CS 440

Project 2: Minesweeper

Professor Wes Cowan

March 13, 2020

Krupal Patel (kp766)
Zain Siddiqui (zas30)
Suraj Kakkad (spk101)
Melanie Wong (mw845)

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?

We represented the board using two 2-dimensional lists within a Game object. One of the two boards contains a minefield that is displayed to the player. This board consists of '#' for unrevealed cells, 'F' for flagged cells, and 'M' for mines. Furthermore, the other board is used to store hidden information which, when clicked, is revealed about each cell. This board consists of a 0 for safe cells and -1 for mines. We stored the information/knowledge about clue cells within the hidden board, once we have established all the safe cells in the minefield. Thus, when a player reveals a safe cell, the respective cell will display the clue from the hidden board. We can represent inferred relationships between cells by storing information about each cell's neighbors and whether these neighbors are mines or safe cells. Essentially, this is done through each cell's clue value. Additionally, we added a Tile object which helped us store long term information about cells within our game board. Specifically for the improved agent this assisted with keeping track of the constraint values and equations for each cell which helped us make inferences using our constructed knowledge base. To represent the environment we also implemented a GUI using Tkinter as our frontend to show and play the Minesweeper game.

Figure: Dimension = 15, Mines = 15

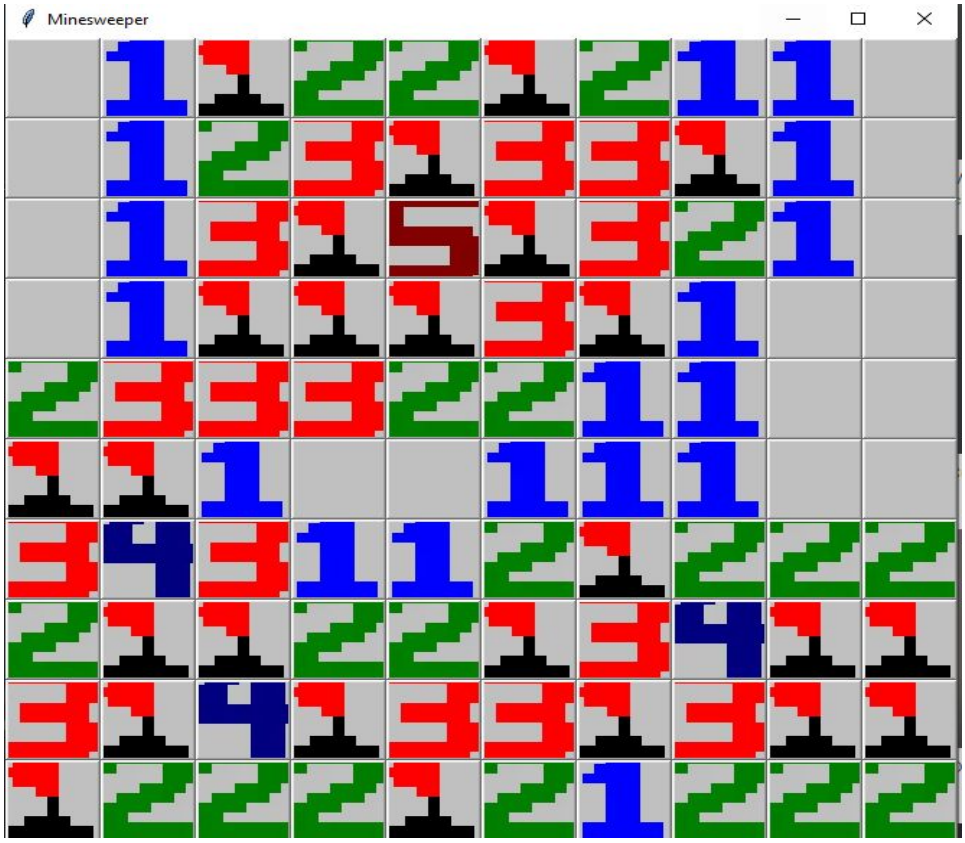


Figure: Dimension = 10, Mines = 25

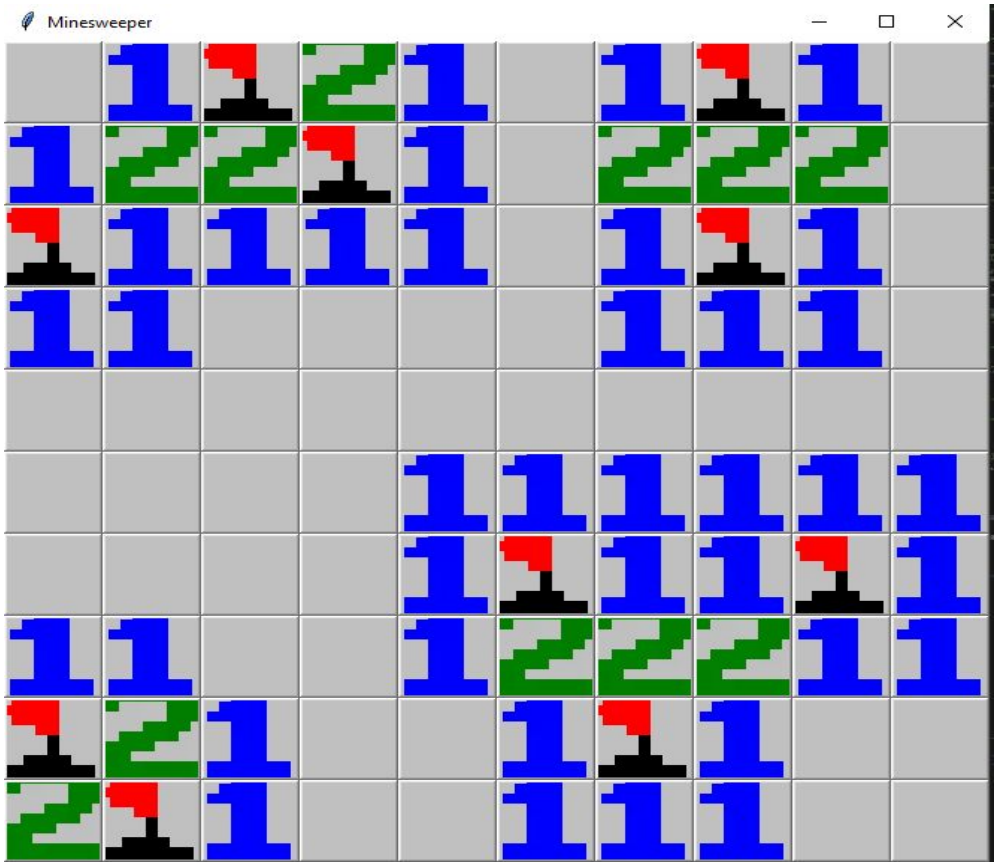


Figure: Dimension = 10, Mines = 10

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?

When we collect a new clue from a given cell, we update the constraint equations and constraint values for the current cell. Furthermore, we add these constraint equations and values to the global knowledge base consisting of constraint equation and constraint value tuples. We also update the list of known mine cells and safe cells, if we arrive at a solid conclusion based on the clue. Since deducing every equation can be computationally intense, our program iterates through all equations in the knowledgebase and reasons what it can from that single run. With greater computing power and memory resources, we can run multiple iterations of the knowledge base to make stronger conclusions of the clues present.

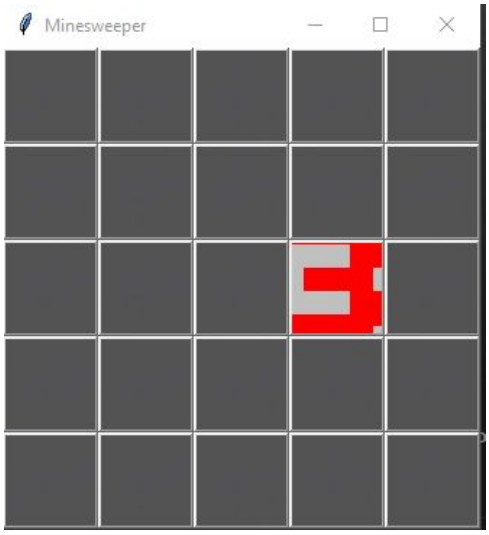
Performance

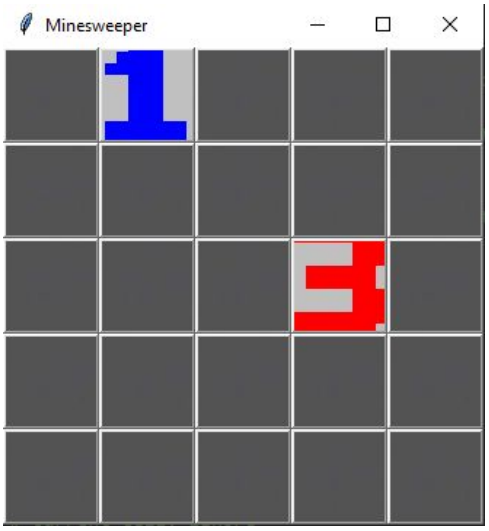
For a reasonably-sized board and a reasonable number of mines, include a play-by-play progression to completion or loss. Are there any points where your program makes a decision that you don't agree with? Are there any points where your program made a decision that surprised you? Why was your program able to make that decision?

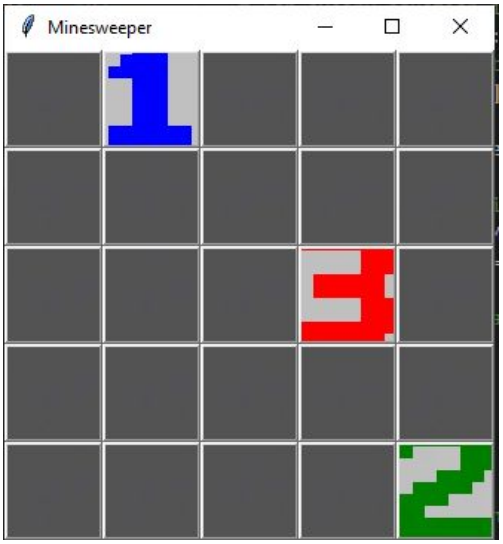
During the play by play progression for the 5x5 board with 5 mines the decision making was very good and resulted in a completion of the game. However, there are points in the play-by-play where the agent makes the correct move, but fails to apply all of it's knowledge before making the move. Although the move is correct and uses the backend constraints of our improved algorithm, at some points it will reveal a cell instead of flagging known mine cells until a later iteration. Although this may not seem like a terrible situation as a player I expect the agent to first apply more important knowledge and flag a mine cell before revealing another one. An example of this is seen at iteration 20 where the agent knows there is a mine cell at (3,4), but chooses to reveal cells before flagging.

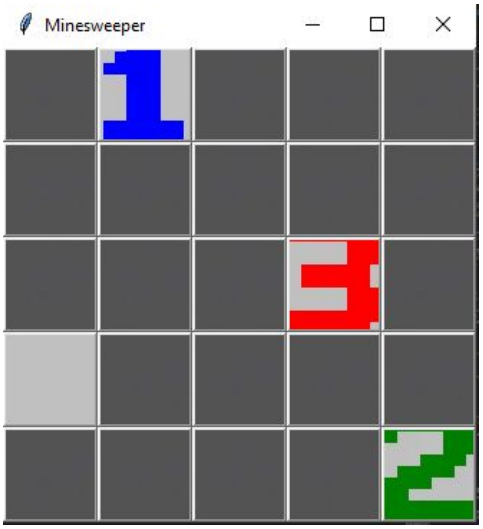
During the progression there wasn't much that was surprising. The program operated only on constraints and knowledge given aside from the initial random cells. After the first safe cell was found and we gained information about the board it was easier for the agent to only make decisions based on our knowledge base at that point. This progression was very generous, but something that would have been surprising would have been if we didn't have any concrete knowledge and the agent had to choose a cell with limited knowledge or choose randomly.

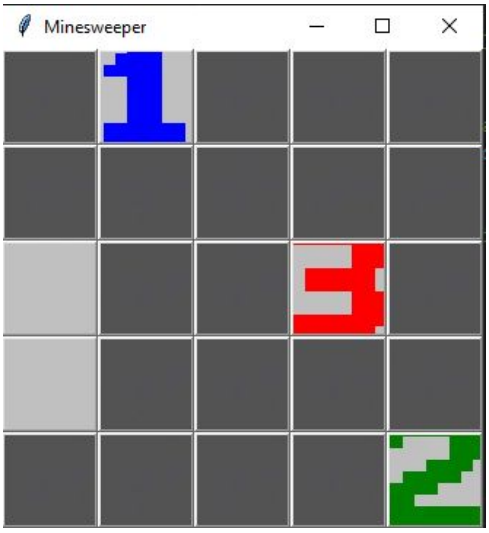
Below is a play-by-play progression to completion of a board of dimension 5 with 5 mines. The moves are depicted by the screenshots from left to right,

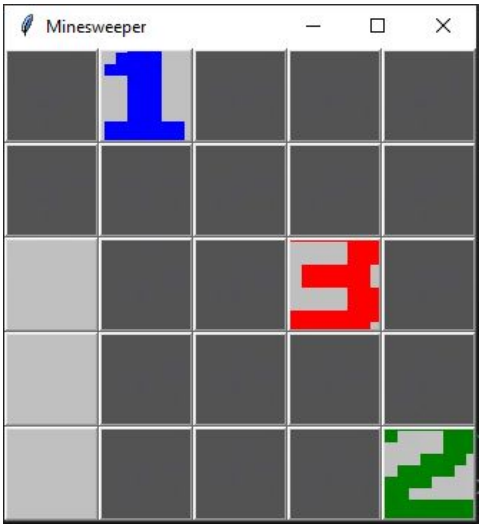

Minesweeper

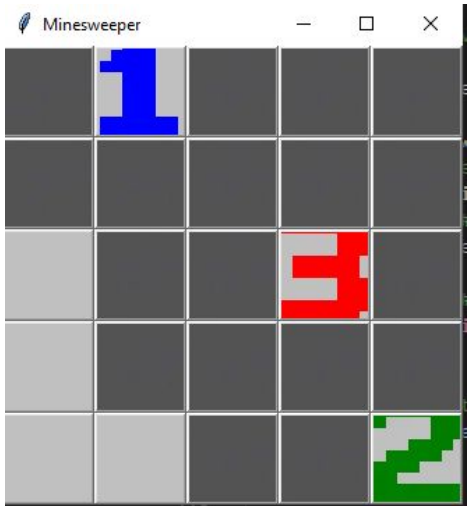

Minesweeper

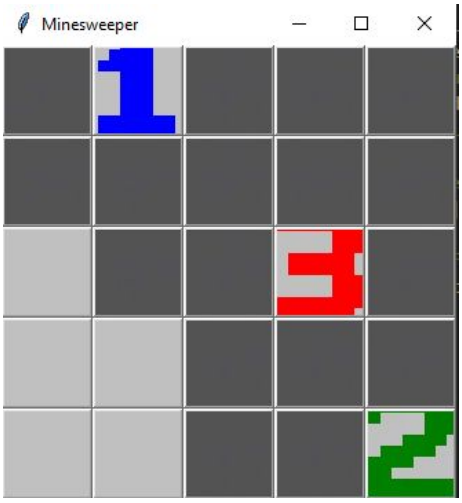

Minesweeper

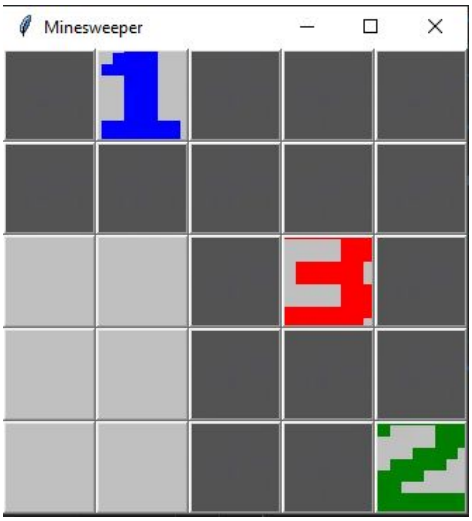

Minesweeper

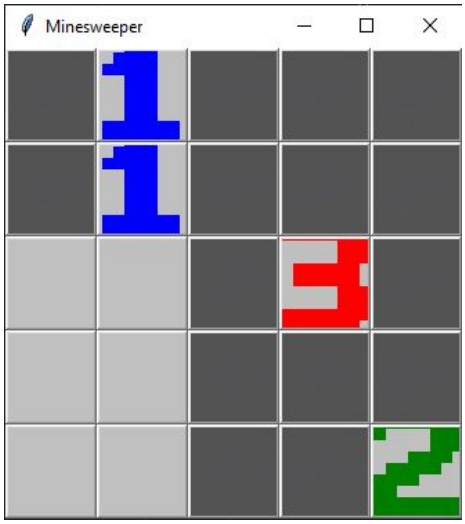

Minesweeper

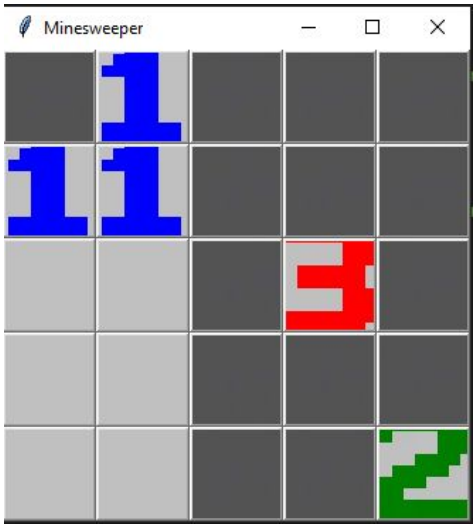

Minesweeper

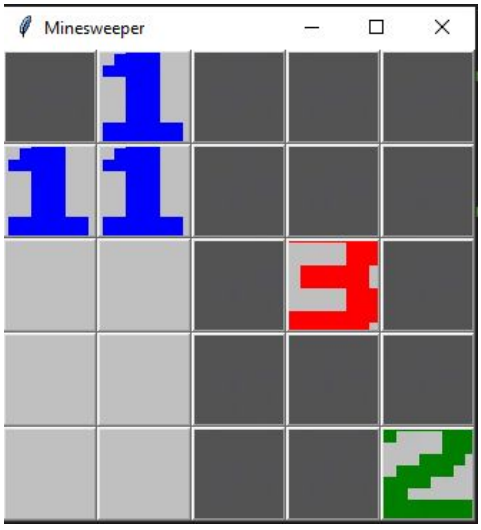

Minesweeper

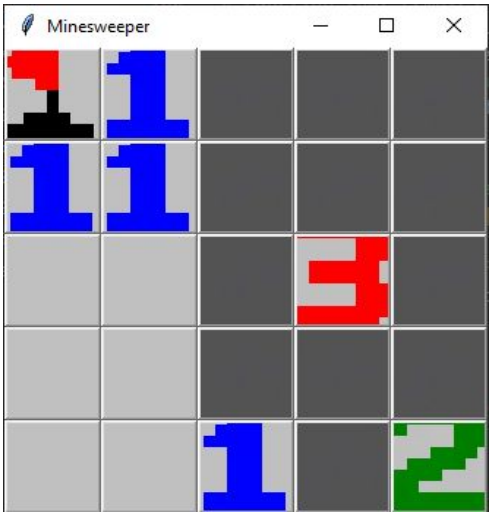

Minesweeper

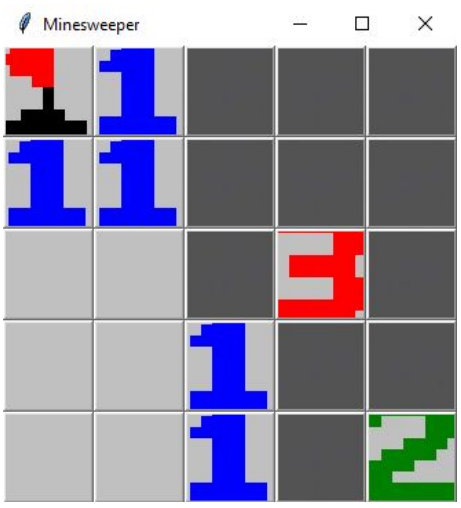

Minesweeper

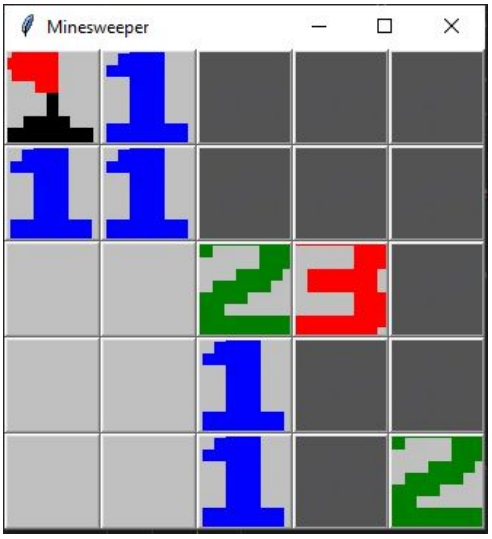

Minesweeper

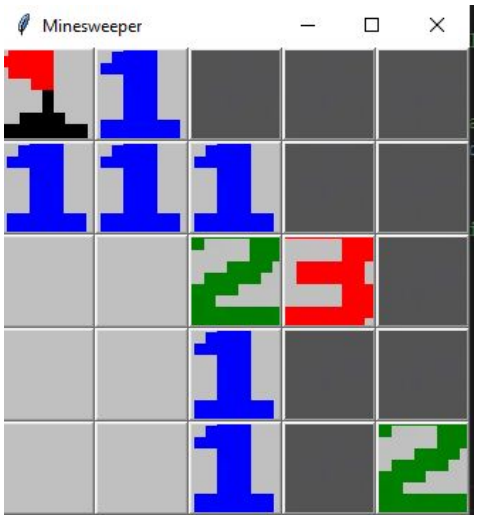

Minesweeper

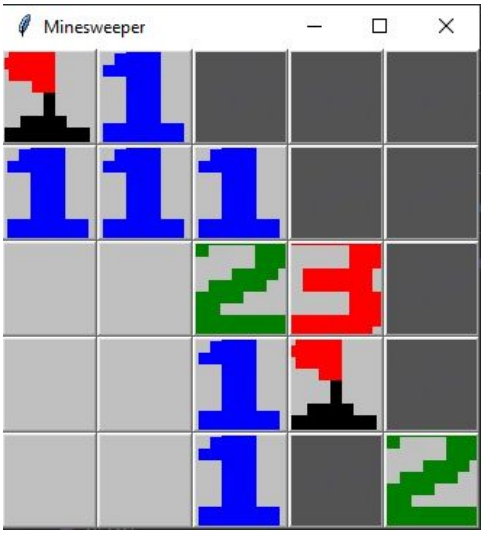

Minesweeper

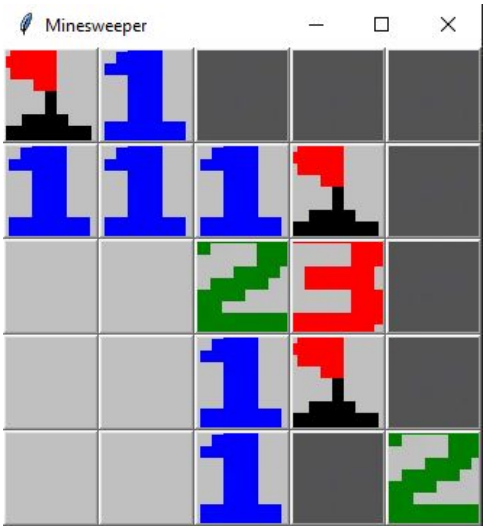

Minesweeper

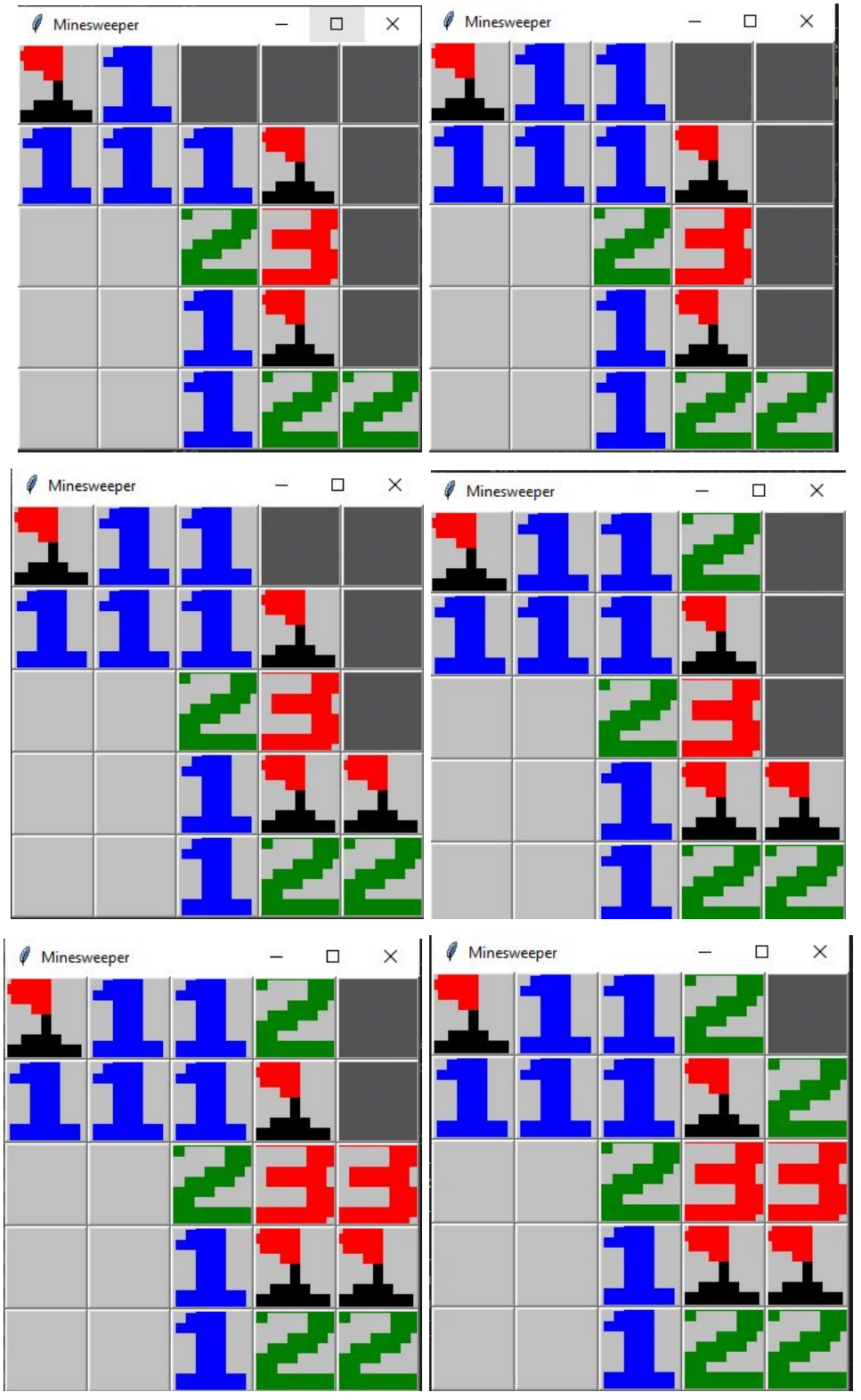

Minesweeper


Minesweeper


Minesweeper


Minesweeper


Minesweeper


Minesweeper
Minesweeper
Minesweeper
Minesweeper
Minesweeper
Minesweeper

We won!! :)

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?

In order to generate the Average Final Score vs. Mine Density plot, we ran each of the two agents 15 times with densities ranging from 0.0 to 0.5 on a board of dimension 5,

Figure: Average Final Score vs Mine Density for Basic Agent

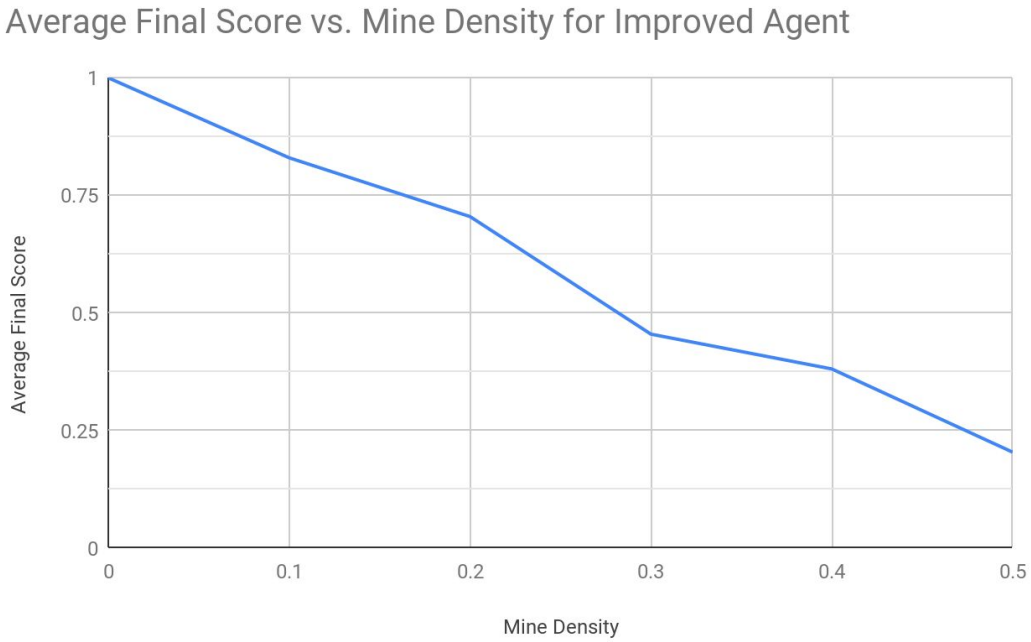


Figure: Average Final Score vs Mine Density for Improved Agent (Agent007)

Based on the two plots, as the mine density increases the average final score decreases. The average final score of 1 means that all the mines have been successfully identified. We can observe that the average final score decreases as the number of mines or mine density increases within the given board. This agrees with our intuition as the increase in mine density in turn increases the difficulty of the game, which makes it harder for both agents to safely identify a mine. Furthermore, minesweeper becomes ‘hard’ when the agent's average final score drops significantly. In these instances, the agents are left with unsolvable clues in which they must rely on luck when randomly selecting cells. This can be seen in both plots when the mine density becomes greater than or equal to 0.3.

According to the plots, we can observe that the improved agent algorithm beats the basic agent algorithm for mine densities greater than or equal to 0.1. This can be due to the fact that the basic agent chooses more random cells for its next move while the improved agent continues to represent relationships between cells and work through the various constraint equations. The basic agent algorithm may be better than the improved agent when the mine density is less than or equal to 0.1 and the size of the game board is fairly large. This is because the basic agent is able to make simple deductions that do not require much representation of constraints between cells.

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?

Implementing the improved agent (Agent007) we did encounter some time and space constraints. Since our knowledge base is represented as a list in python, when the agent removes a constraint equation from the knowledge base, it will go through the entire knowledge base to remove the specified tile. Hence, a time consuming operation. For a very large grid space and a large knowledge base (higher mine density), it will increase our time and space complexity. Also, on instances in which we could not gather any information from what we know, we would choose a random cell for our next move, which proves to be a costly operation. Lastly, our improved agent utilizes the board in which each cell is represented as a tile object. Thus, in order to create this board representation, we would have to iterate through each cell which could be a computationally intensive operation for large grid sizes.

Thus, our space complexity for this algorithm is roughly O(n^2), while our time complexity is O(n^3).

Our problem mentioned above is an implementation specific constraints and can be removed/resolved by changing the way we store our constraint equations to the knowledge base. As an improvement to the program implementation, we could represent the knowledge base as a dictionary instead of a list. This is because dictionaries hold key:value pairs in which we can use a key to index into a specific equation more efficiently.

Improvements

Consider augmenting your program's knowledge in the following way - tell the agent in advance how many mines there are in the environment. How can this information be modeled and included in your program, and used to inform action? How can you use this information to effectively improve the performance of your program, particularly in terms of the number of mines it can effectively solve? Re-generate the plot of mine density vs expected final score for your algorithm, when utilizing this extra information.

The information about the total number of mines in the environment is modelled in such a way that if the number of cells flagged becomes equal to the total mines, our improved agent will blindly open all the closed tiles without any inferences. This check can be added in various stages in the code to make the agent solve faster. With our basic agent, the agent makes the check once all obvious constraints have been checked i.e, if the number of mines present in the board becomes equal to the number of uncovered mines, the agent uncovers all other tiles and exits game.

For our improved agent (Agent007), in the subset solver phase the agent makes the check after solving the subsets at every iteration. Similarly like suggested for the basic agent if the count found is equal, the agent uncovers all the cells without inference and cuts down on the run time of the algorithm significantly. This information helps the agents be a lot more efficient. It saves time checking the remaining tiles. Hence time to play the game decreases by a lot. The improved agent will have extra information which makes it stop evaluating the equations once the total mines are opened or flagged.

Figure: Average Final Score vs Mine Density for Improved Agent knowing Number of Mines (Agent007)

Contributions

Krupal
- Environment
- GUI
- Basic Agent
- Improved Agent
- Report

Zain
- Environment
- Basic Agent
- Improved Agent
- Report

Suraj
- Environment
- Improved Agent
- Report

Melanie
- Environment
- Improved Agent