Rutgers University

Artificial Intelligence
01:198:440:01

Assignment: Minesweeper

Authors:
Azim Khan
Aditya Anikode
Vrinda Jain
Net ids (auk11,aaa387,vj140)

March 23, 2021

RUTGERS
THE STATE UNIVERSITY
OF NEW JERSEY

1 Abstract

To win the game Minesweeper, it is important to accurately choose moves based on data collection and inferences. In this report we explore the challenges of creating a minesweeper solving algorithm using different strategies and methods like Logic and Satisfiability, Constraint Satisfaction, and Search. The aim of this project was to learn how data collection and inference can impact future data collection. We built this project by dividing it into 6 classes: an environment class which consisted of generating the board and adding mines; the basic agent class which chooses cells based on the clues picked up by previous cells; the improved agent which implements additional strategies to win the game; the clue which incorporates all the conditions for both agents; the Minesweeper Gameplay which runs the basic agent board; and the Improved Agent Gameplay that runs the improved agent board. By incorporating inferences in the improved agent class, the likelihood of winning the game increases by about 16.3 %. The improved agent works the best at the midpoint mine density as it allows the agent to flag more mines compared to the basic agent.

2 Introduction

Minesweeper is a single player game that contains a board with some mines. The goal of this game is to determine where all the mines are located without triggering the bombs by utilizing clues from neighboring cells. The first cell the player selects is random and it will either reveal a number which represents the amount of neighbors that have mines or it will be the mine itself. Usually, the game ends when the mine cell is selected, however in our version, the player can continue to predict where the mines will be until all cells are uncovered, marked as clear, or marked as mined. The player can choose the number of mines they want to find; the higher the number, the more difficult the game becomes. Instead of there being a player selecting a cell every round, we generated an agent which selects the cells based on its knowledge base. Ideally, this agent should play better than a human player as it is storing all the information and making inferences. We designed the algorithm to make better predictions on where the mines will be utilizing constraint satisfactions.

3 Project Approach

Before we started working on the code, we familiarized ourselves with the game of minesweeper by playing a couple of games online and also looked up different tips and tricks for the game which could be implemented into our code. Next, we drew up a general rule for our game/environment and what we wanted to include in the game. Based on our environment we explored how we would implement our two different agents into the environment. Later we wanted to find ways to implement the different tips and tricks we found online into our game and one of the first tricks that we found was the idea of starting at the corner because in the later moves the corners provide the least amount of information so it is better to eliminate some corners from the start, so we do not have to guess for the next move. However, we found out that is only a viable strategy when the first move of the game has no penalty (which is a rule in some implementations of the game) and since we decided that we were not going to make our environment that way, we decided not to explore that idea.

4 Code Explanation

4.1 Environment Class

The environment class is the base of our project where the board is generated given a dimension and number of mines. The init function initializes the board's characteristics: height, width, and number of mines. These can be accessed from other functions within the Environment class. We initialize an empty board with no mines, and then we randomly place the mines. We also initialize a set with no mines (we will later add all the discovered mines in this set). The "is mine" function takes in the cell we are on and returns the space in the board with that cell location. The "mineNeighbor" function returns the number of mines that are within one row and column of a given cell, not including the cell itself. The "won" function checks if all mines have been flagged.

4.2 BasicAgent Class

This basic knowledge-based agent makes decisions by considering its knowledge base, and makes moves based on that knowledge. This agent maintains knowledge containing new information gained from querying the

environment and inferring that data to generate new information. We must identify the number of safe squares, mines, and hidden cells around each cell. In the init function, we initialized a set called "mineSet" and "safeSet" to keep track of cells known to be safe or mines. To keep track of all the cells that have been clicked on, we initialize a set called "track moves" and "track moves.add(cell)" which adds all locations (x,y) into all cells. We also initialized a list called "knowledgeBase" which keeps track of all the clues about the game known to be true. The "MarkMine" marks a cell as a mine, and updates all knowledge to mark that cell as a mine as well. The "MarkSafe" marks a cell as safe, and updates all knowledge to mark that cell as safe as well.

4.3 Improved Agent Class AKA MC SKIPA (MineCweeper Smart Knowledge Inference Prediction Agent)

Figure 1: MC SKIPA

We have split up our knowledge base on the terminal based on the move the agent made, then the AI's knowledge base adds a set containing the confirmed cells and another set containing the confirmed mine. For example, in our code, we start with making a random move, which we always store in a set, since we currently have no knowledge base and no confirmed safe move to make. After the first move is made and we get our first count, the number of mines surrounding the cell, depending on the count, our knowledge base is updated and similarly our confirmed safe and mine set is also updated accordingly. So, if the count shows a 0 we insert all of that cell's neighbors to the "confirmed safe" set. We make our future moves based on the "confirmed safe" set, whenever the confirmed safe set is empty we are forced to make a random move. Up until now the improved agent is similar to the basic agent however now we have introduced the inference algorithm into the improved agent that uses the power of inference/deduction to extend its knowledge base to make better decisions. We have implemented an algorithm that looks at each "sentence",a set of cells that equal to a value greater than 0 so any probed cells that are not free, in our knowledge base and compares it to one another and see if there are any subsets. Now, we can make our new sentence that contains unique cells from the previous two sentences compared and the value related to that sentence is the difference between the counts of the compared sentences. With this operation we are able to expand our knowledge by shortening the number of cells in a sentence which allows us to make better moves and in some instances we are able to herd out the mine and find new safe cells. This process simplifies our mine finding algorithm and makes it accurate by 16.7

4.3.1 Simplify Knowledge()

In order to shorten the clue for the AI's knowledge base we decided to also include a function which checks if there are any redundancies as if a confirmed safe cell or confirmed mine cell or moves made cell is part of the clue. By implementing this function we were able to make the AI's knowledge base more precise and improved its decision making. While we have not tested it before and after we have implemented it into our basic agent and improved agent for better decision making.

4.4 Clue Class

There are a few conditions we must take account of in the Basic Agent, but we made a separate Clue class for that. The first one being: "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" [1]. Our MinesKnown function is responsible for that condition as it equates the length of the cells with the number of mines; it returns the set of all cells in self.cells known to be mines. The second condition we must look at is "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"[1]. The MinesSafe function suffices that condition by returning the set of all cells in self.cells known to be safe. If a cell is known to be a mine, the MarkMine removes it from the set of all board cells, and decrements count by 1. If a cell is known to be safe, the MarkSafe removes it from the set of all board cells. This class will be called in the Basic and Improved Agent to simplify the mine identification process. [1]

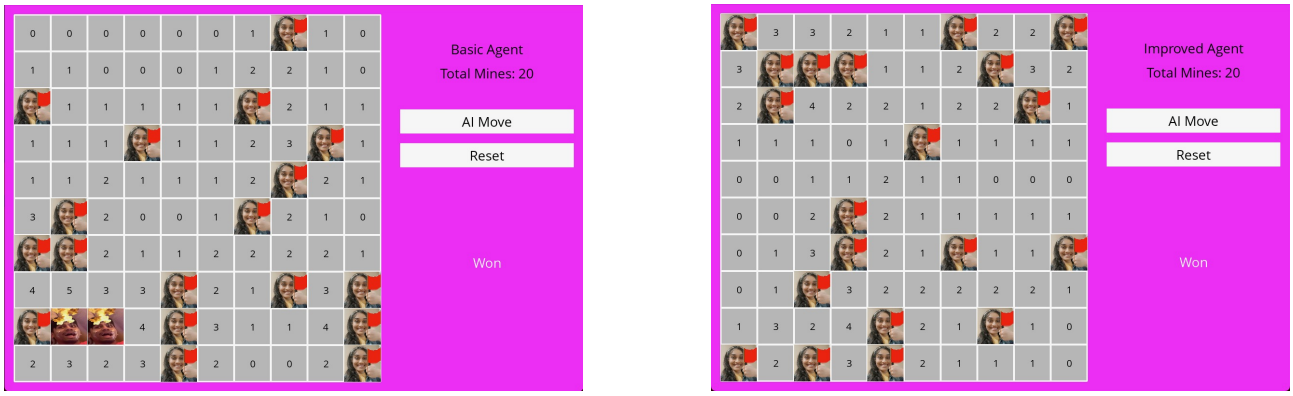

Figure 2: Mine Density at 20 % in 10x10 board with Basic and Improved Agent. The Improved agent flags all of the mines while the Basic runs into 2 mines.

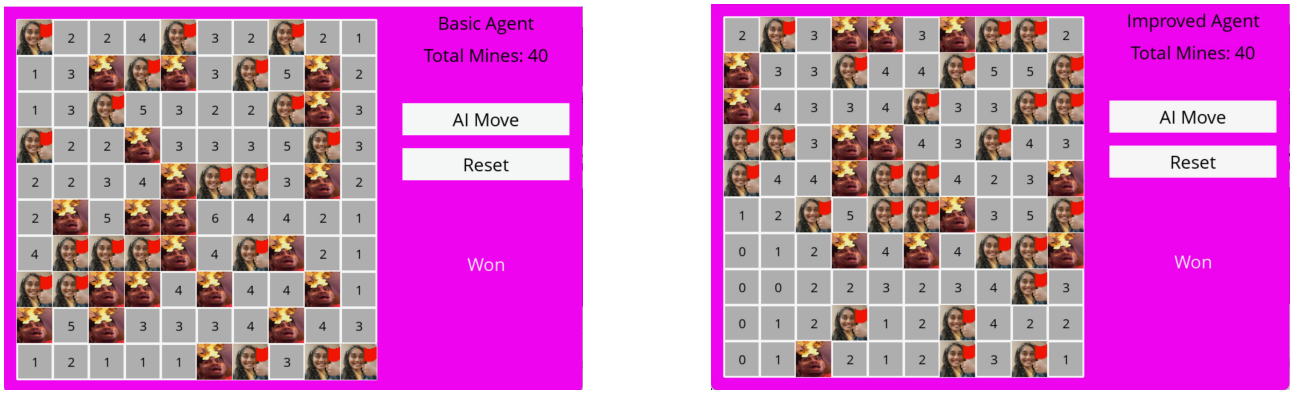

Figure 3: Mine Density at 40 % in 10x10 board with Basic and Improved Agent. The Improved agent flags 26/40 mines while the Basic flags 20/40 mines.

4.5 MinesweeperGameplay Class

This is the runner for the basic agent and incorporates the GUI. We utilized the GUI from the Harvard Computer Science department [2].

4.6 ImprovedAgentGameplay Class

This is the runner for the improved agent and incorporates the GUI. We utilized the GUI from the Harvard Computer Science department [2].

Representation

The board was represented as a two dimensional matrix with rows and columns, or a nested list. The user has the ability to provide an input on this, as well as set the mine density to change the difficulty of the board. There are two sets we use to keep track of the mines. The mines found is a set which contains locations (row,column) that updates as the player finds more mines while the set mines represents the total number of mines within the board at randomly placed board locations. All these aspects were covered in the environment class. The GUI was created in our MinesweeperGamePlay and ImprovedAgentGameplay classes using the Harvard code as a reference [2]. However we adjusted it in both of the GamePlay classes so that whenever the agent lands on the mine, it displays Azim's face with an explosion and Vrinda's face with a red flag whenever the agent predicts a mine cell. In our code, we have added different print statements which helps us keep track of the knowledge the AI has at all times. In the terminal every time the AI makes a move, we add it to a set which holds all of the moves it makes and displays the current move made as "move". Additionally, we print out a "knowledge base" which holds all of the clues the AI has gained throughout the game. Lastly, we show two other sets "Confirmed Safe" and "Confirmed mine" which hold a set of cells that the AI has deemed safe and will use for future moves and a set of cells that the AI has deemed to be a mine and will flag respectively. The AI uses this information to infer more information for the future.

Inference

As explained above on our improved agent code explanation, we collect our clues from probing a cell and gaining a count, number of mines surrounding the cell. From this information we create a set or "sentence"/ clue that consists of all of the unprobed and unflagged neighbors surrounding the probed cell and make it equal to the count that we found. This sentence is then added to the knowledge base of the AI. As the AI progresses through the game and gains knowledge, it compares its own knowledge base and uses inference operations to make its knowledge base more accurate and precise for the next move. With these new information/knowledge through inference, the AI is able to add new cells to "confirmed safe" or "confirmed mine" based on its deduction and eventually win the game. Sometimes inference can not be used due to the lack of clue and our program attempts to deduce everything it can from a given clue however a fresh clue might not have any information, so the program moves onto its confirmed safe set from which it selects its next move. However, as the number of clues increases, the program is able to gain more information and deduce everything it can from the clues it has available. One of the biggest flaws in our program is the random move the agent has to make when it has either run out of clues or can not infer any new knowledge that could make a cell safe to explore. Currently in our code whenever the agent is stuck, it chooses randomly from the set of free cells that it has not been flagged nor explored already. This is quite a dumb approach and One way this could be improved is by computing the probability of a cell containing a mine for each cell by using bayesian network and picking from the least probable cell each time a random move needs to be made.

Decisions

Our program will always make its next move based on the "confirmed safe" set and if that set is empty, the program will move on to making a random choice. Keep in mind that the "confirmed safe" set is updated every time a move is made, either based on the count it gets from the cell or use of inference on its knowledge base, so while it may sound quite simple, the decision making behind the AI's next move is thought out.

Performance

When running our 10x10 board with 40 % Mine Density, there were times when the improved agent had the same performance compared to the basic agent. This was not expected because we ran multiple test cases where that did not occur. However, when we increased the board size, the Improved agent once again performed better than the basic. We increased the board size to a 50x50 dimension with a 100 mines and we still won the game with just the basic agent and found all the mines. This was shocking because even with such a large dimension our algorithm worked.

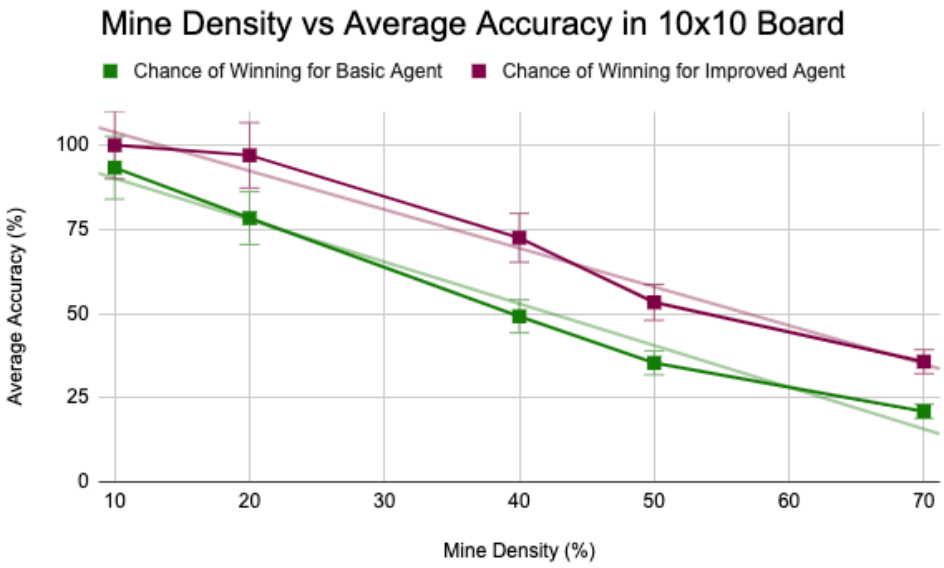

Figure 4: Mine density vs Accuracy of finding Mines for Improved Agent (purple) and Basic Agent (green)

Our graphs agree with our intuition as the improved agent provides us with better accuracy compared to the basic agent. We noticed as the mine density increased the accuracy of identifying the mines started to drop down. Our improved agent is 16.7 % better than the basic agent overall, however we noticed a significant increase in accuracy during the midpoint mine density. In a 10x10 board, when the mine density is at 40 %, there is 73 % accuracy of winning in the improved agent but only 49 % in the basic agent.

Efficiency

We noticed that as the number of dimensions increases, the agent takes longer to finish the game. In a 30x30 dimension with a mine density 50, the game takes 7 minutes to complete for the improved agent. This could be due to the PC we used to run the program and the time complexity of n^2. It was n^2 because we implemented several nested loops, worsening the time complexity. The max dimension we were able to play with was with a 50x50 dimension, however the cells were too small and it was difficult to see all the faces. Additionally, it took around 20 minutes to finish playing which means our algorithm works faster for smaller dimensions. These were implementation specific constraints because when making the GUI, we had made the window size small, if we had made it bigger, we could've played with higher dimensions. In the future, we can simplify the way we write our algorithm so it uses fewer nested loops. When researching about this project, we noticed that some people made equations in one line which took us multiple nested loops to perform. By condensing the information in one line and fewer loops, we can potentially improve the time and space complexity of the game.

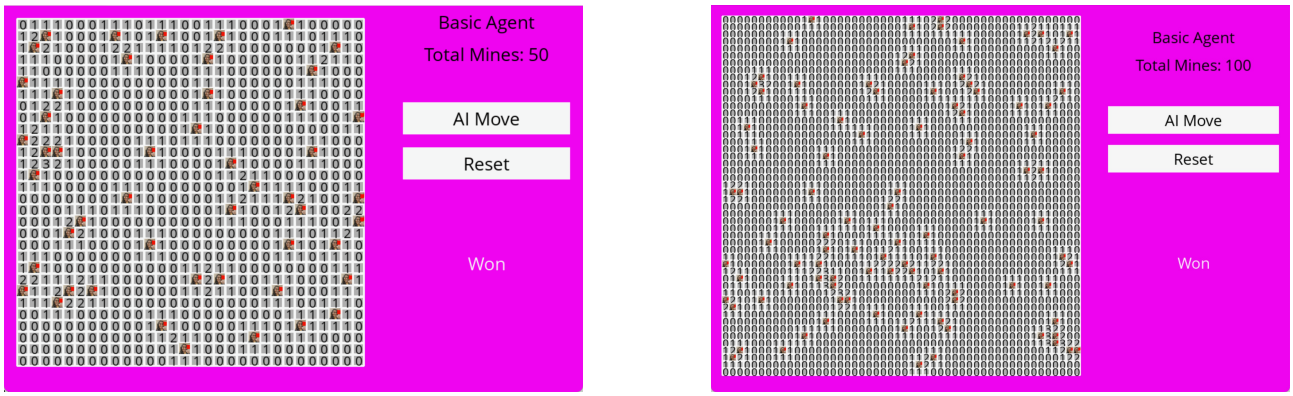

Figure 5: Left: Mine Density at 5.6 % in 30x30 board with Basic Agent. The time it took to complete was 7 minutes Right: Mine Density at 4 % in a 50x50 board with Basic Agent. Time it took to complete was 25 minutes

Global Information

We incorporate how many mines are on the board within the knowledge base by comparing the set of mine cells that the agent has identified by calling the neighboringCells function to the master set of mines, which is initialized in the environment class. The agent uses this information to essentially "win" the game when all the mines have been identified rather than when all the cells on the board have been uncovered, terminating the game early and conserving time and space.

Conclusion

Building the game Minesweeper is a great way to learn how to efficiently use a knowledge base and inference techniques to make smarter decisions. Knowledge Based Systems is a major area of artificial intelligence and makes decisions based on the data and information that resides in the network [3]. This system helps improve decision making and is useful when information must be stored effectively for future use. It also allows us to integrate knowledge on a large scale; in our case it would be the 50x50 dimension board. It is capable of generating new knowledge by using the stored data and making tasks much more efficient and less time consuming. In Minesweeper, the knowledge base consisted of all the safe cells and mines discovered. By using this data, our agents were able to flag potential mines to prevent clicking on them. Since it made predictions, it lowered the chance of the agents from stepping on the actual mine, thus helping them win the game. We had

created two agents, one with a basic knowledge base and the other with a knowledge base that uses inference to constantly update itself based on overlapping clues and compared the accuracy for both. Our algorithm was not perfect and there are still several improvements to make. Due to the time complexity of n^2, it could not finish a game without taking 20+ minutes whenever we had any board about a 50x50 dimension. To improve this, we will work on simplifying our code and utilizing fewer nested loops. Overall, this project improved our understanding of Knowledge Based Systems, Inference Techniques, Logic and Satisfiability, Constraint Satisfaction, and Search.

Contribution statements

In order to divide the work fairly, we decided to make a server on discord where we would meet regularly and work on the project together. We started off by creating pseudo code for each search algorithm and implementing them as we went. Our group as not as proficient as the project needed us to be, so we spent a lot of time learning things on our own in order to write our code.Overall, I believe everyone worked hard and fairly on this project. - Azim Khan

Our group initially struggled with the coding logic for our project. This is because we are all novice programmers and did not have much exposure to Python prior to this project. Therefore, we initially started by making pseudocode for each method and algorithm we were tasked with creating. We spent a lot of time individually learning through youtube tutorials and reading online materials regarding the topics of interest. After individually building our skills, we came together and communicated via discord and messenger video call to bring our ideas to the table. We met regularly, pretty much every day, and diligently worked on the project. Although we were novice coders and struggled initially with the project, we were able to piece it together little by little to form a finished final project.- Aditya Anikode

This project was a challenge. My group met regularly via discord and facebook messenger and collaborated fairly and evenly on this project. We initially struggled with writing the code, so we started by collectively writing pseudo code to better understand the task at hand. Moreover, we spent a lot of time individually learning the concepts and ideas behind each search algorithm, so we could make progress when we met. Overall, I believe everyone put a great deal of effort into this project and the work was divided equally. Although we struggled, a great deal of information was learned. - Vrinda Jain

Honor Code

We have read and abided by the rules of the project, we have not used anyone else's work for the project, and we did all of the work by ourselves.

[4]
[1]
[3]
[2]

References

[1] Wesley Cowan. Assignment 2- inference-informed action, 2021.

[3]Chase Labrador. Knowledge-Based System, url=https://www.kmslh.com/glossary/knowledge-based-system/: :text=A knowledge-based system, Oct 2019.

[4] Wiki. Strategy, url=http://www.minesweeper.info/wiki/Strategy, 2014.

[2] Brian Yu. Minesweeper - CS50's Introduction to Artificial Intelligence with Python, url = https://cs50.harvard.edu/ai/2020/projects/1/minesweeper/?fbclid=IwAR2yRjo6KmYtAt9AN2S8xzBeu2LzmqahzPNC9vjEKCw5F5vxS1R1cQojdUI, 2018.