CS 440 **Assignment 2 - Inference-Informed Action**
Spring 2021 **GITHUB:** https://github.com/Norden-Tenzin/440ArtificialIntelligence

Name: (Sangkyun Kim) **NetID**: (sk1998) **Section**: (05)
Name: (Tenzin Norden) **NetID**: (tn266) **Section**: (08)

1. **Instructions**
   (a) To RUN:
      i. Use arguments:
         "-ui": To view and interact with the game.
         "-test": To run basic and advanced algorithm 100 times each
         it also writes the average of each to a results.txt file and prints that data in the terminal as well.
         "-cmd": This command requires other commands to be used with it. it uses the terminal to output its data.
         A. "-basic": This runs the basic algorithm and prints out the score and some relating data.
         B. "-advanced": This runs the advanced algorithm and prints out the score and some relating data. but it requires "True" or "False" argument along with itself. If True it uses the better decisions to pick its querys. But if False it uses the Advanced random pick.

         Examples:
         "python main.py -ui"
         "python main.py -cmd -advanced True"
   (b) Inside UI:
      i. Cells: The cells can be left clicked to reveal the number or mine underneath. They can also be right clicked to flag the cell for a mine.
      ii. Helper: Is a toggle button which when pressed displays the data gathered by the basic algorithm. It shows green color for safe cells and yellow for mine cells.
      iii. New Maze: When clicked it generates a new maze.
      iv. Reset Maze: when clicked it resets the maze to its initial state.
   (c) To change the dimensions of the UI edit the DIM inside the constants.py. When changed it will also change the size of each cell in the UI as well.
      Mine Density can also be changed inside constants.py
2. **Representation**
   (a) The minesweeper game is represented as multiple 2D array in mineMap.py. It stores the editable 2D array (curr) which is used to keep track of the game's current condition it also holds a 2D array (answers) which knows the positions of each cell.

(b) The knowlede we get from clues is stored in another 2D array (help) which is also inside mineMap.py. This data is used by the UI to display safe and mine containing cells. For the UI we used pygame to display the game and make it intractable for the user.

(c) We created knowledge class in knowledge.py to model the inferred relationship between cells. We modeled constraint equation as tuples. In the tuple, the first value is the list of cell's coordinates(unclicked neighbor cell) and the second value is the clue. This tuples are saved in the list.

(d) dimension of the board and the number of mines can be changed by modifying variables in constants.py.

3. **Inference**

   (a) When the program query a cell, unclicked neighbor cells are appended to a list and saved in a tuple as the first value and the clue of the queried cell will be save in the tuple as the second value(def add knowledge).
   This knowledge information continues until the game ends. Knowledge is also modified when some cell in the information is flagged, or revealed as a safe cell(def remove knowledge), or one of the equations in the information becomes part of the other equations subset(def update knowledge).
   If one of the cell in the knowledge is revealed as a safe, the program remove this cell from all of the information in the list. If the one of the cell in the knowledge is revealed as a mine, the program also remove this cell and edits the second value of the tuple(clue) by decrementing it by 1. When one of the information in the list is part of the other information, the program delete the subset.

   (b) After adding and modifying information in the knowledge, the program deduce this knowledge to figure out which cell in the knowledge is safe or mine. The program calculate the length of first value of tuple(list of coordinates), if this length is same as the second value of the tuple(clue value), all the coordinates in the list are mine cell. However, if the clue is 0, all the cells are safe cell.

4. **Decisions**

   (a) The program compute which cell is safe or a mine and it saves the cells in a safe cell list or mine cell list. After this process the program select cell from one of the lists and queries it until both lists become empty. When both of the lists are empty, the program picks a random unclicked cell on the board.

5. **Performance:**

   (a) There are cases when I'm surprised with the programs decision. In the figure 1, in this situation the program deduce that cell d, f, g are safe mine. In the knowledge, a

a	b	c
d	2	f
g	2	mine

Figure 1: Minesweeper

+ b + c + d + f + g = 1 and d + f + g = 1. Since second equation is the subset of the first one, there is only one equation (a + b + c = 1) remains in the knowledge. The program can do this because it can mix all the information in the knowledge. However, I cannot play like the program does.

(b) I don't agree with the programs random querying. When there is no possible knowledge to figure out, The program has to pick a random cell on the board. However, sometimes it does not care about the probability and can select the cells that have a higher probability of an existing mine.

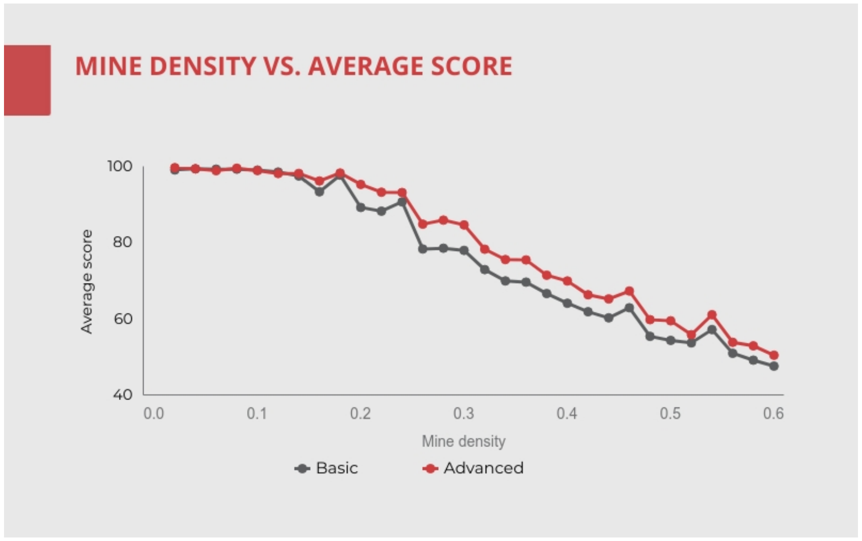

Figure 2: Performance

(c) We take 16 x 16 environment gird and set mine density from 0.01 to 0.6 to compare the final score of both basic and advanced algorithms.The plot regarding the result is in Figure 2. When the mine density is low, the basic algorithm's average final score is better sometimes and the speed of processing also much faster. However, when minesweeper becomes hard (mine density >= 0.15), the advanced algorithm reaches higher score. This is because when the mine density is high, that means the program more likely select mine cell when they randomly picking. However, the number of picking randomly of advanced algorithm is always lower than the basic one. Hence When the game becomes hard, the advanced algorithm's performance gets better.

6. **Efficiency**

   (a) While implementing the advanced Algorithm, we did encounter a time constraint.The knowledge base of algorithm is represented by a numpy list with the equations as tuples. When the cell is removed from other constraint equations, it pass through its total knowledge base. This is time consuming process. If the size of grid is getting larger, it leads to increase in time complexity.

   (b) We can improve this problem by accessing the data directly rather passing through entire list.

7. **Better decisions**

   (a) We used concept of probability as a better selection mechanism. The idea is we calculate all the possible scenarios based on the knowledge by using combination and product. Then we compute the probability of each cell those in the equations in knowledge(number of cell showing up in the scenario / number of total scenario). The rest of unclicked cell in the map share probability like this (total number of covered mine / total number of unclicked cell in the map). We put all the cells with probability in the dictionary picking minimum value from it.

8. **Acronym**

   (a) S.H.A.R.T
   Sharp Handy AI in Real Time

9. **Summary**

   (a) We split the work 50 50 and worked on the entire project together. We had daily meetings to discuss our progress. We used github as a collaboration tool. We would talk about the issue at hand before tackling it and try to break down the problem into smaller more easy to solve problems.