CS440 Project 2: Minesweeper

Parth Patel (psp116)

March 2021

Introduction

Implementation of basic and improved agents to solve minesweeper with a given board of size d with n mines.

Language used: *python*
Libraries required: *random*, *numpy*, *math*, *matplotlib*

Representation

- **Board representation**

  The board was generated using *numpy*, specifically using the method $full$. The method *make_board* handles the creation of the board and returns a board of given size *size* and number of mines *mines*.

  ```
  def make_board(size, mines)
  ```

  Initially, the board is $size * size$ 2D matrix that contains all 0s. Then using $rand.randint(0, size - 1)$ coordinates of a cell that isn't already a mine are calculated and filled with a value of -1 which represents a mine.

  The next step iterates through the entire board and assigns a clue value for each non-mine cell where the clue value is the numbers of mines around it. The following shows an example of how a board looks: M for mine and a *number* represents a non-mine with a clue.

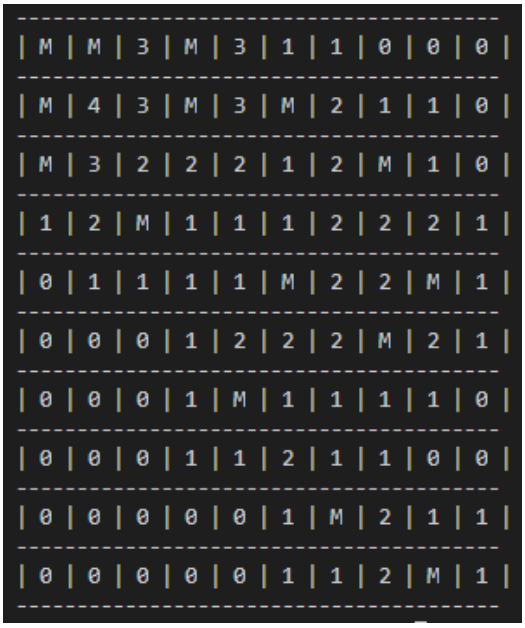

Figure 1: board representation

- **Clue representation**

  The clue representation was probably the most crucial part of this project since it influenced everything from implementation to efficiency. To represent clue I went with an equations approach.

  Each clue is a *tuple* of a *list* and a *number*, the *list* contains all the *variables/cell coordinates* which are part of that equation and the *number* represents the value of the equation(essentially the clue value).

  Example:

  ```
  ([(0,0), (0,1), (2,0)], 2)
  ```

- **Knowledge base**

  The knowledge is essentially a *list* of *clues*.

  ```
  knowledge base: [ ([(3, 0)], 1), ([(4, 1), (3, 0), (4, 0)], 3) ]
  ```

- **Cell info**

  This data structure is represented using a *dictionary* and what it essentially does is, it stores information the agent has about the cells. It is an extension to the knowledge base. This information could have either been found while querying a cell or through inference. To clarify, it only stores whether a cell is a mine or a non-mine. Moreover, this structure also helps update the knowledge base with new information and eliminate unnecessary equations. In the following, -1 represents a mine while 0 represents a non-mine.

  ```
  Cell info: {(2, 0): -1, (0, 1): -1, (1, 1): 0, (1, 0): 0}
  ```

- **Next cell**

  This data structure is a stack that hold non-mine cells that can be queried next. If this stack is empty, a cell is selected at random.

  ```
  Next cells: [(0, 0), (2, 1)]
  ```

Inference

- **Basic Agent**

  The basic was built as specified. The basic flow of the inference was as follows:
  figure out cell to query > query the cell > build the equation based on that clue > add to knowledge base > check if the equation has all mines or all non-mines > update knowledge base > repeat

  The basic agent is able to perform adequately for the most part. It starts with figuring out which cell to query and, as discussed above if $cell\_info$ is empty, it randomly chooses one cell to query. This marks the end of the *decision* step. Then, we query the cell and if a clue is found, we build an equation and append it to the knowledge base. And if a mine is found, we update our $cell\_info$ with a new mine information. Now on to the $inference$ step, we iterate through the entire

knowledge based and do this:

1. **Boil down the equation**

   In this step, we iterate through the variables of the equations and if any variable's value is known to us, we update the equation accordingly. There is also a $flag$ that we turn $true$. This $flag$ decides whether or not the agent will keep this equation for the next iteration of the inference step. For example:
   $A+B+C+D=3$ and we know, $A=-1$ (mine) and $C=0$ (non-mine). Then, the equation will be updated to: $B+D=3-1+0$ and finally: $B+D=2$

2. **Check for an all-mine equation**

   For this step, the equation's value and the number of variables are checked. If they are equal to each other, we know this equation is an all-mine equations. In other words, the equation's variables are all mines. This allows us to further expand our $cell\_info$ to include these variables. All the mines found here, contribute to the *score* of the game. Lastly, the flag that we turned on in the previous step will be turned $false$ since we have completely utilized the equation. For example:
   $A+B+C=3$ then, A, B and C are all mines.

3. **Check for a no-mine equation**

   For this step, the equation's value is checked and if it is a 0, we can deduce that it represents all non-mine and safe cells. With this new information of non-mines, we update our $cell\_info$ and append the variables to $next\_cell$ stack. And similar to the previous step, we turn the $flag$ false since the equation has been completely utilized.

4. **Filtering**

   Lastly, to clear the clutter and boost efficiency a tad bit, we clear out the unnecessary equations from the $knowledge\_base$ with the help of out $flag$. Intuitively, if the flag is $true$ then we keep the equation otherwise we discard it.

Although the basic agent is not able to infer extensively, it is still able to perform fairly good.

Here is a graph of the basic agent solving a board of dimension 20 with varying mine density (number of mines/size of board) vs average score. Each mine density goes through 15 runs and the scores are averaged:

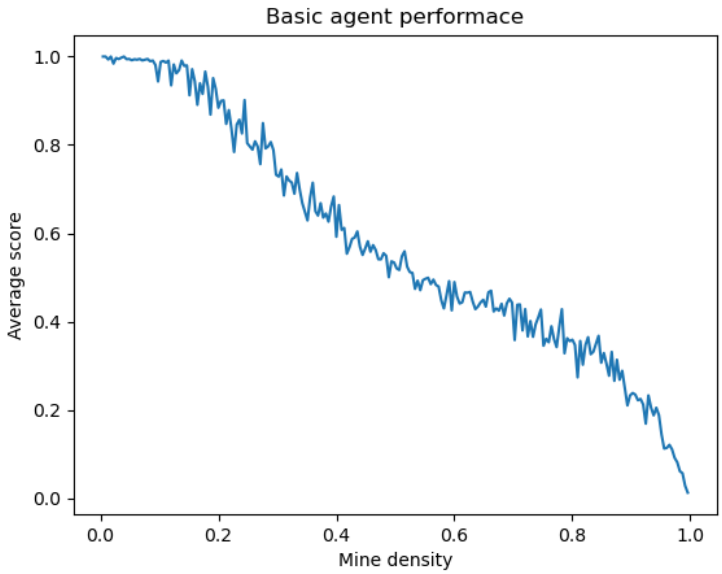

Figure 2: Basic agent

- **Improved Agent**

  The improved agent was built as an extension to the basic agent. It utilizes all the inference procedures of basic agent and adds its own ability to solve subsets to it. The acronym that I came up with is ICE (Intertwined and Common Equations solver). We add this step over the $inference$ step of basic agent:

  1. **Subset solver**

     We add the ability to solve subsets with a method called *subset_solver*. This method takes in two equations and turns both of their variables into sets. Followed by this, it compares if one is a subset of the other and if it is, it subtracts one from the other and yields a

new clue or equation. An example of how it works:
Given $A + B + C = 2$ and $B + C = 1$, *subset_solver* will check that B, C is a subset of A, B, C and proceed to subtract the subset from the set to yield another equation/clue. In this case, it will deduce $A = 1$ (mine).

ICE is able to effectively use the clue found from *subset_solver* and the other inference procedures. But does it deduce everything? No. This is a limitation of my representation of clues. *subset_solver* is only able to solve subsets and there are going to be equations where maybe one or more variables overlap which have the potential to yield more clues. Regardless, ICE is most certainly able to improve on basic agent.

Here is a graph of the improved agent (ICE) compared with the basic agent. Both solve a board of dimension 20 with varying mine density (number of mines/size of board) vs average score. Each mine density goes through 15 runs and the scores are averaged:

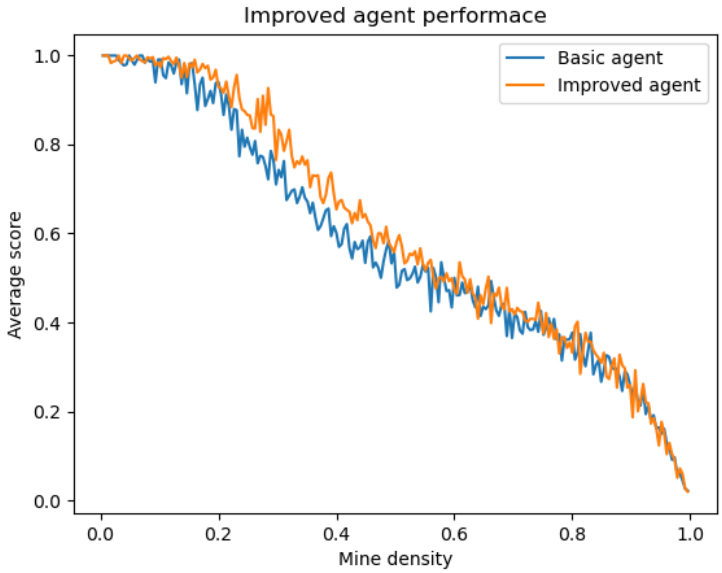

Figure 3: Improved agent (ICE)

- **Performance**

The graph does agree with my intuition but I expected even better improvements with ICE. Both the agents perform similar for each end of the spectrum of mine density. It is actually in the middle section that we truly see the advantage of improved agent and it makes sense since we have a good balance of mines and non-mines. For both the agents, the game becomes hard when there is an abundance of mines (approx mine density ¿ 0.5). Conclusively, the improved agent either does better or similar to the basic agent.

Decisions

1. **Default decision mechanism**

   As discussed before, the default decision mechanism is as follows: if *next_cell* stack is empty, we choose a cell at random that is not already known.

2. **Better decision mechanism**

   This mechanism involves three options: if *next_cell* stack is empty, we choose an unknown cell at random from the knowledge base. This is given that the knowledge is not empty and in a case where it is empty, we choose a cell at random. This mechanism greatly improves both the agents.

   Here are some graphs of improved agent (ICE) and basic agent with the better selection mechanism. Both solve a board of dimension 20 with varying mine density (number of mines/size of board) vs average score. Each mine density goes through 15 runs and the scores are averaged. The last graph compares all four agents with a dimension of 20 and 30 runs each. Also, $v2$ at the end indicates the usage of the better selection mechanism:

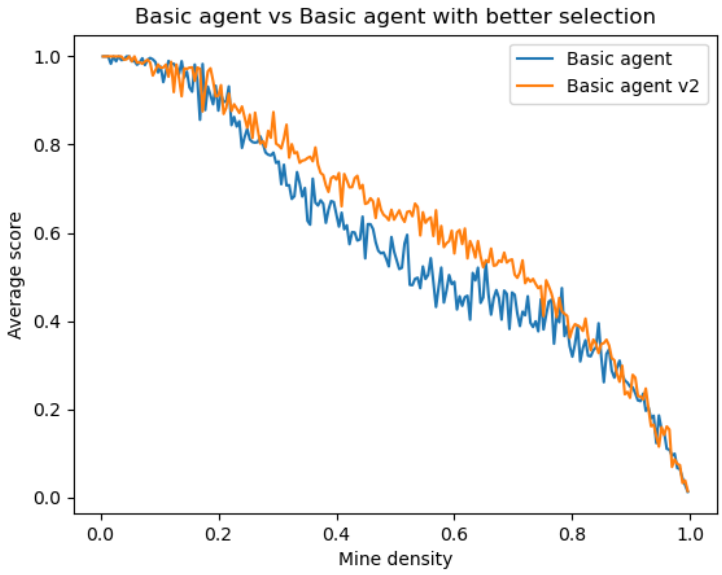

Figure 4: Basic agent with better selection

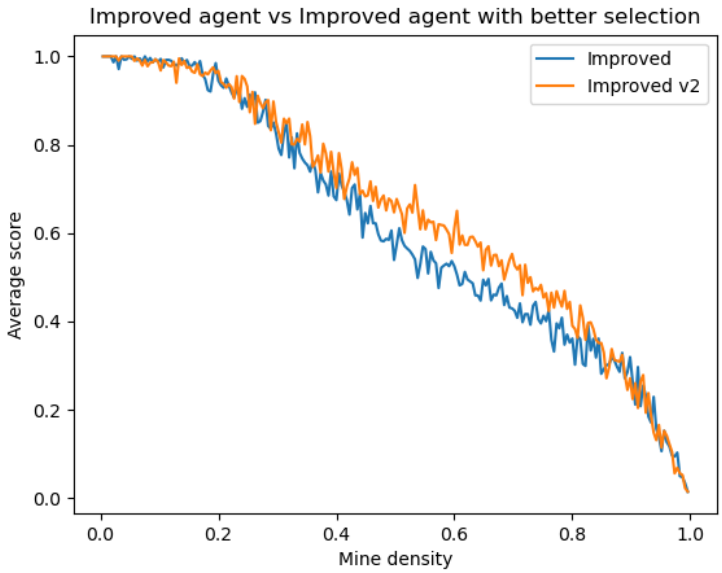

Figure 5: Improved agent (ICE) with better selection

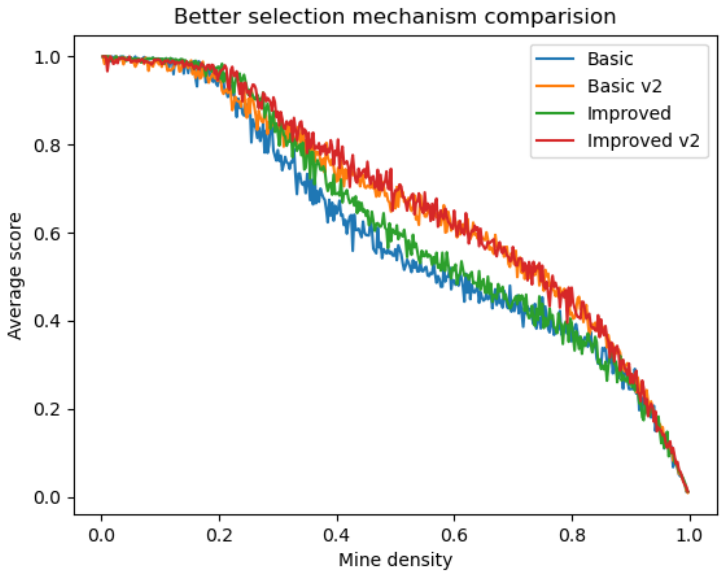

Figure 6: All the agents compared

Performance

1. **Performance of improved agent (ICE) with better selection**

   A play-by-play execution of ICE agent with better selection on a board of size 4 and 7 mines. Score = 5/7:

```
--------------------------------------------------
knowledge base: []
Known safe cells: []
Cell info: {}
Queried cell: (3, 3)
--------------------------------------------------
knowledge base: []
Known safe cells: []
Cell info: {(3, 3): -1}
Queried cell: (0, 2)
--------------------------------------------------
```

```
knowledge base: [([(1, 2), (0, 1), (0, 3), (1, 3), (1, 1)], 1)]
Known safe cells: []
Cell info: {(3, 3): -1, (0, 2): 0}
Queried cell: (0, 3)
----------------------------------------------------
knowledge base: [([(1, 2), (0, 1), (1, 3), (1, 1)], 1)]
Known safe cells: [(1, 3), (1, 2)]
Cell info: {(3, 3): -1, (0, 2): 0, (0, 3): 0, (1, 3): 0, (1, 2): 0}
Queried cell: (1, 2)
----------------------------------------------------
knowledge base: [([(0, 1), (1, 1)], 1), ([(2, 2), (1, 1),
    (0, 1), (2, 3), (2, 1)], 3)]
Known safe cells: [(1, 3)]
Cell info: {(3, 3): -1, (0, 2): 0, (0, 3): 0, (1, 3): 0, (1, 2): 0}
Queried cell: (1, 3)
----------------------------------------------------
knowledge base: [([(0, 1), (1, 1)], 1),
    ([(2, 2), (1, 1), (0, 1), (2, 3), (2, 1)], 3),
    ([(2, 3), (2, 2)], 1), ([(2, 3), (2, 1), (2, 2)], 2)]
Known safe cells: []
Cell info: {(3, 3): -1, (0, 2): 0, (0, 3): 0, (1, 3): 0, (1, 2): 0}
Queried cell: (2, 3)
----------------------------------------------------
knowledge base: [([(0, 1), (1, 1)], 1),
    ([(2, 2), (1, 1), (0, 1), (2, 1)], 3), ([(0, 1), (1, 1)], 1)]
Known safe cells: [(3, 2)]
Cell info: {(3, 3): -1, (0, 2): 0, (0, 3): 0, (1, 3): 0,
    (1, 2): 0, (2, 3): 0, (2, 2): -1, (2, 1): -1, (3, 2): 0}
Queried cell: (3, 2)
----------------------------------------------------
knowledge base: [([(0, 1), (1, 1)], 1),
    ([(1, 1), (0, 1)], 1), ([(0, 1), (1, 1)], 1)]
Known safe cells: [(3, 1)]
Cell info: {(3, 3): -1, (0, 2): 0, (0, 3): 0, (1, 3): 0,
    (1, 2): 0, (2, 3): 0, (2, 2): -1, (2, 1): -1, (3, 2): 0, (3, 1): 0}
Queried cell: (3, 1)
----------------------------------------------------
knowledge base: [([(0, 1), (1, 1)], 1),
```

```
    ([(1, 1), (0, 1)], 1), ([(0, 1), (1, 1)], 1)]
Known safe cells: []
Cell info: {(3, 3): -1, (0, 2): 0, (0, 3): 0, (1, 3): 0, (1, 2): 0,
    (2, 3): 0, (2, 2): -1, (2, 1): -1, (3, 2): 0, (3, 1): 0,
    (3, 0): -1, (2, 0): -1}
Queried cell: (1, 1)
--------------------------------------------------
knowledge base: [([(1, 0), (0, 0)], 1)]
Known safe cells: []
Cell info: {(3, 3): -1, (0, 2): 0, (0, 3): 0, (1, 3): 0, (1, 2): 0,
    (2, 3): 0, (2, 2): -1, (2, 1): -1, (3, 2): 0, (3, 1): 0, (3, 0): -1,
    (2, 0): -1, (1, 1): 0, (0, 1): -1}
Queried cell: (1, 0)
```

The agent works just as expected. The decision step and the inference step of the program work in harmony. The one thing that caught my attention was duplicate equations and after a little inspection, I was able to figure out that the source was the subset solver. Specifically, $([(2,2),(1,1),(0,1),(2,3),(2,1)],3)$ and $([(2,3),(2,1),(2,2)],2)$ resulted in figuring out that $([(1,1),(0,1)],1)$. Now, python believes that $([(0,1),(1,1)],1)$ this is a different equation and that results in having duplicate equations.

Efficiency

I did encounter some constraints while implementing the minesweeper agent. One being the fact that I was using a list for the knowledge base and removal operation takes a lot of time. I overcame that by using a separate list and filtering as I iterate through the original knowledge base. This helped with the time constraint but put a load on the space constraint. Moreover, my subset solver iterates using nested loops and that adds to the running time of the agent. Some things I believe could have improved was using a constraint specific representation of the clues and knowledge base.

Academic integrity

I have read and abided by the rules laid out in the assignment prompt, I have not used anyone else's work for my project, my work is only mine.

Signed by: Parth Patel