Minesweeper Solving AI Agent

Rishabh Kala, Jay Joshi, Rohan Mathew, Cole Sanders
rkala@ncsu.edu, jmjoshi@ncsu.edu, rjmathe2@ncsu.edu, cgsande2@ncsu.edu
Department of Computer Science
North Carolina State University
Raleigh, North Carolina, USA

I. Abstract

Minesweeper is a classic game where a player must choose correctly select tiles from an unmarked grid while steering clear of randomly hidden mines within the grid. We undertook the task of creating an AI based agent which can play Minesweeper and do it well. Minesweeper is a challenging game with the difficulty increasing exponentially with an increase in the size of the grid. The performance of these models can provide us valuable insights into AI problem-solving capabilities and the results can be used to extend this work to other real-world problems that require similar deductive reasoning. These methodologies have the capabilities of leaving a significant positive impact. In this report, we have summarized our findings.

II. Introduction

The objective of Minesweeper is simple, where we need to simply identify the safe sections of the grid without triggering any mines in the process. As we increase the difficulty, to successfully play this game, a player needs strong knowledge of logic, probability and strategy. An AI agent with knowledge of these concepts can be used to efficiently play this game and the results can be extended to genuine real-world problems. We built different agents and have compared their performance against one another:

1. A Knowledge-Based CSP Agent
2. Stochastic/Probabilistic Agent
3. Reinforcement Learning Based Agent

Below, we explain our methodologies and the results we got from each of these models.

III. Interface Generation

The UI was constructed using the Python library tkinter. The documentation can be found here [5]. There are twelve types of tiles in the Minesweeper game: number tiles (from 1 to 8), a ”mine” tile, a ”blank opened” tile, an ”unopened” tile, and a ”flag” tile. Images were found for all twelve types. The UI is constructed to display the correct corresponding image at each grid location to represent the state of the board. A thirteenth tile, a ”smiley face” tile is shown over the mine locations if the player wins.

Fig. 1. This figure shows the image tiles laid out in a grid format to represent the board state.

IV. Methodology

A. *Knowledge-Based CSP Agent*

The knowledge-based inference logic uses a matrix called $known$ which keeps track of all the tiles that are certain mines and certain safe tiles.

A matrix called $state$ keeps track of all the tiles that have been opened so far. The unopened tiles are represented by a special character in the matrix.

The following logic is used to find tiles that are safe and tiles that are mines for certain:

1) If an opened tile has a number x (indicating the presence of x neighbors that are mines) and only has x neighbors that are not opened yet, we can infer that all of these unopened neighbors surrounding this opened tile are mines. These tiles are flagged in the $known$ matrix. This inference method is demonstrated in Figure 2.
2) If an opened tile has a number x (indicating the presence of x neighbors that are mines) and we observe from the 'known' matrix that x of its unopened neighbors have been flagged for mines, then we can infer that all of its unflagged and unopened neighbors are safe tiles. This inference method is demonstrated in Figure 3.
3) In order to simplify the inference process, we subtract the numbers at each of the opened tiles of $state$ by the number of known unopened neighbors. After this simplification, the numbers in the $state$ matrix represent the number of neighboring tiles that have not been flagged as mines instead of the total number of neighboring mines.
4) If the agent cannot find a certain safe tile using these rules at any given step of playing the game, it picks an unopened tile randomly.

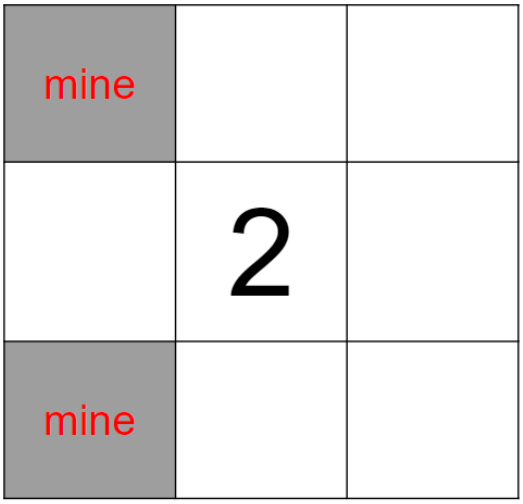

Fig. 2. This figure demonstrates how mines are found using the number of unopened tiles in the neighborhood

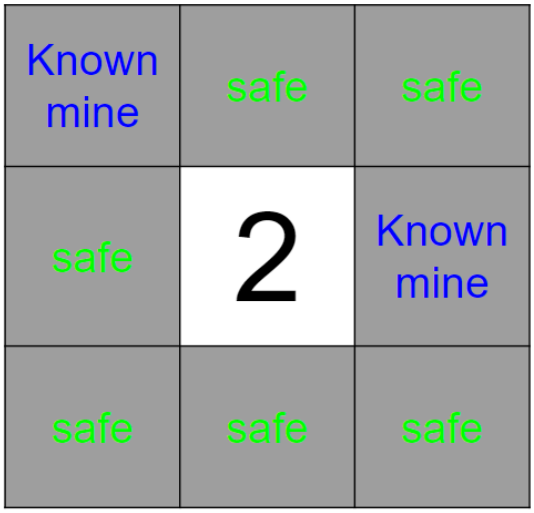

Fig. 3. This figure demonstrates how safe tiles are found when known mines are present in the neighborhood

B. Stochastic/Probabilistic Agent

In our project, the stochastic approach only takes effect when the knowledge-based system cannot find a safe tile to pick next using its rules. In such a scenario, the agent selects the tile with the least "likelihood" (or probability) of being a mine. The likelihoods are calculated in the following manner:

1) If an opened tile has a number x and it has k unopened neighbors, then each of the unopened neighbors has a probability of x/k of being a mine (this is assuming that none of the neighbors is adjacent to any other open tile). Let us call this probability p. The opened tile assigns a likelihood score of p to each of its unopened neighbors. An example of this is shown in Figure 4. In this figure, an opened tile with the number 3 has 8 unopened tiles surrounding it due to which each of them is given a score of $3/8$.
2) If an unopened tile has multiple opened neighbors, then each of its opened neighbors assigns a likelihood score to it. The effective likelihood score at such a node is the sum of these individual likelihood scores assigned by the opened neighbors. An example of this is

shown in Figure 5. The opened tile numbered 2 assigns a likelihood score of $2/5$ to each of its neighbors (because it has the number 2 and there are 5 unopened neighbors). Similarly, the opened tile labels 3 assigns a likelihood score of $3/5$ to each of its unopened neighbors. Since the 3 tiles in the middle column are adjacent to both the opened tiles, its effective likelihood score is the sum of the individual likelihood scores $2/5 + 3/5 = 1$. The likelihood scores here are not equivalent to probability, i.e., a likelihood score of 1 does not imply that there is a probability of 1 that a tile is a mine.

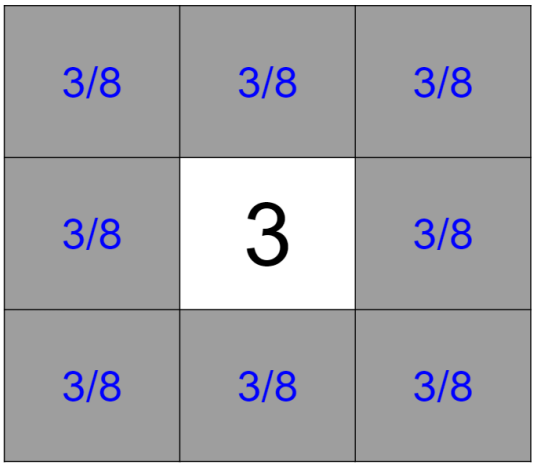

Fig. 4. This figure demonstrates how the likelihood scores for a tile being a mine are assigned to unopened tiles.

If the knowledge-based system does not find a safe tile, we select the tile with the least likelihood score for our next move. The tile with the least likelihood score is the least likely to be a mine. Please note that the likelihood score does not give the exact probability. It could have any value between 0 and 8 since it is the sum of the probabilities assigned to it by each of its neighbors.

C. *Reinforcement Learning Based Agent*

We implement a double deep Q-network for our reinforcement learning based approach.

Model Architecture:

As this is a DDQN, both the target network and online network share the same architecture. The input size is equal to the state size, which in the case of training was 36 since the training board is

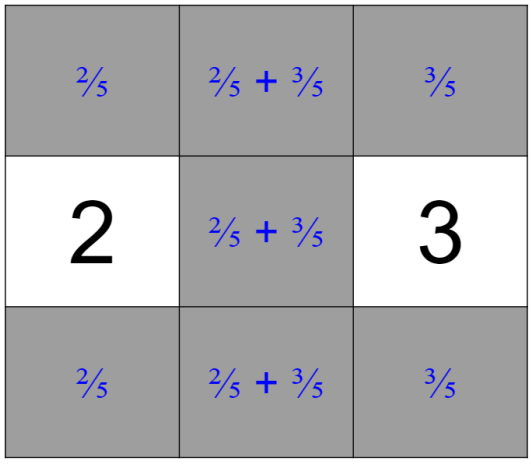

Fig. 5. This figure demonstrates how the likelihood scores for unopened tiles are aggregated from the individual likelihood scores assigned to them by their opened neighbors.

6 by 6 and there are 36 cells. Similarly, output size is also 36. There are two hidden layers with 128 neurons each. After the feature extraction layer, the network branches into two:

i Advantage: Which estimates the advantage of each possible action.
ii Value: Estimates overall value

The forward pass has the following steps:

i The input state is normalized by dividing by 8
ii A safe softmax function is used. This softmax ensures that the values which cannot be taken are masked.
iii The advantage and value functions are computed using the safe softmax function. The final Q-values are calculated by combining the value and advantage sub-networks.

The model selects an action either by choosing the highest Q-value or a random valid action to promote exploration. A replay buffer is required for DDQNs to break up the correlation the model may learn between each subsequent time step. We set it as 100000 and batch size as 4096.

Training process:

1) Initially, a training loop is set. This loop runs for 20000 epochs. Also, we use the epsilon-greedy strategy for choosing actions.
2) When an action is executed, agent receives next state, reward, and whether the game has

reached a terminal state.

3) Each transition is stored in the buffer.
4) After each addition of 4096 episodes, a batch is sampled from the buffer.
5) For each sample in batch, online model predicts the Q-value. Target Q-values are predicted by target network and temporal difference loss is calculated using the Bellman equation.
6) Online model's weight are updated for each such sample. However, target model's weights are updated after certain periods.
7) Actions may also be selected randomly. This randomness is reduced over time and reliance on model is increased.

V. Results

For the first two agents (i.e. the knowledge-based and stochastic agents) we compared their performance based on their win rate percentage as shown in figures 6 and 10, average board completion percentage as shown in figure 9 and average clicks per game as shown in figure 8. These metrics were calculated for each model for 500 games, over varying grid sizes and number of mines - 6x6 grid with 6 mines, 8x8 grid with 10 mines, 16x16 grid with 40 mines and 30x16 grid with 99 mines. These parameters need to be varied to correctly assess the agents as increasing the size of the grid increases the difficulty which must be taken into account. Moreover, increasing the number of mines, increases the difficulty and the probability to choose a mine but on the other hand, this increase also gives us better equations for the knowledge base, as we have better constraints with more absolute equations. Following are the metrics:

TABLE I
KNOWLEDGE BASED CSP AGENT

Size	Mines	Win%	Board%	Clicks/Game
6x6	6	30.00	58.08	6.70
8x8	10	34.40	54.13	11.60
16x16	40	27.00	54.44	46.36
30x16	99	0.40	17.95	34.13

Graphical representation of the results shows comparision of the Knowledge Based Agent with the Stochastic Agent (KB Agent is represented in orange and Stochastic Agent in orange)

TABLE II
STOCHASTIC/PROBABILISTIC AGENT

Size	Mines	Win%	Board%	Clicks/Game
6x6	6	37.80	58.45	7.77
8x8	10	43.80	60.24	13.30
16x16	40	34.00	51.18	50.16
30x16	99	1.60	22.85	47.39

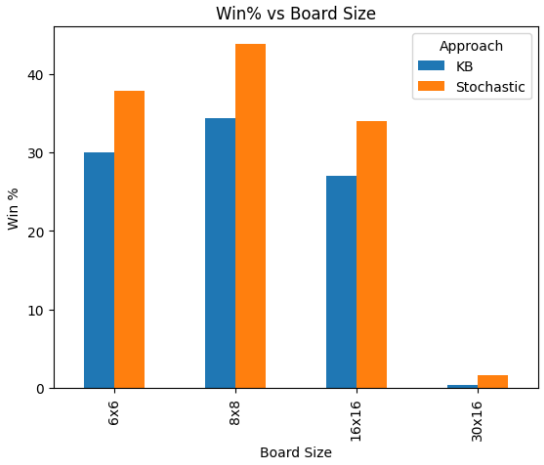

Fig. 6. Bar-graph depicting Win% vs Board Size

From the above graphs, we can clearly see that the Stochastic agent performs better than the Knowledge Base (CSP) agent where it consistently shows a better Win percentage for every board size and number of mines, with the best Win Percentage of the Stochastic Agent being 43.80% at the 8x8 board with 10 mines. We can also observe that initially the win percentage increases but then it decreases and this shows that the number of increased mines do provide better metrics for the agent to find probabilities and establish constraints but the factor of increasing grid affects the difficulty more. Though the performance of the Stochastic agent proved to be better, it can still be improved and for that purpose, we perform reinforcement learning and compare its results.

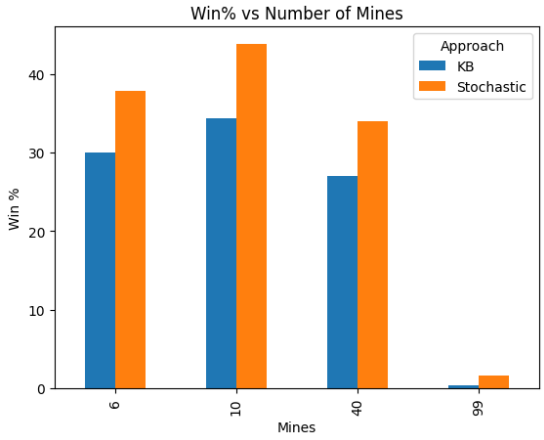

Fig. 7. Bar-graph depicting Win% vs Number of Mines

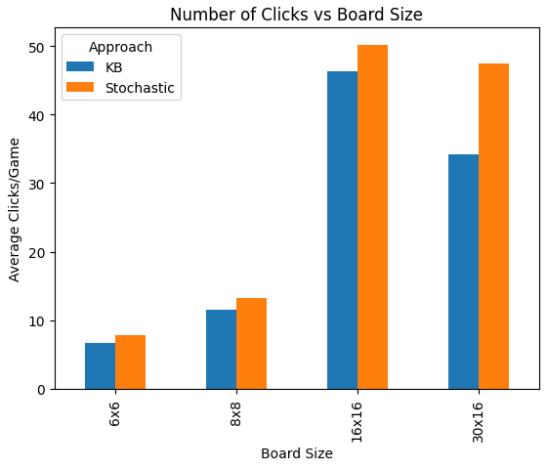

Fig. 8. Bar-graph depicting Average Clicks vs Board Size

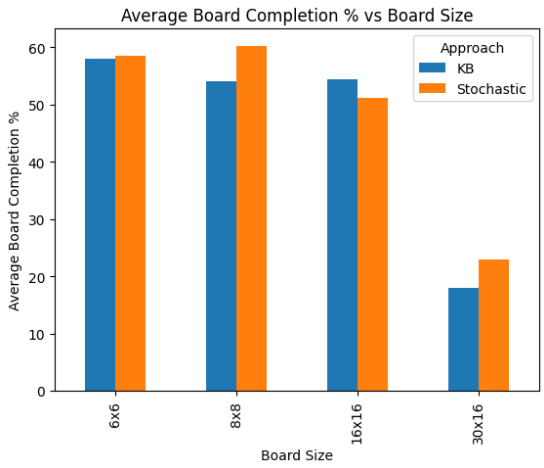

Fig. 9. Bar-graph depicting Average Board Completion vs Board Size

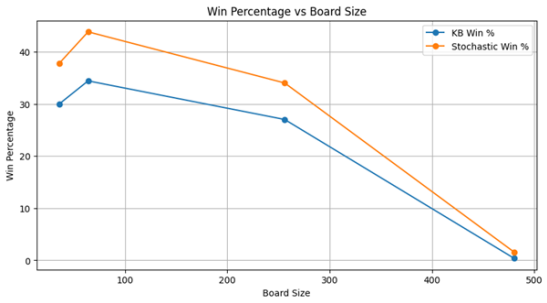

Fig. 10. Line-graph depicting Win% vs Board Size

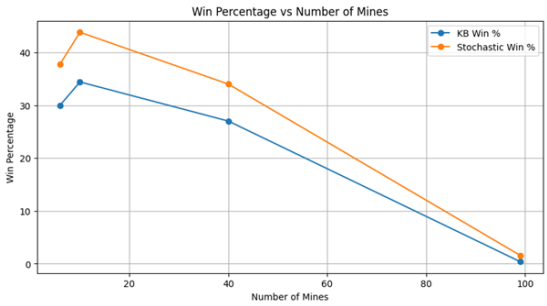

Fig. 11. Line-graph depicting Win% vs Number of Mines

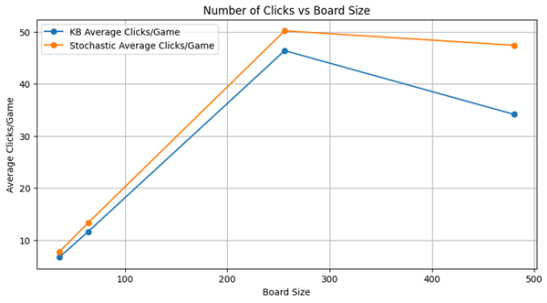

Fig. 12. Line-graph depicting Average Clicks vs Board Size

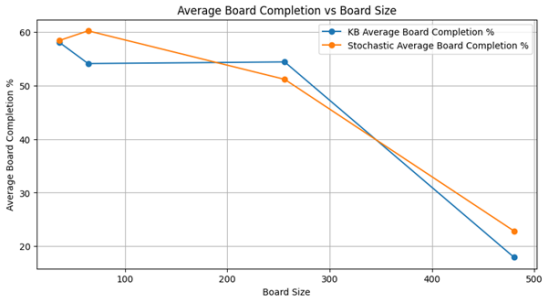

Fig. 13. Line-graph depicting Average Board Completion vs Board Size

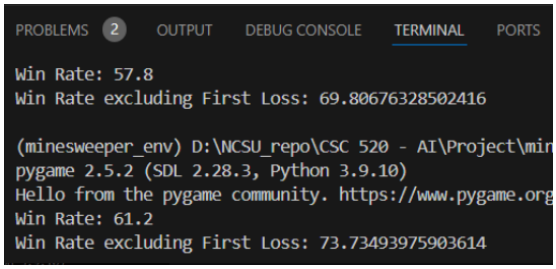

Fig. 14. Win Rate Percentage for the Reinforcement Learning Based Agent

For the Reinforcement Learning Agent, the Average Win% was in between 68% and 72% when 500 games were played over a 6x6 grid. Additionally, we notice that as the number of batches increases, the win rate steadily increases, plateauing over 50% when considering the first loss, and on an average of 70% when not considering first loss. Here, first loss means that the first click was unfortunately a mine.
Few instances of that depict this output are shown in Figure 14. This result clearly shows that Reinforcement Learning outperforms Knowledge Base CSP and Stochastic approaches. With same constraints such as the number of games, the board size and the number of mines, the Reinforcement Learning Agent has a Win% which is more than the Win Percentage of both the other agents combined. This is illustrated in the bar-graph shown in Figure 16.

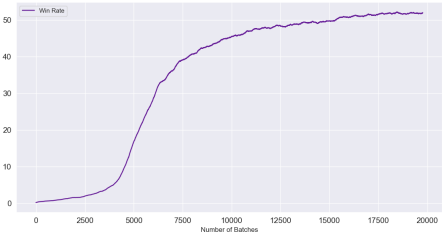

Fig. 15. RL agent Win rate vs Number of batches

VI. Conclusion

In conclusion, the reinforcement agent when compared to the basic and stochastic agent performs better. This is because it is able to figure out the best probability constraints for the environment. The probabilistic rules for the other two agents were designed by us and may have some rules missing

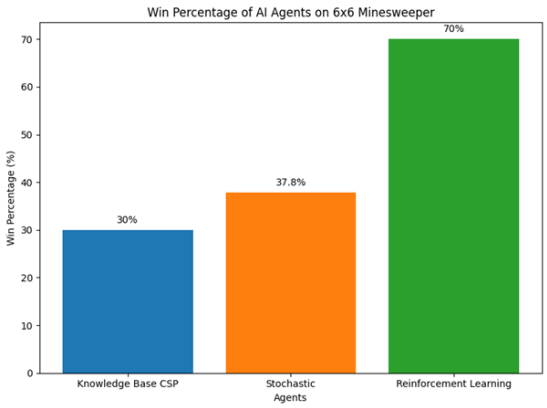

Fig. 16. Win Rate Percentage for the Reinforcement Learning Based Agent

which the RL agent may have implicitly learnt. However, RL agent was only trained on 6x6 board, which took around 10 hours to train. Given enough time and compute, RL would be able learn better and solve other unseen scenarios as well.
In comparison, the other two approaches do not require this much computation power but the construction of the probabilistic constraints is more complex and needs to be further refined.

References

[1] *Stuart J. Russell and Peter Norvig Artificial Intelligence: A Modern Approach*
[2] *Introduction to AI with Python* https://cs50.harvard.edu/ai/2020/projects/1/minesweeper/
[3] *Chris Studholme. Minesweeper as a constraint satisfaction problem*
[4] *Reinforcement Learning AI: Minesweeper* https://sdlee94.github.io/Minesweeper-AI-Reinforcement-Learning/
[5] *tkinter - Python interface to tcl/Tk* https://docs.python.org/3/library/tkinter.html