CS440 Project 2: Minesweeper

Mustafa Sadiq (ms3035)

March, 2021

Introduction

We develop a game of minesweeper and two types of agents: basic and improved. Basic agent looks at a single clue to determine the state of the board while the improved agent uses the previous knowledge available and uses methods of inference and proof to combine multiple clues to draw a conclusion when possible or necessary.

Libraries required: *numpy*, *random*, *matplotlib.pyplot* and *math*

Environment

Environment represents the board and where the mines are located. The environment takes dimension d and a number of mines n and generates a random $d \times d$ board containing n mines. We can query the environment, which reports the element in the cell queried.

For each cell, a mine is represented using *'X'* and a non-mine cell using *'C'* where C is a digit from 0-8 representing number of surrounding mines.

Using *numpy.array* we created a board and devised the following algorithm to place mines and count neighbouring mines:

```
while n != 0:
    if board[random dim][random dim] != mine:
        cell = 'X'
        n = n - 1

for cell in board:
    if cell not a mine:
        cell = count_neighbouring_mines

def count_neighbouring_mines(board, point):
    count = 0
    for neighbour in neighbours of point:
        if neighbour == 'X':
            count += 1
    return count
```

Agent

Agent represents the player. An agent knows the size of the board and maintains a knowledge base containing the information gained from quering the environment. It takes these clues from queries to infer or deduce more about the environment.

Untraditional, the agent keeps on going even if it queries a mine until the entire board is revealed. The agent keeps a count of correctly queried cells and total mines.

The agent maintains a knowledge of the following:

- A set containing the **hidden cells**. At the start of the game it is initialized using the function *initialize_hidden_cells* which returns a set of all possible cells given a dim.
- A set containing all **safe cells**, initialized as an empty set.
- A set containing all **mine cells**, initialized as an empty set.

We do this to have a board that can be visualized at any point in the game, since a board has these three types of cells. Also, since the agent does not have knowledge of total mines, it is easier to implement, starting with list of all coordinates to *hidden* cells.

Basic agent

The basic agent has two type of inference given a clue:

- If clue - known neighbour mines = number of hidden neighbours then every hidden neighbour is a mine.
- If number of neighbours - clue - known neighbour safe = number of hidden neighbours then every hidden neighbour is safe.

The basic agent tries these two methods of inference for every safe cell untill new knowledge is found. If no inference can be made then it chooses a random cell from the hidden cells to query. Following is a overview:

```
def basic agent:
    for safe_cell in safe_cells:
        try to infer new knowledge
if hidden cells:
    if basic agent worked:
        hidden, safe, mine updated
    else:
        query random cell from hidden
else:
    game finished
```

Improved agent

The improved agent is implemented on top of the basic agent. We call it **DILIGENCE** for its more thoughtfull inferencce, almost double intelligence.

When the basic agents fails, DILIGENCE builds and solves a knowledge base of equations from every safe cell in the following form:

$$\textit{set of cells} = \textit{number of mines}$$

An example $\{(1,1),(2,2)\} = 2$, represents the cells (1,1), (2,2) and since the number of mines are 2, both (1,1) and (2,2) are mines.

An equation is stored as a list, the first element being the set of cells and the second element being the number of mines. A list of these equations makes up the knowledge base.

Whenever basic agent is not able to infer any new knowledge, DILIGENCE builds a knowledge base of these equations from safe cells and uses the following methods of inference on it:

- if one equation is a subset of another, then we subtract them to get a new equation. after a new equation is found it is added to the knowledge base and it is again sovled using this new knowledge. For example if we have A+B+C+D = 3 and A+C = 1, we can infer that B+D = 2

- if two equations have equal number of cells and their intersection has one less cell, then we substract the one with lesser mines from the other. If the result is in the form X-Y = 1, we know for sure that X = 1 and Y = 0. This new knowledge is added to the knowledge base. For example if we have A+B+C = 1 and A+C+D = 2, the intersection is A+C, we can subtract the second one from the first to get D-B = 1, from which we can infer that D = 1 and B = 0.
- if for any equation, length of set of cells = number of mines then all cells are mines
- if for any equation, number of mines = 0 then all cells are safe

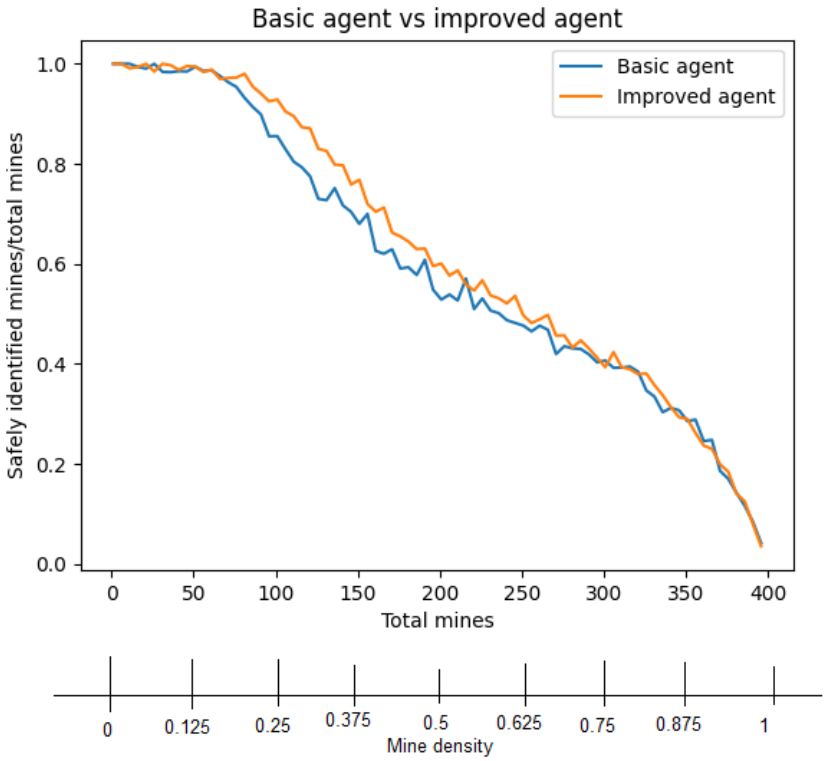
If no inference can be made then it chooses a random cell from the hidden cells to query. Following is a overview:

```
def basic agent:
    for safe_cell in safe_cells:
        try to infer new knowledge
def improved agent:
    for safe_cell in safe_cells:
        build a knowledge_base of equations
    try to infer new knowledge from this knowledge_base
if hidden cells:
    if basic agent worked:
        hidden, safe, mine updated
    elif improved agent worked:
        hidden, safe, mine updated
    else:
        query random cell from hidden
else:
    game finished
```

Basic agent vs improved agent

For $dim = 20$, following is a performance comparison with basic agent:

Figure 1: Basic agent vs improved agent performance



Analysis:

- The graph makes sense, as with less mine densities there is more success as we have more safe cells to infer knowledge from. At high mine densities, most cells are mines so we have less safe cells to infer knowledge from. It also shows how the agent is not aware of total mines.
  - At low mine density, mainly from 0 to 0.25, both agents perform similarly.
  - At high mine density, mainly from 0.75 to 1, both agents perform similarly.
  - At no point does basic agent perform better than improved agent.
- Minesweeper becomes hard as mine density increases.
- At middle densities, mainly from 0.25 to 0.75, improved agent performs noticeably better. Basic agent does not perform well only looking at one clue at a time, thus the improved agent which looks at multiple clues has better success.

Performance

Improved agent is able to infer every possible knowledge that can be inferred with certainty. It tries to infer every possible knowledge using the basic agent model and then tries to recursively solve a knowledge base built from safe cells. It is possible to increase the performance using global information of total mines and better decision making when picking a random cell.

Efficiency

Since at any mine density basic agent it able to perform almost at par with improved agent, we try the basic agent to infer any knowledge. The basic agent does not maintain a knowledge base of equations but rather the three sets: hidden, safe and mines. When the basic agent is not able to infer any new knowledge, improved agent build a knowledge base of equations from all safe cells.

The basic agent does most of the work which utilizes a bunch of set membership checks in constant time. When new knowledge is inferred, *hidden, safe and mine* sets updated accordingly. This is efficient than updating the knowledge base of equations and recursively trying to infer new knowledge.

Improved agent: play by play progression

For $dim = 5$ and $n = 5$ and a run where improved agent came into play, following is a play by play progression:

Figure 2: Environment board

```
--------------------
| 1 | X | 1 | 0 | 0 |
--------------------
| 1 | 2 | 2 | 1 | 0 |
--------------------
| 0 | 2 | X | 3 | 1 |
--------------------
| 0 | 2 | X | X | 2 |
--------------------
| 0 | 1 | 3 | X | 2 |
--------------------
```

Figure 3: Player initial board

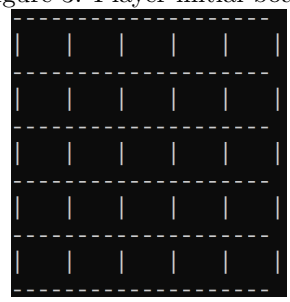
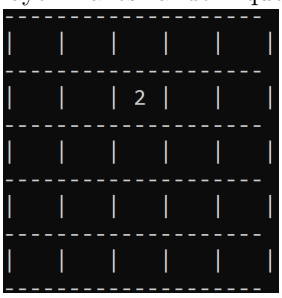


Figure 4: Player makes random query at (1, 2)



Knowledge base:

(0, 1), (2, 1), (1, 1), (0, 3), (2, 3), (0, 2), (2, 2), (1, 3) : 2

Figure 5: Player makes random query at (0, 4)

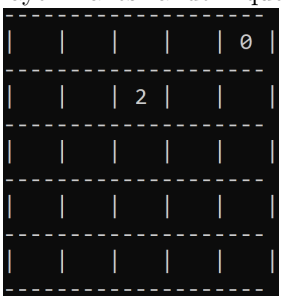


Figure 6: Basic agent flags (0, 3), (1, 4) and (1, 3) as safe

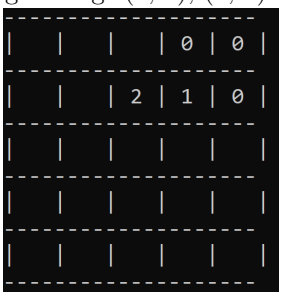


Figure 7: Basic agent flags (0, 2) as mine

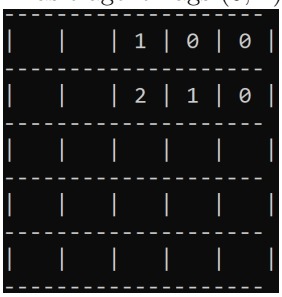


Figure 8: Basic agent flags (2, 4) and (2, 3) as safe

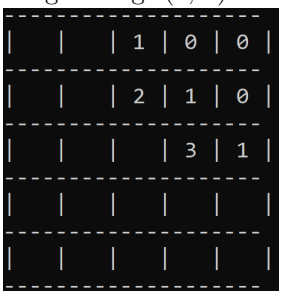
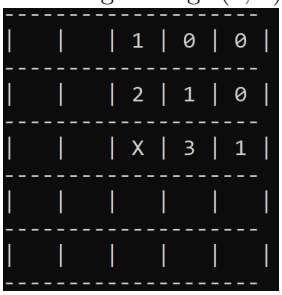


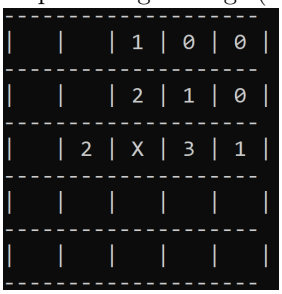
Figure 9: Basic agent flags (2, 2) as mine



Knowledge base:

(3, 3), (3, 4) : 1
(0, 1), (1, 1), (2, 1) : 1
(3, 2), (3, 3), (3, 4) : 2
(0, 1), (1, 1) : 1
new knowledge: [(2, 1), 0]

Figure 10: Improved agent flags (2, 1) as safe



Knowledge base:

(3, 3), (3, 4) : 1
(0, 1), (1, 1) : 1
(3, 1), (1, 1), (2, 0), (3, 0), (1, 0), (3, 2) : 1
(3, 2), (3, 3), (3, 4) : 2
new knowledge: [(3, 2), 1]

Figure 11: Improved agent flags (3, 2) as mine

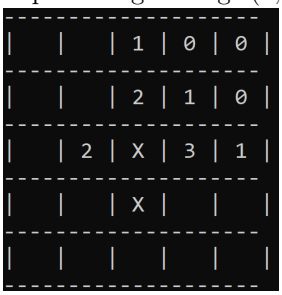


Figure 12: Basic agent flags (1, 1), (2, 0), (3, 1), (1, 0) and (3, 0) as safe

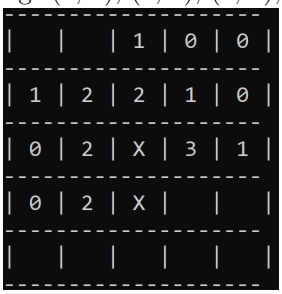


Figure 13: Basic agent flags (0, 1) as mine

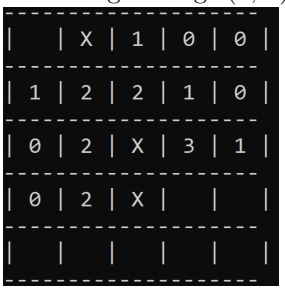


Figure 14: Basic agent flags (4, 1), (4, 0) and (4, 2) as safe

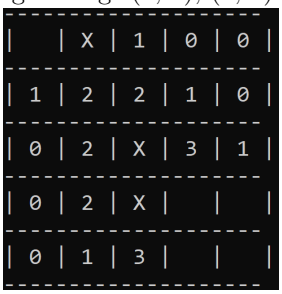


Figure 15: Basic agent flags (0, 0) as safe

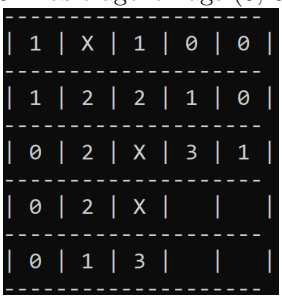


Figure 16: Basic agent flags (4, 2) and (3, 3) as mine

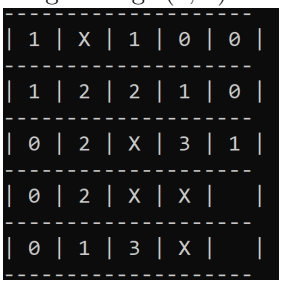


Figure 17: Basic agent flags (3, 4) as safe

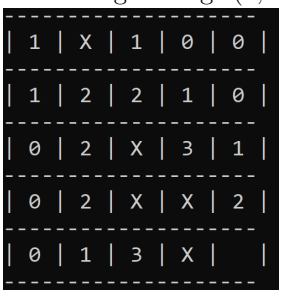
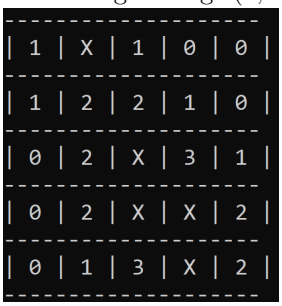


Figure 18: Basic agent flags (4, 4) as safe



Addendum

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: Mustafa Sadiq