks 1 (row:2, col:4) since 1 is a neighbor of 1. Checks 1 (row:2, col:1) since 1 is a neighbor of 1 No inferences could be made.

X	X	X	0
1	X	1	1
X	2	4	X
X	X	-1	X

7. Randomly picked 0 (row:1, col:4) since no inferences could be made.

X	X	0	0
1	X	1	1
X	2	4	X
X	X	-1	X

8. Opened all neighbors of 0 (row:1, col:4). Checked 1 (row:2, col:3) since 1 is a neighbor of 0 (row:1, col:4). No inferences could be made. Checked 4 (row:3, col:3) since 4 is a neighbor of 1. No inferences could be made. Checked 2 (row:3, col:2) since 2 is a neighbor of 4 (row:3, col:3). No inferences could be made. Checked 1 (row:2, col:4) and marked row:3 col:4 based on inference.

X	X	0	0
1	X	1	1
X	2	4	M
X	X	-1	X

9. Checked all neighbors and neighbors of neighbors of M (row:3, col:4). (said this for simplicity since we explicitly defined neighbors and neighbors of neighbors for all the prior steps).

X	1	0	0
1	1	1	1
X	2	4	M
X	X	-1	X

10. Opened all neighbors of 0 (row:1, col: 3). Checks 1 (row:1, col:2) and marks row:1 col:1 as mine based on inference. Checks 1 (row:2, col:1) and opens all neighbors based on inference. Checks neighbors of 1 (row:2, col:1) and marks row:1 col:1 as Mine based on inference. Checks neighbors and neighbors of neighbors till it reaches 4 (row:3, col:3). Marks row:4 col:4 as Mine based on inference. Marks row:4 col:2 as Mine based on inference.

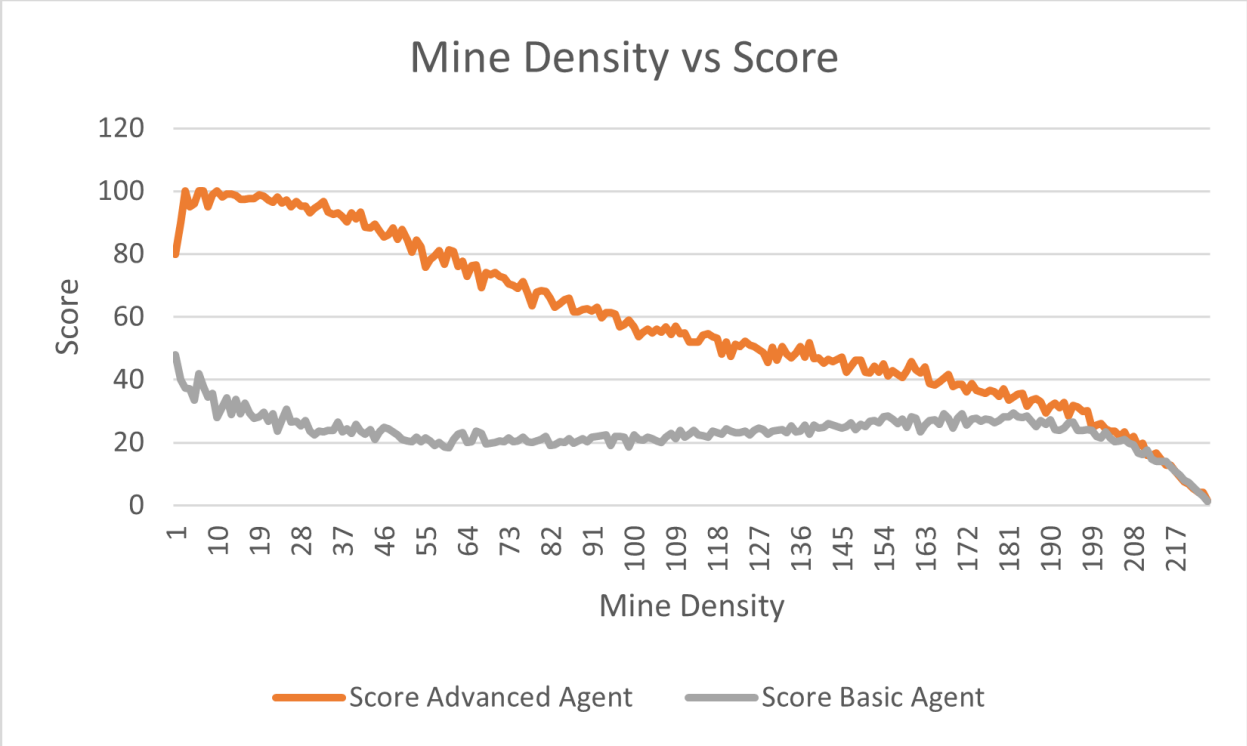
M	1	0	0
1	1	1	1
1	2	4	M
X	M	-1	M

11. Opens last remaining hidden square (row:4, col:1).

M	1	0	0
1	1	1	1
1	2	4	M
1	M	-1	M

FINAL SCORE: 4

Most of the moves made by the agent are the same moves that I as a human player would make. There were a couple of redundant moves that were made that could have been avoided, improving the performance of the AI. In all instances, we checked neighbors of neighbors. This would be extremely redundant because if a neighbor was unable to make any inferences, then the neighbors neighbor would also be unable to make any inferences since no new knowledge has entered the agent. Rather than always check the neighbor of a neighbor, we would want to make a conditional saying if any update was made to the map based on the neighbor, then only check neighbors of neighbors. This change would not have changed the actual score, it would solely have improved the efficiency of the AI. The score given to the agent would be the same score we as intelligent beings would have gotten.



Our logic and intuition agree with the patterns shown in the graph. As we can see, the advanced agent performs significantly better than the basic agent. This is because the improved agent has a better decision making process and thus, can make more informed inferences. With the improved agent, we see its performance gradually decrease as the mine density increases and then a sharp decline in performance when the mine density is extremely high. Additionally, the graph shows the basic agent staying relatively the same in terms of performance no matter the density size. We believed this would happen since the basic agent primarily makes decisions by randomly picking cells at nearly every step. The only time the basic agent changes in performance is at the very beginning where it did better with an extremely low mine density and at the end where it declined with an extremely high mine density.

The improved agent seems to drop below 80 percent performance at approximately 60 mines and then it hovers at above 50 percent performance at approximately 120 mines. Then we see the minesweeper become very difficult to solve at about 200 mines. At 200 mines, we see a sharp decline in both the basic and improved agent. The improved agent performs better than the basic agent at every mine density except for very high density values. At these points, we see the basic and improved agent perform similarly and the basic agent even performing slightly better at extremely high mine density points.

1.5 Efficiency

Our program is very inefficient in terms of space as well as time. We utilize recursion on lengthy queues on each iteration of the method. Using recursion helps us keep our code simple, and allows the agent to effectively traverse and solve the map, but it also causes the run time to be very slow. When we were running our basic agent on a map of size 15x15, it took mere minutes to run through 5,625 total iterations. On the other hand, for our advanced agent, it took closer to 30-45 minutes to execute the same 5,625 iterations. We can attest this to the fact that we check every neighbor, and its neighbor, for each execution of the advanced agent. Like mentioned earlier, this allows the advanced agent to make smart inferences, but it reduces our run time significantly. Same goes for space complexity. Recursion and Python do not go well together. When we tried running extremely large mazes, we hit max recursion depth many times. Furthermore, we make new queues for each set of neighbors, and each set of neighbors of those neighbors. For bigger mazes and smaller mazes alike, this uses up a lot of space in the memory and causes the program to be inefficient.

In order to alleviate some of these inefficiencies, we tried to implement a visited list. This list would store neighbors that have already been visited in one iteration of the method. We then passed this list back to the recursive method as a parameter to ensure that everything was only being checked once. We would reset the visited list before the agent moved on to another random cell. This helped us improve our run time because it no longer checks everything twice, as well as our space complexity because it does not add unnecessary cells to our queue. Even after improving the efficiencies with this, the difference was vast between the simple agent and the advanced agent.

I would say that a lot of these issues are implementation specific constraints. This ties back to our choice of using recursion over an iterative approach. This allows for simplicity in the code, as well as making sure the agent is deducing everything in the maze. If we wanted to change our implementation and hopefully improve our efficiency, I would thinking going with an iterative approach would help. That way we don't have to worry about creating new queues for every single cell, and don't have to worry about the run time issues that revolve around recursion. Furthermore, if we implemented our queues a little better, by maybe passing them as parameter and not creating new ones every time, we should have been able to save some space issues.

2 Bonus

2.1 Clever Acronym

For our improved agent we nicknamed it Agent 007 for its absolutely incredible performance and for the basic agent we nicknamed Agent 003.5 because it is about half as good as the improved agent. :)

2.2 Global Information

Suppose you were told in advance how many mines are on the board. Include this in your knowledge base. How did you model this?

While we did not get the chance to implement this, we did come up with an idea that should increase the performance of the improved agent if given the number of mines on the board in advance. First, we want to mention that if given the number of mines in advance, we can implement an early termination program that could increase space and time complexity by a large margin. It can even increase performance on certain occasions. First we would check to see if the total number of mines is equal to the number of mines that are marked and the number of mines that are revealed. If so, then we can terminate the program even if certain cells have not been revealed yet. Another situation we can use this extra information is in a unique situation where there is a cell that contains a certain number of mines surrounding it and none have been marked or revealed. If the total number of mines remaining is equal to this number, then we can mark all cells that are not neighbors as safe. With the safe cells we can then likely mark the mines accurately and subsequently, perform better than both of the other agents.

2.3 Better Decisions

While we did not get the chance to implement this, we did come up with an idea for creating a better decision making process than just going randomly. We would use probability to determine whether we should either select a random cell from the entire board or select a random cell that is a neighbor of the current cell. We can calculate these probabilities if given the total number of mines in advance. This information would allow us to calculate the probability of selecting a mine at random and then the information stored in the current cell will give us the probability we select a mine at random when only choosing amongst the neighbors.

2.4 Statement and Contributions

Arpan:

Statement: "All of my work is my own and I did not copy or take from online or any other student's work."

Contributions

1. Worked on Improved Agent
2. Assisted on Basic Agent
3. Answered Inference, Performance, and Efficiency Questions
4. Assisted on writing report and everything Fawaz did

Fawaz:

Statement: "All of my work is my own and I did not copy or take from online or any other student's work."

Contributions

1. Worked on Basic Agent
2. Assisted on Improved Agent
3. Answered Representation, Decisions, and Performance Questions
4. Assisted on writing report and everything Arpan did