Inference- Informed Action: MineSweeper

Jerry Yang

March 2021

1 Abstract

In this project, I worked on two different AIs that attempts to beat the game of Minesweeper. The board is essentially a grid of cells and a number of the cells hide 'mines'. The agent will try to mark all of the mines by collecting clues and information and infer the safe and dangerous cells. Unlike ordinary minesweeper however, I will be implementing a 'keep going' rule. In the case the agent sets off a mine, he will keep playing, and that mine will be usable as information moving forward. This project is done in Python and Pygames is used for the GUI. Numpy and matplotlib were used for the graphs. I will also have some brief pseudo code in the report. Note that the pseudo code is only meant to assist in the report and is not going to reflect the code as it is in the python files for simplicity. The algorithms below and of the minesweeper game and AI in general can be found in the minesweepai.py file. The minesweeperui.py is used mainly for GUI purposes and does not encompass any of the algorithms below.

Figure 1: Random Snapshot of a given gamestate

2 Basic and Improved Agent: A Brief Overview

Two agents were created to play this game of minesweeper. In this report, we will refer to them as the Basic Agent, and the Improved Agent. The former simply uses the following logic. For a given cell, if the clue minus the cells revealed is the number of hidden neighbors, every hidden cell is a mine. If the number of safe neighbors minus the the number of revealed safe neighbors is the number of hidden neighbors, every hidden neighbor is safe. Otherwise, choose a random cell to uncover. The Improved agent keeps the two guaranteed basic cases but adds constraint satisfaction to the algorithm. It will project possible solutions that satisfy the constraints, and choose cells based on that. I will go into more details on this in the later sections. For now, here are two brief pseudo codes for the two agents.

Algorithm 2: improvedAgent(*board*)

```
confirmation = False     // Tells us whether or not a guaranteed logic case is applied
for cell in board:       // For each not hidden cell in board
    if cell.clue - cell.mineNeighbors == cell.hiddenNeighbors:      // Basic Logic 1
        confirmation = True
        updateBoard(cellNeighbors,mined)     // Update cell Neighbors as mined
for cell in board:       // For each not hidden cell
    if 8 - cell.clue - cell.safeNeighbors == cell.hiddenNeighbors:      // Basic Logic 2
        confirmation = True
        updateBoard(cellNeighbors,safe)    // update cell neighbors as safe
if confirmation == False:     // If no logic can be used, generate CSP solutions
    boolean, solution = generateValidSolutions()     // generate possible solution
 that satisfies constraints
    if boolean == false:     // If no solutions can be found, choose random cell
        updateRandomCell
    else:     // otherwise, update a safe cell based on solution
        updateSafeCell from solution
```

Algorithm 1: basicAgent(*board*)

```
confirmation = False     // Tells us whether or not a guaranteed logic case is applied
for cell in board:       // For each not hidden cell in board
    if cell.clue - cell.mineNeighbors == cell.hiddenNeighbors:      // Basic Logic 1
        confirmation = True
        updateBoard(cellNeighbors,mined)     // Update cell Neighbors as mined
for cell in board:       // For each not hidden cell
    if 8 - cell.clue - cell.safeNeighbors == cell.hiddenNeighbors:      // Basic Logic 2
        confirmation = True
        updateBoard(cellNeighbors,safe)    // update cell neighbors as safe
if confirmation == False:     // If no logic can be used, choose random cell
    updateRandomCell
```

3 Board Representation

The board is represented as a simple 2D array of cells. These cells can be initialized with Cell(x,y) with x and y being the coordinates. The Cells themselves then carry many attributes, including information that the agents can use to make inferences. The Cells of course carry their x and y coordinates. They then also carry the state property and a mine property. The state property is their current state on the board and is a string. In this case, it can be 'covered', 'clear', or 'mined'. If the state is 'covered', then it is one of the cells not yet explored. If it is 'mined', then the agent has declared that there is definitely a mine hiding under the cell. If the state is 'clear' and the mine property of the cell is false, it indicates the cell has been explored and will unveil the clue on the number of mines surrounding it.

The Cell then has what I will call knowledge base properties. These properties are not only based off the current state, but they can only be accessed or drawn on if the cell state has been cleared. In the minesweepai.py file, you will see these properties as *neisafe*, *neiHidden*, *neiMine*, *neiBoom*, and *safety*. Somewhat self explanatory, but they represent number of safe neighbors, number of hidden neighbors, number of neighbors that have been marked as a mine, and number neighbors that went boom(Cell with mine was uncovered). The functions on how the information is stored are all in the code, but this information is then stored as a property of the Cell. Likewise, this remains updated, so flipping over a mine will affect the property 'number of neighbors that went boom' around them.

The inferred relationships between Cells is then enforced through the use of a checkConstraint function, which is consistently used to check the validity of projected solutions. In general, this function checks that for any given cleared cell in the board, the number of mine neighbors(marked as mine) and the number of neighbors that went boom are not greater than the clue (actual mine neighbors). This works since those properties are updated as the board changes.

4 Inference: Processing New Information

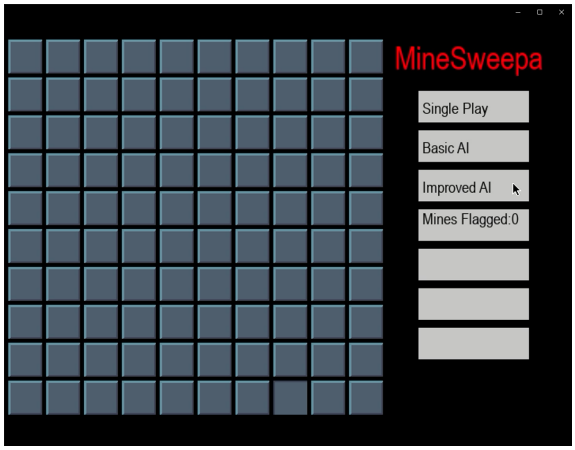
Once a new clue opens up, the improved agent will update it's knowledge base (the cell properties that the agent has access to) through functions of updating *neisafe*, *neiHidden*, etc. Since the board itself has changed, many of the cell clues will also have changed so the agent will re-examine all of the hints prior to making its move. The agent will actually reexamine every single clear cell to update the information, so this is actually somewhat inefficient, but it does reinforce the idea that the cell information that the agent has access to is up to date. The current improved agent does not deduce everything it can from a given clue before continuing. This becomes evident in the case where there are multiple mine placements that satisfy a series of constraints. Our Improved Agent will generate possible solutions that generate that constraint until it finds an optimal solution. However, it does not compare optimal solutions, and returns the first optimal solution it finds. With that in mind, our agent can be improved as it is fairly safe to assume that among those optimal solutions, only one can truly be correct as there is only one final right answer(each mine can only be placed in one cell and that doesn't change throughout the game).

5 Decisions: Which Cell to Search Next

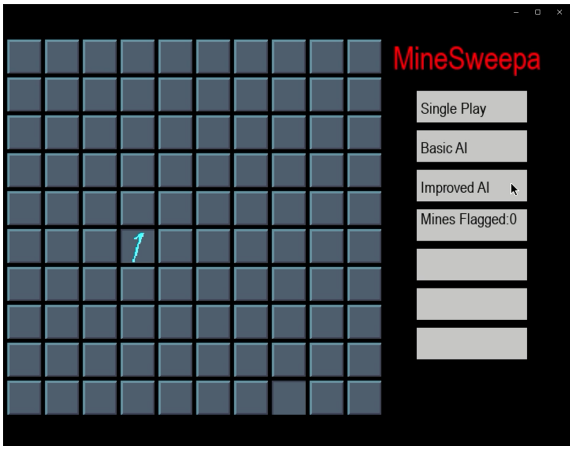
Given a current state and knowledge of the board, the Improved Agent does the following. It first searches for any basic logic cases that can be used. These logic cases are the guaranteed cases of the Basic Agent. The agent will then generate solutions that do not break the constraint of the current board. The agent will then choose a cell that is considered safe in the generated optimal solution. The chosen cell is also going to be a cell that is close to one of the cleared/mined cells. This is because our generated solution does not really tell us anything about covered cells far away from our clues. In the case that our agent is not able to find any optimal solutions, the agent selects a cell at random, just like the Basic Agent.

6 Performance: Smart Decisions

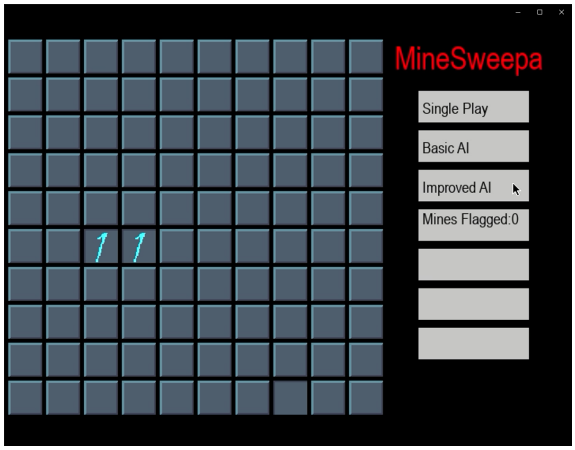
Below is a play-by-play progression of a sample game of our improved agent. Note that the agent will apply things like basic logic to the whole board so when multiple conclusions can be drawn, they will be considered as a single move. The board size used is a 10x10 board with a total of 20 mines so a mine density of about 0.2.



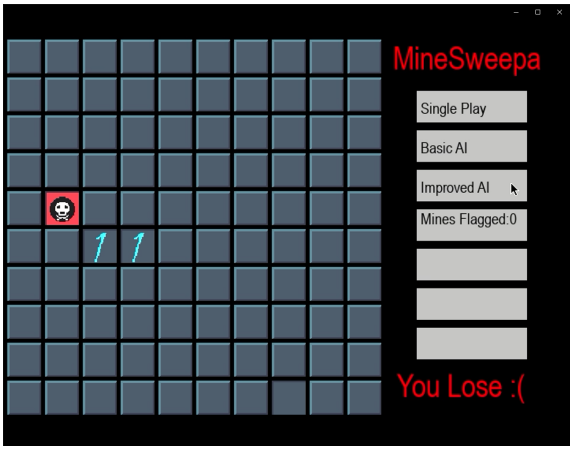
move 1



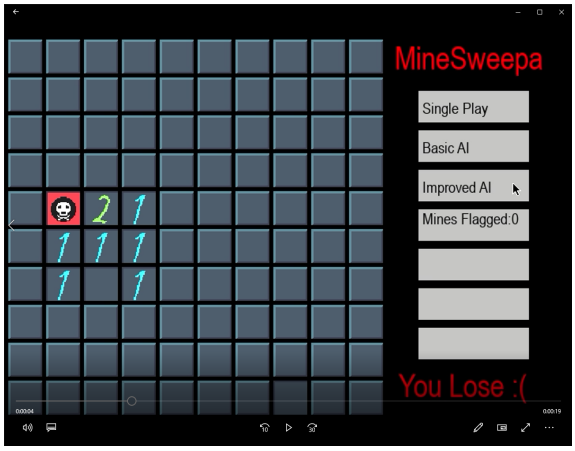
move 2



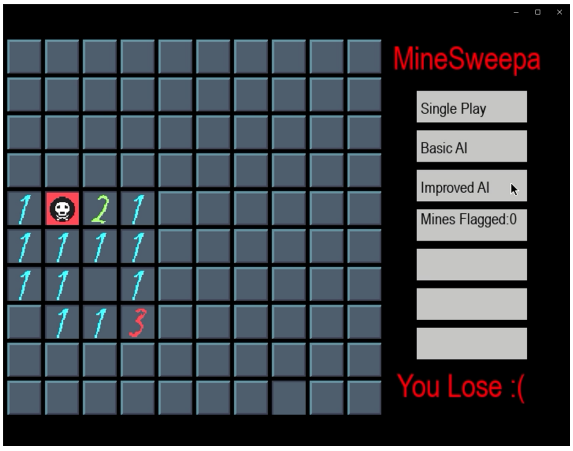
move 3



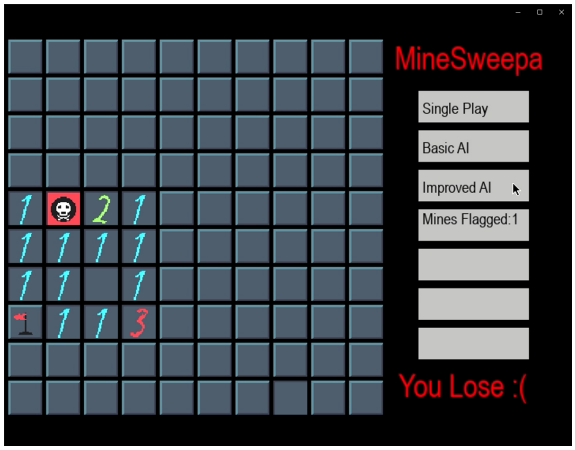
move 4



move 5



move 6



move 7

move 8

move 9

move 10

move 11

move 12

move 13

move 14

move 15

move 16

move 17

move 18

move 19

move 20

move 21

move 22

move 23

move 24

move 25

move 26

move 27

move 28

move 29

I wouldn't say that any where in our program, the program makes a mistake. However, the program does make decisions that I do not agree with. This is evident in move 4 and is a recurring source of the agent blowing up mines. When there is only a single clue, and most of the surroundings are not yet cleared, the agent will generate a possible solution that satisfies constraints, and choose a cell that it deems as safe. However, with the lack of clues, this solution is nothing more than a full on guess. In specific cases like this one, it would actually be smarter to choose a cell not near the clue in order to retrieve information on the surrounding cells which perhaps can tell us with more about the cells surrounding our original cell. However, at the same time, this allowed the agent to perform a series of moves that surprised me as well, which is the series of moves following the marking of the 3 vertical mines on move 12. Through the process of generating solutions that satisfy the constraints and choosing the nearby cells that are deemed safe, it completely surrounded the three vertical mines, and spread out from there, giving itself more clues to work off of.

7 Performance: Basic vs. Improved

Below is the table and plot of the function of mine density and average final score for both of the agents.. The board used is a simple 10x10 board and the mines density increases by 0.1 each time. Each data point is based off of 150 trials.

Mine Density	Mines Identified(Basic)	Mines Identified(Improved)	Total Mines	Total Score %(Basic)	Total Score %(Improved)
0.1	1380	1393	1500	92.00%	92.87%
0.2	2265	2424	3000	75.50%	80.80%
0.3	2632	3091	4500	58.49%	68.69%
0.4	2933	3675	6000	48.88%	61.25%
0.5	3390	4089	7500	45.20%	54.52%
0.6	3846	4589	9000	42.73%	50.99%
0.7	4162	4806	10500	39.64%	45.77%
0.8	4278	4702	12000	35.65%	39.18%
0.9	3244	3628	13500	24.03%	26.87%

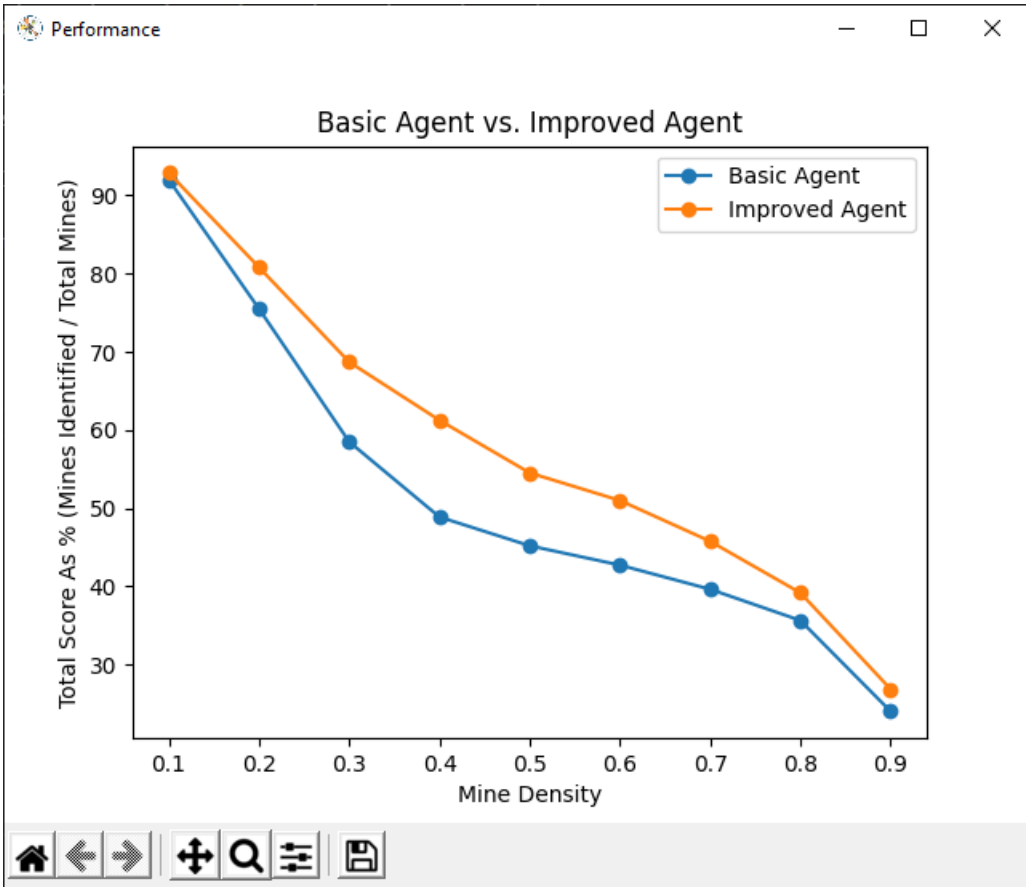


Figure 2: Basic vs Improved Total Performance(150 Trials Per)

The graph agrees with our basic intuition. As the mine density increases, it becomes increasingly more difficult to win or identify all mines correctly. This also follows basic logic as if every piece on the board is a mine, then the agent has no clues to work off of and would thus identify zero percent of the overall mines. The more mines, there are, the less clues the agent has to work off of. Minesweeper becomes harder as the mine density increases. From the data, our Improved Agent beats the simple algorithm in all mine densities. As the mine density converges to 0 or one however, the two agents' performance does converge which makes sense as that means it's a board of all or no mines. It's quite clear that at a mine density of about 0.3 to 0.6, there is the biggest difference in performance between the two agents. Because our Improved agent, is essentially the Basic Agent combined with generating CSP solutions, it is very rare that that the Basic Agent would actually outperform the Improved Agent. This is evident as shown in the graph. The Basic Agent never once outperformed the Improved Agent on average. However, this is not to say the Improved Agent is always better. I alluded to this weakness when talking about the programming making a decision that I do not agree with. In certain cases, it is better to select options farther away from the current revealed clues in order to get more information on the cells around the clues. Since the Basic Agent chooses a cell at random, it's actually possible for it to coincidentally pull this off. However, as long as our Improved Agent can generate a solution that satisfies constraints, it will never attempt a move like this. It's quite evident that the Improved agent can consistently work out things out things the Basic Agent cannot it's consistently above it in the graph. This lies in the fact rather than

choosing the next cell at random, the Improved Agent generates possible solutions that satisfy the constraints, which can actually lead to additional cases where the basic logic can be used.

8 Efficiency

Our Improved Agent is certainly not the best when it comes to efficiency. This was quite evident as we were doing 150 runs per data point when creating the graphs. The Basic Agent would take significantly less than time than the Improved. We believe the problem lies more so in implementation specific constraints rather than problem specific. The first of the issues is the way we make sure our generated solution do not break constraints. Rather than generating the solutions we need, we generate all possible mine placements based on the current state of the board, and then filter out the moves that break the constraints. This algorithm could easily be improved if we did not generate the excess possible mine placements in the first place. Another part of the implementation process we could improve on is the checkConstraint's function itself. Every time we run this, the entire board gets checked for any constraint breaks, despite the fact only a small handful of cells actually changed. This could be improved by simply checking the constraints of the areas around the changed cells, as we already know other parts of the board satisfy the constraints.

9 Bonus: Global Information and Better Decisions

Knowing how many mines are on the board in advance assists us with our endgame. We can add a logical case that if the covered cells remaining + the number of cells we've identified as a mine(or blown up) is equal to the final number of cells, we can assume the rest of the covered cells are mines and mark them as such. Although not the most efficient approach, we just added it to one of the AI's process' to check the board for this case every run. This certainly helps, and as expected is more noticeable at higher density's as shown in the graphs below. It can be seen that that the Global Agent performs consistently better than the regular improved agent, but the difference increases as the mine density goes up.

Because of how we structured our Improved Agent, it only selects a covered cell at random when there are no clues with covered cells next to it remaining. In other words, in general, after the first few moves of the game, it is actually quite rare to see it attempt to a complete random guess. Because of that, improving the random portion would not really bring much of an improvement to our algorithm as it already takes into account random guessing by guessing based on the constraints. However, this isn't to say the selection mechanism cannot be improved. Currently, it makes very risky moves. For example, if a 4 is revealed, and only 5 around it are still covered with none being mines., it will create a solution where 4 of those are mines, and choose a cell there. It would be smarter to avoid this area and gather more clues on it. As a result, we can alter our guess to exclude cases like these. However, it's also possible that the generated solution is indeed the correct one. In a case like this, avoiding those cells, although it seemed risky, would actually be the wrong move. With this in mind, we added a simulated annealing aspect to the algorithm. We essentially altered the Improved Agent so that when choosing the cell from the generated optimal solution, it will check whether or not that cell is *risky* as with our previous example. If it deems the move risky, there will be a 20

percent chance that it in fact will not go with that move, and move on to a different cell in the projected solution, or reproject a different optimal solution. The data results are as expected, but there were cases at 0.1, 0.8, and 0.9 where it actually performed worse on average than the regular Improved Agent. However, given that at extreme densities, all agents will converge, this fact is not outside expectations. Outside of those points, the new Improved Agent did in fact do better than the regular Improved consistently, with the greatest difference at a mine density of 0.5.

Below is the data and graphs alluded to in this section. Each data point is based off of 100 runs.

Mine Density	Mines Identified(Global)	Mines Identified(Better Decisions)	Total Mines	Total Score %(Basic)	Total Score %(Improved)
0.1	921	919	1000	92.10%	91.90%
0.2	1597	1631	2000	79.85%	81.55%
0.3	2109	2078	3000	70.30%	69.27%
0.4	2449	2474	4000	61.23%	61.85%
0.5	2762	2824	5000	55.24%	56.48%
0.6	3160	3085	6000	52.67%	51.42%
0.7	3348	3255	7000	47.83%	46.50%
0.8	3318	3092	8000	41.48%	38.65%
0.9	2978	2383	9000	33.09%	26.48%

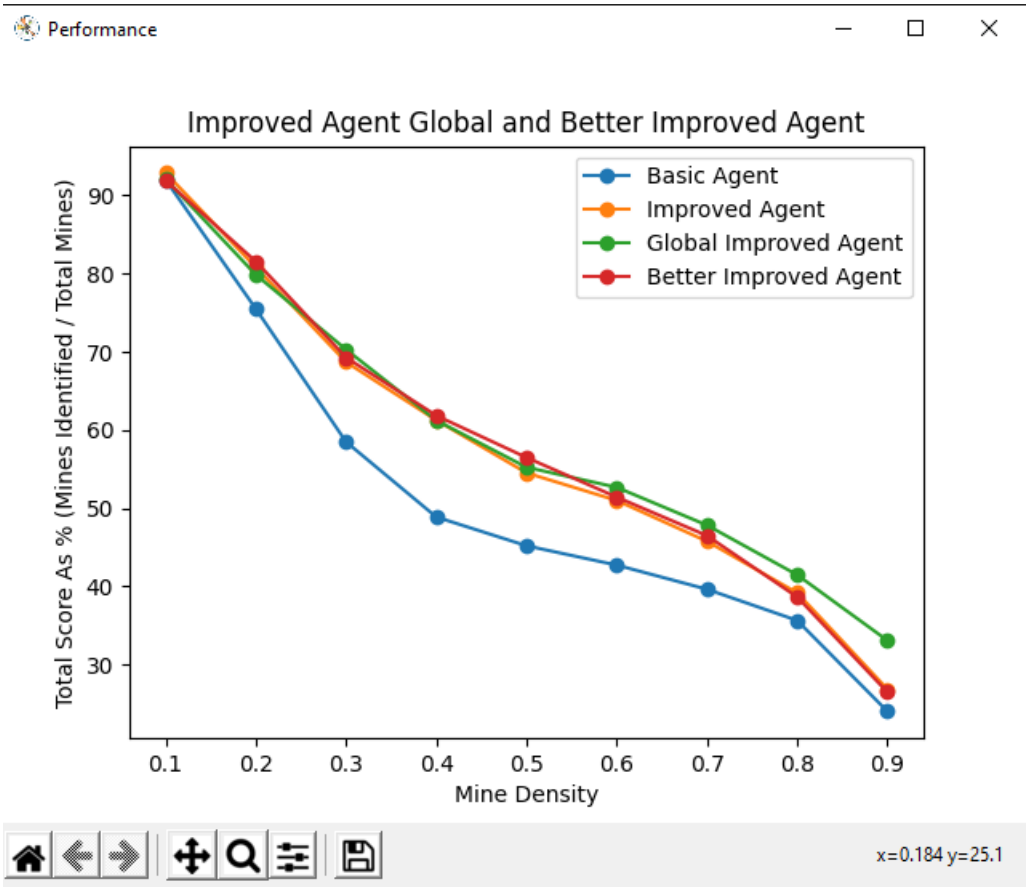


Figure 3: Global Improved Agent and Better Improved Agent Results

10 Additional notes

How to run Code:
Simply run minesweeperui.py to play! If you want an AI to make a move, click Basic AI or Improved AI at the right.