Assignment II

Intro to Artificial Intelligence

CS 16:198:520

Kaustubh N. Jadhav
knj25

Pranav Shivkumar
ps1029

Sanyam Jain
sj770

Introduction

The basic idea of this project is to simulate a Minesweeper game environment and develop agents that would solve the board for the user. In the Minesweeper game, the user is provided a board with covered cells, in which some cells contain mines. The objective of the game is to safely detect all the mines on the board without exploring the cells that contain the mines; if a user clicks on the cell with a mine, it's game over.

The idea behind this project is to simulate the same, with the exception that when the user opens a mine, the game doesn't end; instead, the user gets to keep going while identifying that cell as a mine. We have developed this project using the Tkinter library in Python as the GUI for the game.

Working of the GUI

In order to facilitate the game experience for the user, we created a window where the user can enter the desired size of the Minesweeper board as well as choose the desired agent that he/she wishes to use to solve the board. This window is shown below:

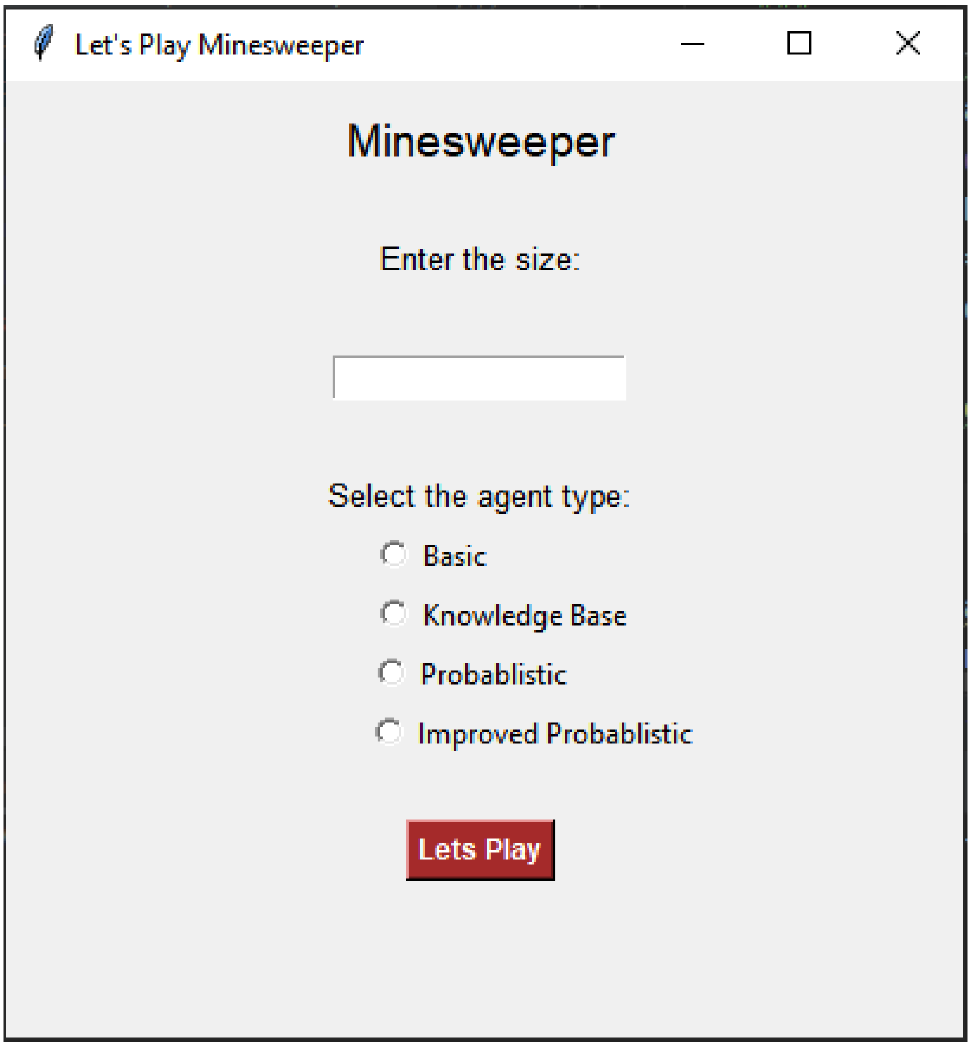


Figure 1: Window where the User can choose the size and agents

Once the user clicks on the "Let's Play" button, the board is generated, along with the first step that the user is recommended to click on. The sample Minesweeper board for a size of 10x10 is shown below:

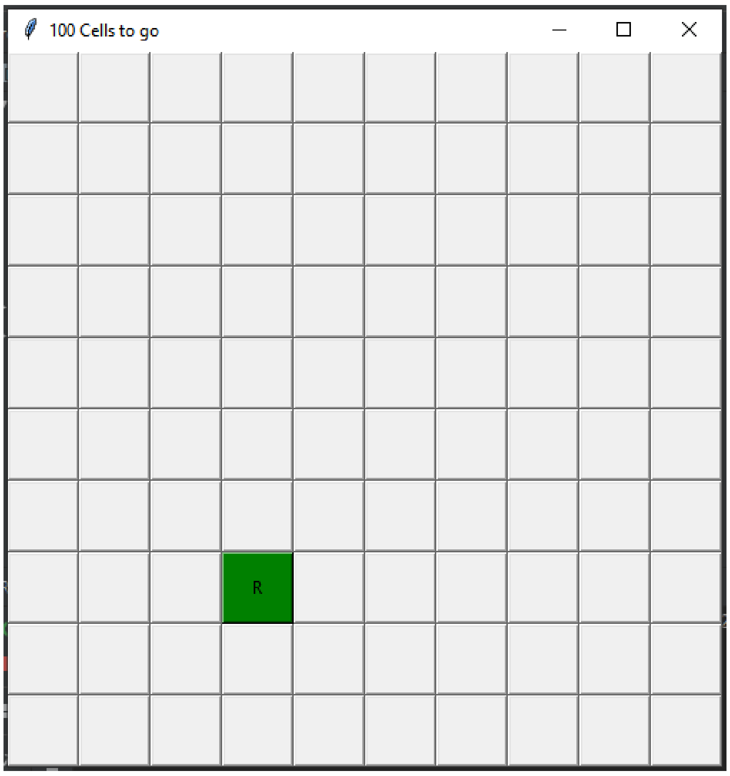


Figure 2: Sample 10x10 Minesweeper Board with a Random cell chosen to open

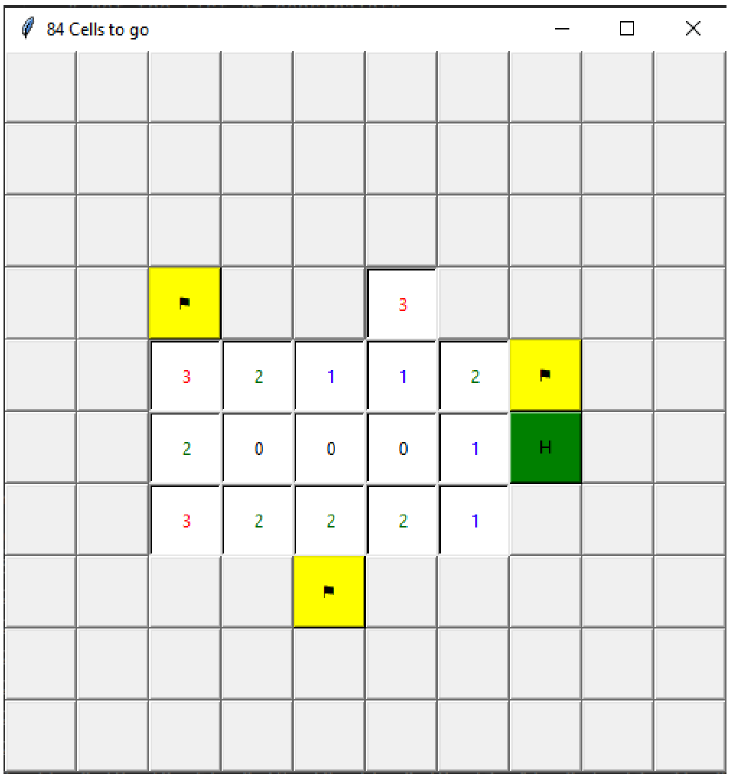
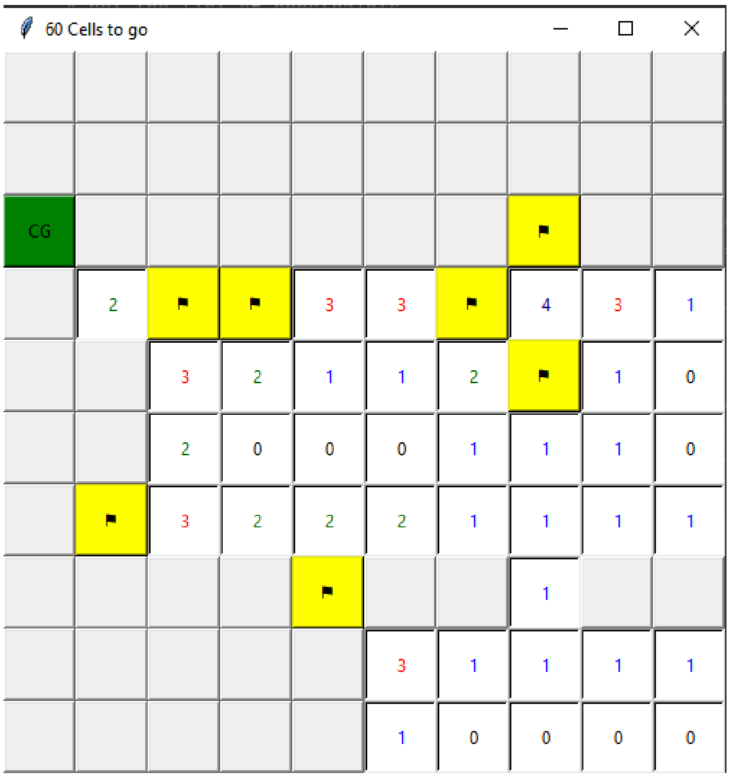
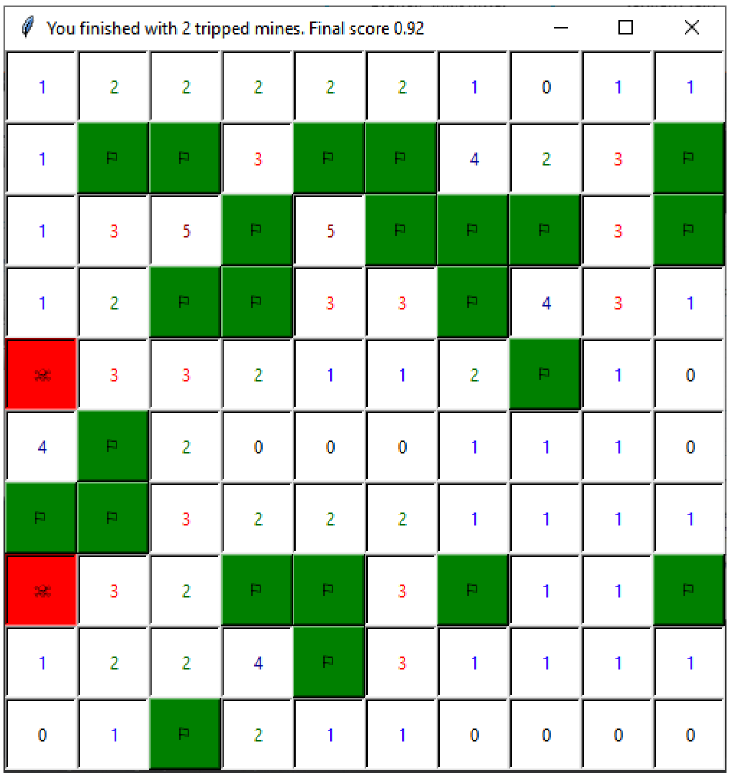


Figure 3: Sample 10x10 Minesweeper Board with a hint generated


60 Cells to go

CG							⚑		
	2	⚑	⚑	3	3	⚑	4	3	1
		3	2	1	1	2	⚑	1	0
		2	0	0	0	1	1	1	0
	⚑	3	2	2	2	1	1	1	1
				⚑			1		
					3	1	1	1	1
					1	0	0	0	0

Figure 4: Sample 10x10 Minesweeper Board with a calculated guess (CG)


You finished with 2 tripped mines. Final score 0.92

1	2	2	2	2	2	1	0	1	1
1	⚑	⚑	3	⚑	⚑	4	2	3	⚑
1	3	5	⚑	5	⚑	⚑	⚑	3	⚑
1	2	⚑	⚑	3	3	⚑	4	3	1
☠	3	3	2	1	1	2	⚑	1	0
4	⚑	2	0	0	0	1	1	1	0
⚑	⚑	3	2	2	2	1	1	1	1
☠	3	2	⚑	⚑	3	⚑	1	1	⚑
1	2	2	4	⚑	3	1	1	1	1
0	1	⚑	2	1	1	0	0	0	0

Figure 5: Completed Minesweeper Board

As seen in the figures above, the green block denotes the recommended cell that the agent recommends the user to open next. This recommendation can be of three types depending on the agent. If agent is Basic or knowledge based suggestion can be either a hint "H" or random suggestion "R". Hints are identified as safe cells while random suggestion is just a random cell from remaining cells. For probabilistic and improved probabilistic agent there is another type of suggestion "CG" which is "Calculated Guess" which represents the cells with minimum probability of being mine as per current broad. All the identified mines are represented by "yellow cell" with "black flag" and the busted mines are represented by "red cell" with "skull". Once the game is completed all the correctly identified mines turns into "green cell" with black flag and final score ratio (number of correctly identified mines/ total number of mine) is displayed in window title. This is shown in Fig 5.

Program Specification

There are two parts to our solution, as mentioned in the problem statement:

- ### A. Environment

  The basic user interface for the game was developed using the Tkinter library in Python, which provides the user with a visual experience of the game.

  The Python files used to implement this are described below:

  1. ***MinesweeperInteractive*** This Python file contains the implementation for all the agents from the perspective of interactive play. As mentioned previously, the Minesweeper board will have cells which can either be a mine or safe, a safe cell will have a clue denoting number of mines around that cell. Game ends when user has identified all hidden mines. For this implementation we won't terminate game if user selects a mine. And at the end of the game we display final score for the game which is ratio number of mines identified/number of total mines.

     We have also created separate python files for the agents so that we can analyse them more closely. The functions of these files are described below:
  2. ***Minesweeper1***
     This file contains the implementation of the basic agent for solving the minesweeper board. We have also added a mode parameter, which can be either *Analysis* or *Test*. In Analysis, we analyze the performance of the agents by plotting the graphs and in the Test mode, we display the final board and score after the mines were detected. The next step is to define the number of mines, which is where the mine density comes into play. Using the size of the board and the mine density, we can get the number of mines in the board. These are randomly assigned locations and are scattered across the board.
  3. ***Minesweeper 2***
     This file contains the implementation of the knowledge base based agent and similar to the previous file, includes the mode parameter to analyze and test the agent.
  4. ***Minesweeper 3***
     This file contains the implementations of the probabilistic and the improved probabilistic agents. This also contains the mode parameter to analyze and test.

- ## B. Agents

  Now that the Minesweeper board is set up, the next step is to play the game. To facilitate the game play for the user, we had designed a couple of agents which would guide the user to open particular cells. Initially, all the cells are covered (C). These agents are described below:

  1. **Basic Agent**
     This agent is basically a simple Minesweeper board solver. On the first step, the agent randomly chooses a cell to open. It then checks if that cell is safe or a mine. This agent stores this information and for a safe cell it also updates the clue which represents the number of mines surrounding the cell. This agent uses the clue for each cell to check if cell surrounding that cells are safe or mines. This is limited as it can give prediction only when all of uncovered cells around the concerned cell are either mine or safe. The inference if based on the below equations:

     $\boldsymbol{\#of\, covered\, cells\; = \; Clue\; - \; \#of\; mines\, discovered\, till\, now}$
     Above equation represents the case when all the uncovered cells around the cell are mine.
     $\boldsymbol{\#of\, covered\, cells\; = \#of\; neighbours\, around\, cell\; - \; Clue}$
     A safe cell holds a value of 0 and a cell with a mine holds a value of 1.
     Above equation represents the case when all the uncovered cells around the cell are safe.

     For this agent hints are generated only on the bases local information for a cell.
  2. **Knowledge Base**
     This agent improves off the previous solver; as it uses the knowledge base updated at each step more wisely. For each and every cell knowledge base contains relevant information like:

     $\boldsymbol{\#ofneighbours,\; list\, of\; neighbours,\; status(Covered,\; Safe\, or\; Mine)\; and}$
     $\boldsymbol{Clue(if\; safe)}$.

     Agent use this knowledge to create local constraints for whole broad at a each step. e.g. let *A, B, C are the uncovered neighbours of cell with a clue = 2* then this knowledge will give rise to constraint like $\boldsymbol{A\; +\; B\; +\; C\; =\; 2}$ *,this signifies that A, B, C must be such that their sum can not be greater than 2 as for previous agent value of A, B, C can either be 0 and 1.* Once we have bunch of constraints like that, they as whole represents the current state of the minesweeper broad. Solving these constraints we can get more information and can mark identifiable cells as safe or mine.

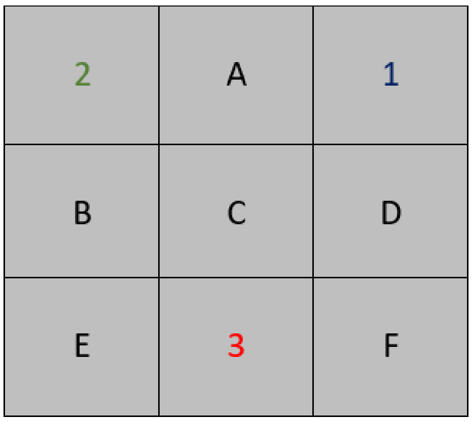


Figure 6: Sample minesweeper broad

Taking example sample minesweeper broad in Figure 6 we will have three constraints.

$$\boldsymbol{A\; +\; B\; +\; C\; =\; 2} \tag{1}$$

$$\boldsymbol{A\; +\; C\; +\; D\; =\; 1} \tag{2}$$

$$\boldsymbol{B + C + D + E + F = 3} \tag{3}$$

Solving these constraint i.e. subtracting *eq 2* from *eq 1* and using the environment constraint that each variable can either be 0 or 1 we can infer that *B is mine* and *D is safe.* Using this new found information we can get new constraints and get some new information. When this agent can not find any new information then this agent also resort to random suggestion.

In Our implementation first we solve "trivial equations" which can be of two types $\boldsymbol{A + B + C = 0\, and\, A + B + C = 3}$. These equations on there own give solution for all constraint variables i.e. either all are safe or all are mine. Once we solve trivial equations, solutions are used to reduce remaining equations and if the remaining equations result in some trivial equations they are also solved(This is done recursively). Once no more trivial equations remains we subtract remaining equation in pairs and check if they can result in some trivial equation if so they solved to fine solution.

3. **Probabilistic (Doubly Improved Agent)**
This improved agent works by assigning probabilities to the cells around the opened cells. To determine the probabilities this agent uses the knowledge base to create constraint and then uses those constraints to generate possible solutions. To generate possible solutions this agent uses backtracking search to keep the assignment for each constraint variable such that it satisfy all constraints imposed on it. Once we have all such solutions it computes probability of being mine for each variable using below equations:

$$\boldsymbol{P(node = mine) = \#of\; solutions\, node\, is\, mine/\#of\; solutions} \tag{4}$$

Taking example sample minesweeper broad in Figure 6, we will have below possible solutions.

$$\boldsymbol{A = 1,\; B = 1,\; C = 0,\; D = 0,\; E = 1,\; F = 1} \tag{5}$$

$$\boldsymbol{A = 0,\; B = 1,\; C = 1,\; D = 0,\; E = 0,\; F = 1} \tag{6}$$

$$\boldsymbol{A = 0,\; B = 1,\; C = 1,\; D = 0,\; E = 1,\; F = 0} \tag{7}$$

Thus the probabilities for each node to be mine will be will be .

$$\boldsymbol{P(A = mine) = 0.33,\; P(B = mine) = 1,\; P(C = mine) = 0.67,} \tag{8}$$

$$\boldsymbol{P(D = mine) = 0,\; P(E = mine) = 0.5,\; P(C = mine) = 0.5} \tag{9}$$

Using these probabilities we can mark *B as mine* and *D as safe.* One major benefit of this agent is that this agent will rarely make a random guess when we are out of safe cells and most if not all of it suggestion will either be safe cell or the cell with lowest probability of being mine.

4. **Improved Probabilistic (Triply Improved Agent)**
This is enhanced version of Probabilistic agent. Here we have made two improvement one is generic optimization for Constraint Satisfaction Problems "which selecting the constraint variable for solution in a specific order". In this implementation we are ordering the constraint variables from most constraint to least constraint. A variable which appears in more constraint equation is more constraint then variable that appears in less number of constraint. Thus for all variables in constraint equations we maintain the dictionary where key variable name and value it some constraint value of equations in which this variable exists. For figure 6 dictionary will be like.

$$\boldsymbol{\{A : 3,\; B : 5,\; C = 6,\; D = 5,\; E = 3,\; F = 3\}} \tag{10}$$

We are using constraint value so that we can avoid variables occurring in same number equations having same value.

Second improvement is based on fact that if we know the total number of mines that are present on minesweeper broad, then we can at each step calculate the remaining mines. we can use this information to reject some of the solutions given by backtracking search. For e.g. for figure 6 we have three solution two of them contain 3 mines (eq 6 and eq 7) while one (eq 5) has 4 mines. If we know that we have only three mine remaining that we can reject the solution with 4 mines and we will have more correct probabilities and we can better information.

$$\boldsymbol{P(A = mine) = 0,\ P(B = mine) = 1,\ P(C = mine) = 1,} \tag{11}$$

$$\boldsymbol{P(D = mine) = 0,\ P(E = mine) = 0.5,\ P(C = mine) = 0.5} \tag{12}$$

Using these probabilities we can mark *B and C as mine* and *D and A as safe.*
While first improvement reduce should reduce the time taken by simple probabilistic agent the second improvement is aims to improve probabilities. Second improvement comes handy at the end of the game.

Questions and Write-up

1. **Representation: How did 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?**

   Ans: The Minesweeper board is represented as a 2D array with row and column equal to the size defined by the user. Form this 2D array specified number of cells (based on the given mine density) are chosen and are updated into list of mines. Information of each cell is maintained as dictionary, Where keys are the indices for each cell and value itself is dictionary given by **{"neighbour": indices of neighbour, "neighbours": number of neighbour, "status": (Covered, Safe or Mine), "clue": (number if safe "ni" if mine)}**. This implementation is the same for all agents.

   For the basic agent, inferred relationship b/w neighbours of cell with clue is represented by a mathematical equation which is solved at each step as per the logic of the basic agent

   For the other agents, we will get a constraint for each clue and we have to store that constraint since when it is combined with other constraints it may give useful insight. Thus for these agents a constraint is represented as a dictionary **{"const": List of variables(neighbours), "val": clue value - mine of cell discovered around the cell}**. At each step for each cell with clue and some covered neighbour constraint is created like this and added into a list. Later we iterate over this list and solve constraint variables.
   For each agent we maintain a list of Open cells, Cells identified as safe, Flagged cells(identified as mine), Cells busted by user and solved cells(cells with clue but all neighbours are uncovered).

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?**
   Ans: When we open a new cell we update our knowledge base. This update is same for each agent. First cell is updated in the list of open cells, then its status is updated in its dictionary, if it is mine status is updated as "M" and if safe status is updated as "S". For safe cells clue column is also updated. Thus after each opening of cell knowledge base is updated.
   Based on the new clue and fact that new cells is revealed basic agent check if there are cells for which its logic of "all covered neighbouring cells are mine" or "all covered neighbouring cells are safe" is applicable. If applicable cells are updated in respective list.

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?**
   Ans:

   - **Basic agent** The agent stores the information of the opened cell neighbors and for a safe cell it also updates the clue which represents the number of mines surrounding the cell. This agent uses the clue for each cell to check if cell surrounding that cells are safe or mines. For all the safe cells, the cells are added in the stack. And to generate the hint, one element is popped out.

- **Knowledge Base agent** Agent use this knowledge to create local constraints for whole broad at a each step. e.g. let *A, B, C are the uncovered neighbours of cell with a clue = 2* then this knowledge will give rise to constraint like $\boldsymbol{A + B + C = 2}$ *,this signifies that A, B, C must be such that their sum can not be greater than 2 as for previous agent value of A, B, C can either be 0 and 1.* Once we have bunch of constraints like that, they as whole represents the current state of the minesweeper broad. Solving these constraints we can get more information and can mark identifiable cells as safe or mine. The safest cell is then opened.

- **Probabilistic agent** At first, a random cell is suggested to open. After first cell is opened, all the cells besides that will have same probability, except 0 case. After probability is assigned to the neighbors, a list is created and all the unopened neighbors are added with specific probabilities. This list is used to suggest the next opening box. A calculated guess is suggested at random due to similar probabilities. After opening the calculated guess, the neighbors for that are added to the same list after calculating their probabilities. And so whoever has the less probability will be opened next from that list.

- **Improved Probabilistic agent** This improved agent works by assigning probabilities to the cells around the opened cells. To determine the probabilities this agent uses the knowledge base to create constraint and then uses those constraints to generate possible solutions. To generate possible solutions this agent uses backtracking search to keep the assignment for each constraint variable such that it satisfy all constraints imposed on it. Once we have all such solutions it computes probability of being mine. And selects the cell to open based on the least probability.

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?**

   Ans:

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?**

   Ans: The plots are shown below:

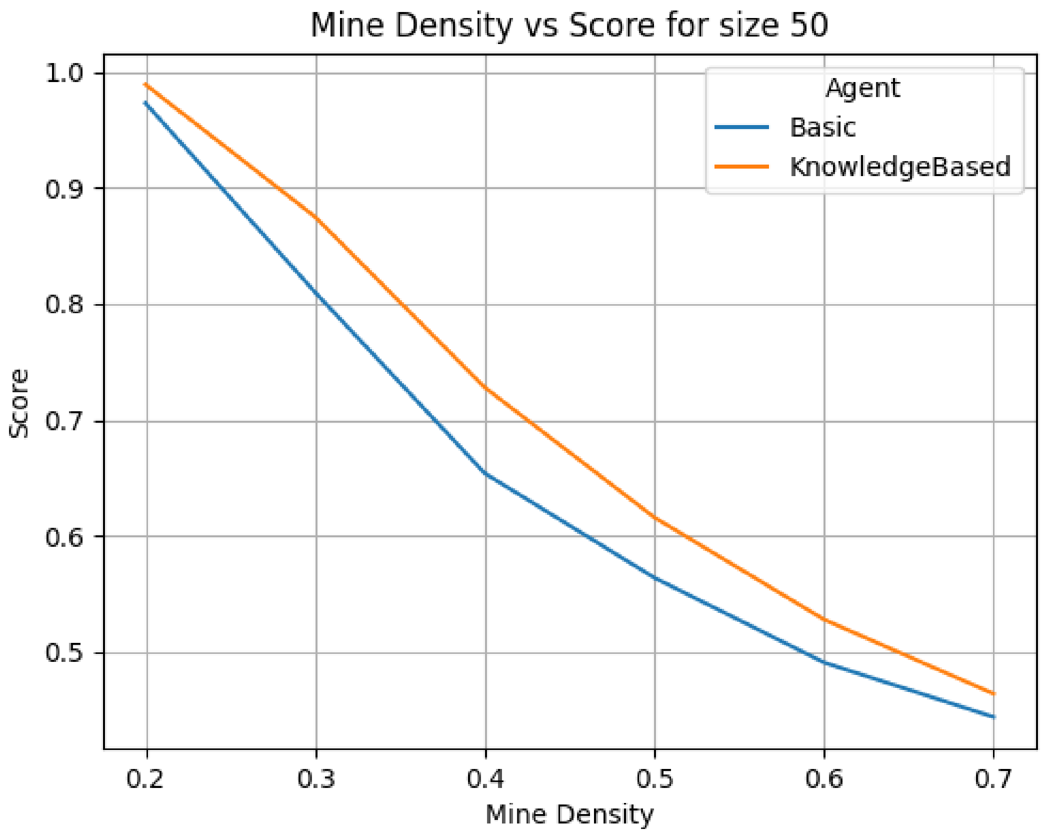


Figure 7: Mine density vs Final score for 50x50 board

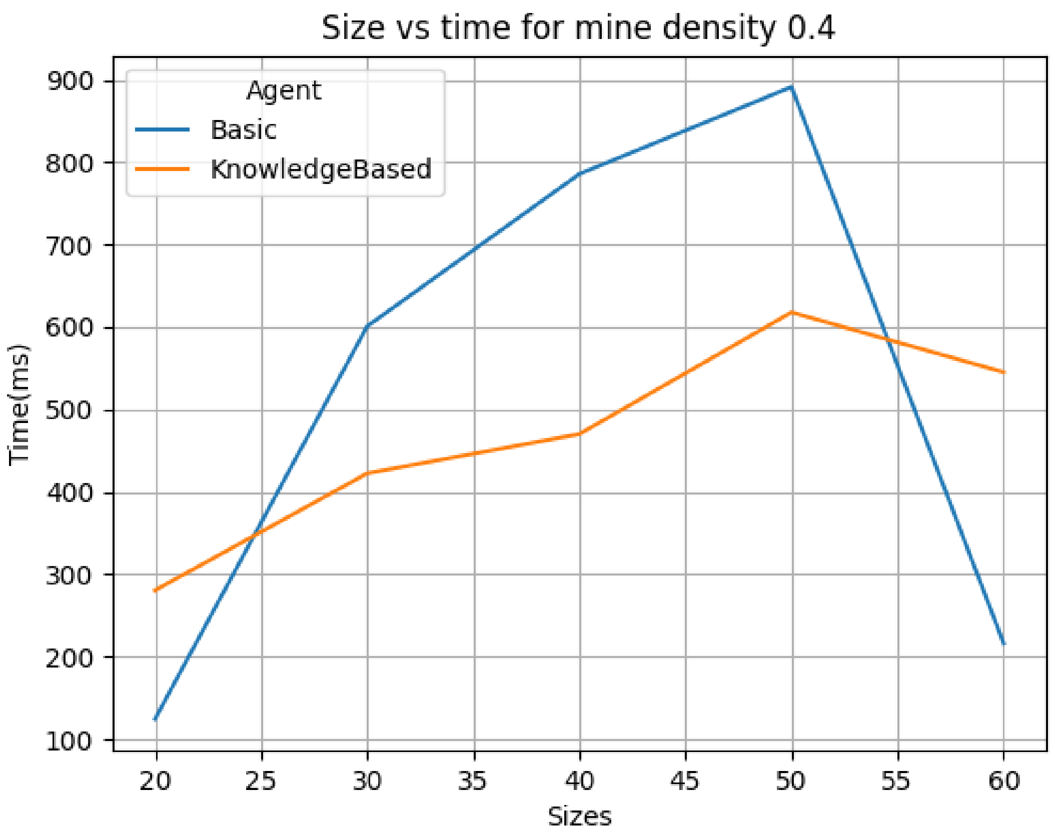


Figure 8: Board Size vs time for mine density of 0.4

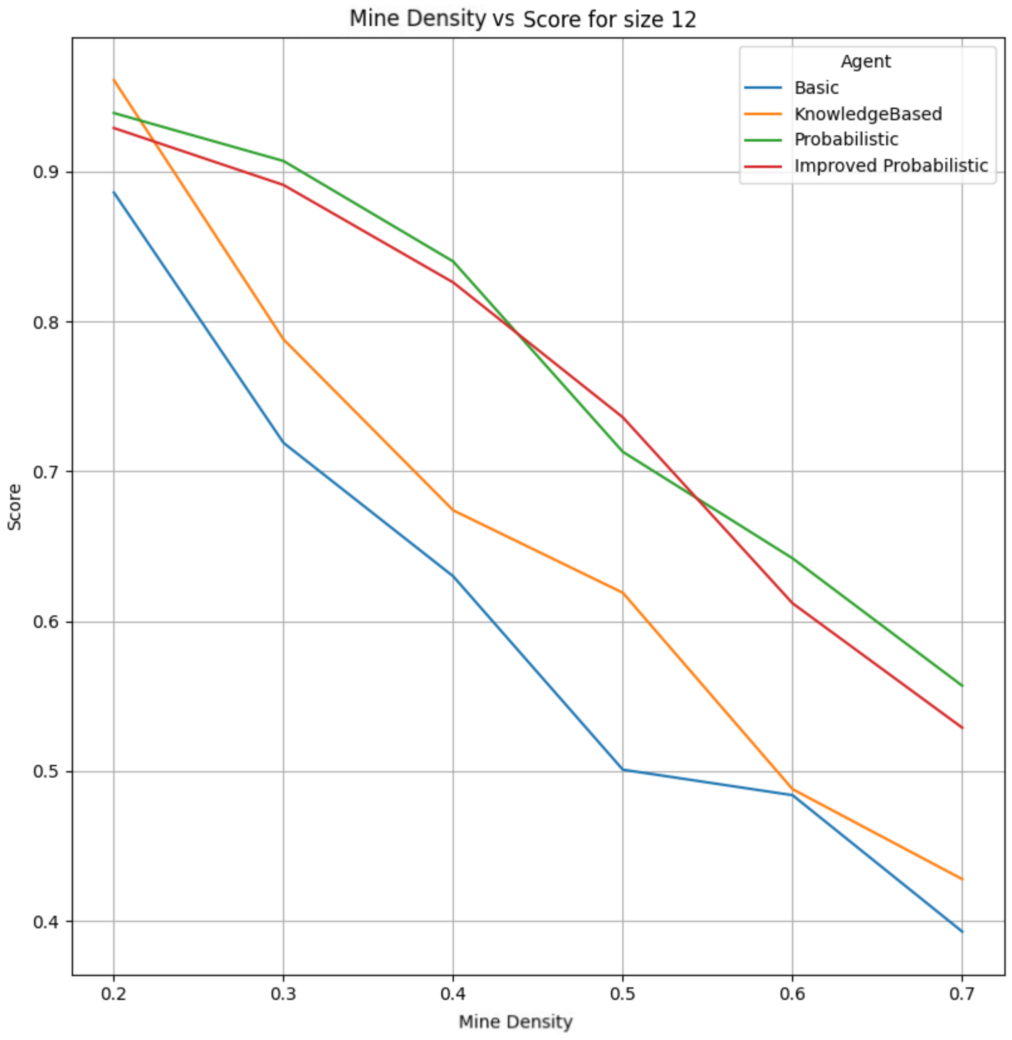


Figure 9: Mine Density vs Score

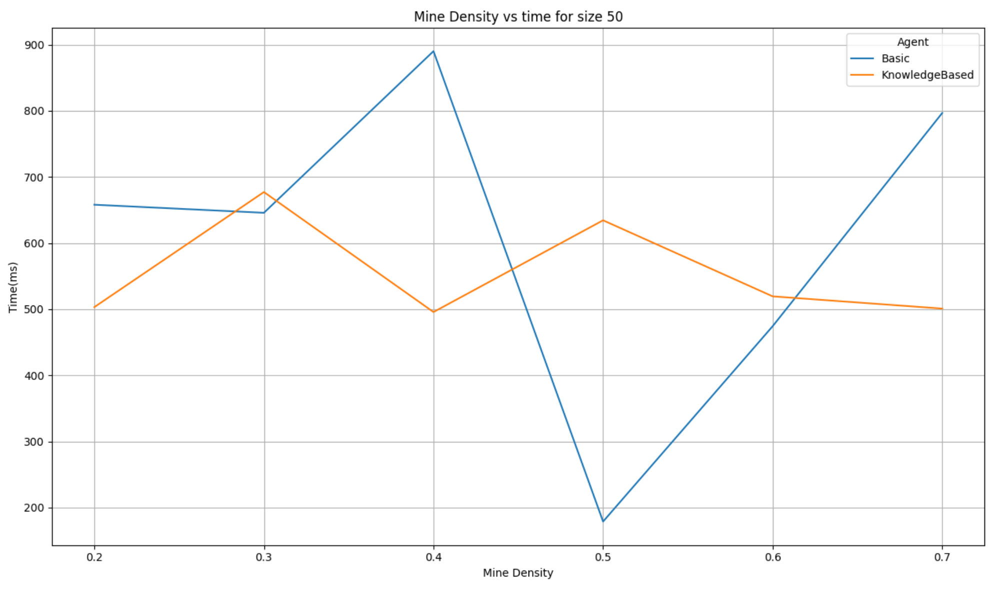


Figure 10: Mine density vs size for 50x50 Board

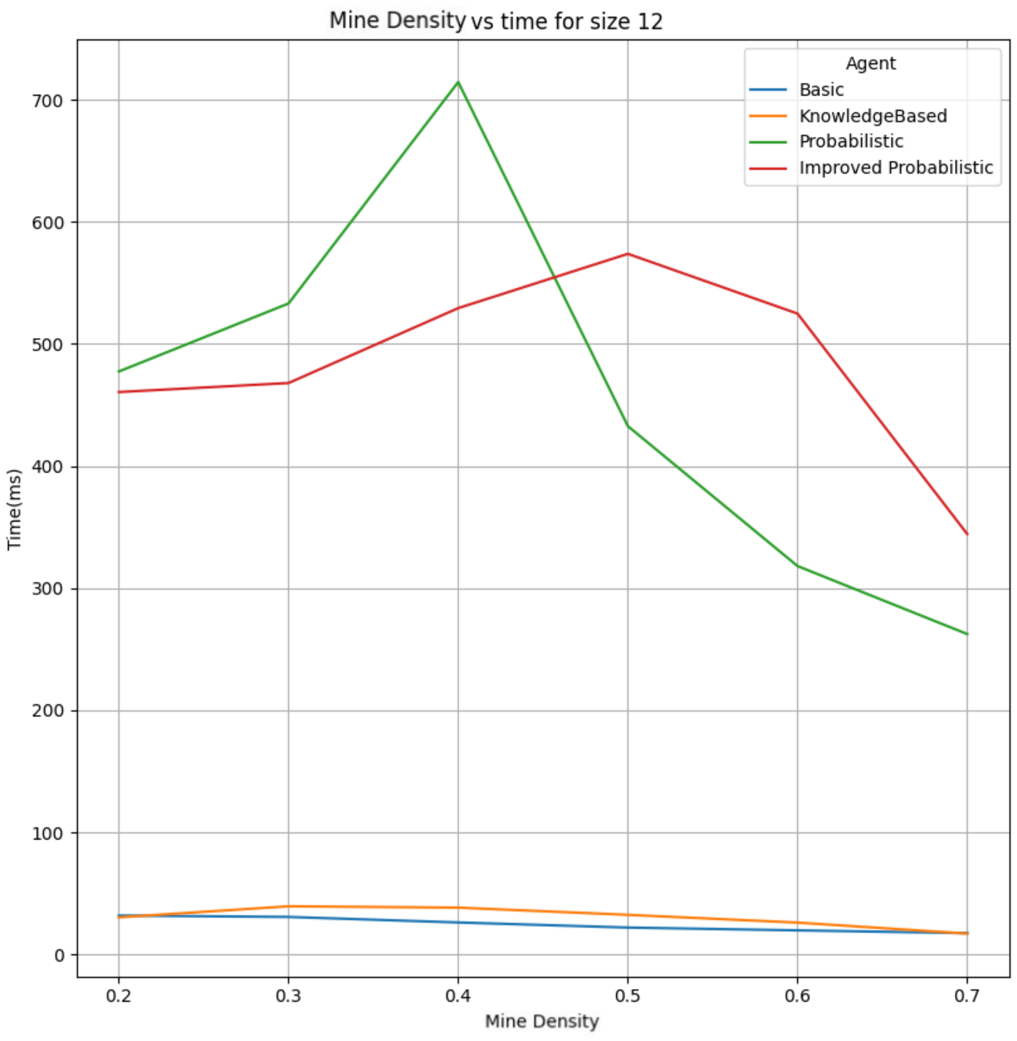


Figure 11: Mine Density vs Time Taken in milliseconds

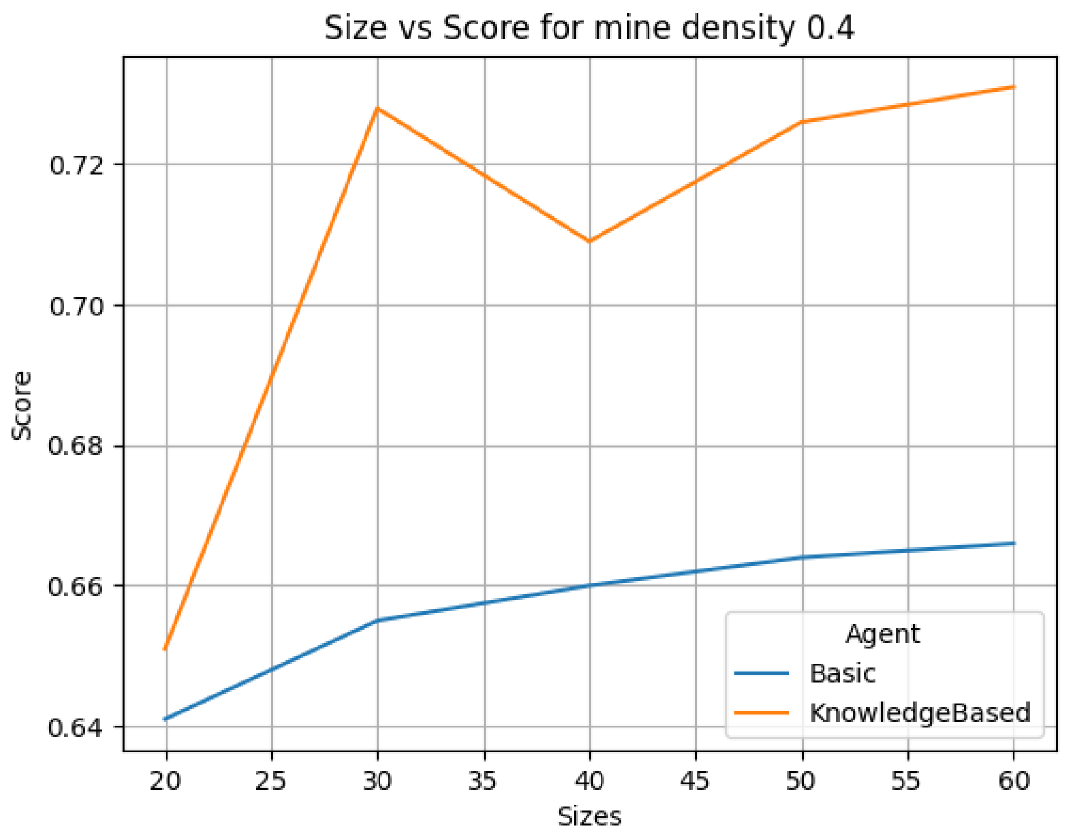


Figure 12: Board size vs Final Score for mine density of 0.4

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?

   Ans: