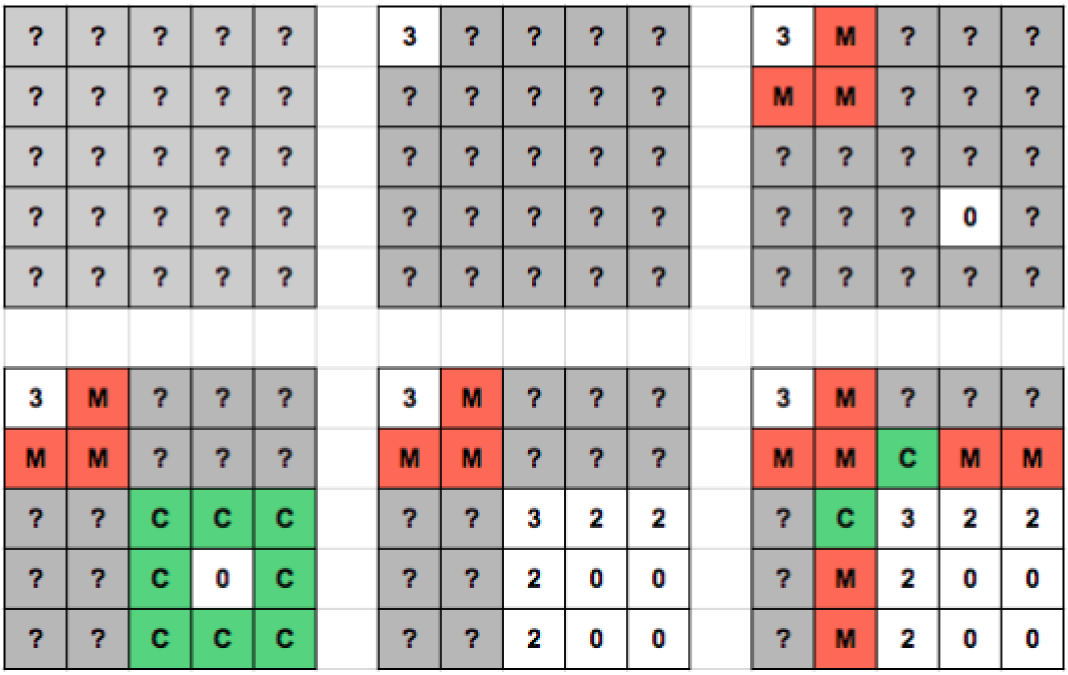
MineSweeper Report

CS 440



Cesar Herrera, Jae Weon Kim, Sunehar Sandhu

Spring 2020

List of Figures

Introduction

The goal of this project is to write a program to play Minesweeper- that is, a program capable of sequentially deciding what cells to check, and using the resulting information to direct future action.

Program Specification

There should effectively be two parts to the program, the **environment** representing the board and where the mines are located, and the **agent**. When the agent queries a location in the environment, the environment reports whether or not there was a mine there, and if not, how many of the surrounding cells are mines. The agent reports whether or not there was a mine there, and if not, how many of the surrounding cells are mines. The agent should maintain a knowledge base containing the information gained querying the environment, and should not only be able to update its knowledge base based on new information, bus also be able to perform inferences on that information and generate new information.

- The environment should take a dimension d and a number of mines n and generate a random d x d boards containing n mines. The agent will not have direct access to this location information, but will know the size of the board. *Note: It may be useful to have a version of the agent that allows for manual input, that can accecpt clues and feed you directions as you play an actual game of minesweeper in a seperate window.*
- In every round, the agent should assess its knowledge base, and decide what cell in the environment to query.
- In responding to a query, the environment should specify whether or not there was a mine there, and if not, how many surrounding cells have mines.
- The agent should take this clue, add it to its knowledge base, and perform any relevant inference or deductions to learn more about the environment. If the agen is able to determine that a cell has a mine, it should flag or mark it, and never query that cell. If the agent can determine a cell is safe, it's reasonable to query that cell in the next round.

- Traditionally, the game ends whenever the agent queries a cell with a mine in it - a final score being assessed in terms of number of mines safely identified.
- However, extend you agent in the following way: if it queries a mine cell, the mine goes off, but the agent can continue, using the fact that a mine was discovered there to update its knowledge base (but not receiving a clue about surrounding cells). In this way the game can continue until the entire board is revealed - a final score being assessed in terms of mines safely identified out of the total number of mines.

The last modification allows the game to 'keep going' and avoids the situation where you accidentally find a mine early in the game and terminate before the game gets interesting.

You may either do a GUI or text based ineterface - the important thing for the purpose of the project is the representation and manipulation of knowledge about the mine field. This will draw on the material discussed in a) Search, b) Constraint-Satisfaction Problems, and c) Logic and Satisfiablility. You must implement a basic MineSweeper agent, described in the next section, and your own agent as an improvement on the basic one.

A Basic Agent Algorithm for Comparison

Implements the following simple agent as a baseline strategy to compare against your own:

- For each cell, keep track of
  - whether or not it is a mine or safe (or currently covered)
  - if safe, the number of mines surrounding it indicated by the clue
  - the number of safe squares identified around it
  - the number of mines identified around it
  - the number of hidden squares around it.
- If, for a given cell, the total number of mines (the clue) minus the number of revealed mines is the number of hidden neighbors, every hidden neighbor is a mine

- If, for a given cell, the total number of safe neighbors (8-clue) minus the number of revealed safe neighbors is the number of hidden neighbors, every hidden neighbor is safe.
- If a cell is identified as safe reveal it and update your information
- If a cell is identified as a mine, mark it and update your information
- If no hidden cell can be conclusively identified as a mine or safe, pick a cell to reveal at random.

An Improved Agent

The algorithm described for the basic agent is a weak inference algorithm based entirely on local data and comparisons - it is effectively looking at a single clue at a time and determining what can be conclusively said about the state of the board. This is useful, and should be quite effective in a lot of situations. But not every situation - frequently multiple clues will interact in such a way to reveal more information when taken together.

Your improved agent should model the knowledge available, and use methods of inference to combine multiple clues to draw conclusions when possible or necessary. Not that 'knowledge' to model includes potentially: a) whether or not the square has been revealed, b) whether or not a revealed cell is a mine or safe, c) the clue number for a revealed safe cell, and d) inferred relationships between cells.

Questions and Writeup

Answer the following questions about the design choices you made in implementing this program, both from a representational and algorithmic perspective.

- Representation: How did you represent the board in your program, and how did you represent the information/knowledge that the clue cells reveal? How could you represent inferred relationships between cells?
- 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 we be sure of this, and if not, how could you consider improving it?

- Decisions: Given a current state of the board, and state of knowledge about the board, how does your program decide which cell to search next? Aside from always opening cells that are known to be safe, you could either a) open cells with the lowest probability of being a mine (*be careful - how would you compute this probability?*) or b) open cells that provide the most information about the remaining board (*what could this mean mathematically?*). Be clear and precise about your decision mechanism and how you implemented it. Are there any risks you face here, and how do you account for them?

- 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?

- Performance: 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 intution? When does minesweeper become 'hard'? When does your algorithm beat the simple algorithm, and when is the simple algorithm better? Why?

- Effeciency: 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 impove on?

- 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.

Questions and Writeup Answers

1. **Representation**:

   A Basic Agent

   (a) The board is represented by a matrix which is constructed of a 2D matrix using a GUI(Pygame) with a user defined n x n grid.

   (b) The information is represented in a list(fringe) which checks if the current tile has a mine. We enumerate respectively: true = 1, false = 2, hidden = 3. We keep track of potential mines and previous mines in a list and use them to update our knowledge base. It uses previous clues to make more informed decisions for picking future tiles.

   (c)

   Improved Agent

   (a) The board is represented by a matrix which is constructed of a 2D matrix using a GUI(Pygame) with a user defined n x n grid.

   (b) The information is represented as a list(fringe) which checks if the current tile has a mine. The improved agent performs probability checks if there is no tile to safely visit. Agent takes a guess if there is nothing clear and probability checks every viable option. The first guess is the middle of the board.

   (c) Takes a baseline inference at the start which is a hypothetical inference for the score rate

      i. The agent then takes an inference based off a system of equations to check if cells are mined or safe which are then stored in a list

         A. This is then used in our processQuery function which checks the surrounding neighbors and marks them as safe or mined

      ii. The agent does a probability inference which checks the probability of a hidden mines

2. **Inference**:

(a) After collecting a new clue, our knowledge base implements the use of a fringe by checking all the adjacent neighbors

(b) There are 3 things we need to do:

i. Compute value of the block we have just explored

ii. Update all neighbors' values

iii. When we check a cell at random, we return -1 if there is no mine or a number if there are neighboring mines. Then we choose an adjacent neighbor, however, we do not check it. We check the adjacent neighbor's neighbors' and choose a cell. We compute the probability of those neighboring cells and when we go back to the original cell we chose, we add the probability of that number's cells. Based on the computed probabilities, the ones with the higher probabilities are flagged as potential mines. We continue this process and see if our agent makes the right choices. However, one thing to note is that since combining clues can be time-consuming without a proper direction or heuristic, it's best to stop combining with an already known clue. For a given knowledge base, even if we add more hints to it, the statements we can deduce are still the same. It is not economical to spend so much time combing clues when we can explore different deductions from somewhere else because new hints might lead to the same exact statements.

iv. There are still ways to combine clues together to improve the program. The structure of some hints, for example, value relationship and geometric relationship tend to be more solvable. We can improve our proof by contradiction by combining a large number of hints at one time, however, this makes it a little more time consuming.

3. **Performance**:

(a) When we check the play by play progression of the game, we find some uncertainties. *See Figure: below*

4. **Performance**:

5. **Efficiency**:

(a) Overall space complexity: $O(n^2)$

(b) Overall time complexity: $O(n^2)$

(c) Space complexity is problem specific but we need an n2 matrix to make the board

6. **Improvements**:

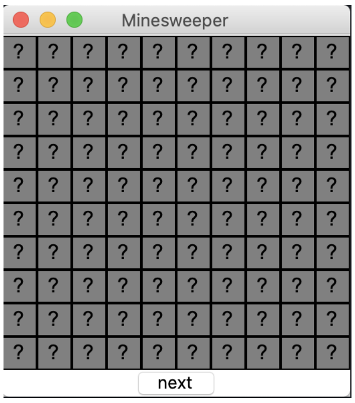
   (a) Given that if the board knows the number of bombs, we can keep a count of how many mines are left.

   (b) Our knowledge base can utilize the facts that it knows how many mines are left and once it computes the probability values for the adjacent cells, it would be able to pick the best option, given all of the information.

   (c) Then we would decrement the counter and every time the bomb count becomes less, the knowledge base updates and knows which cells are safer than others more effectively.

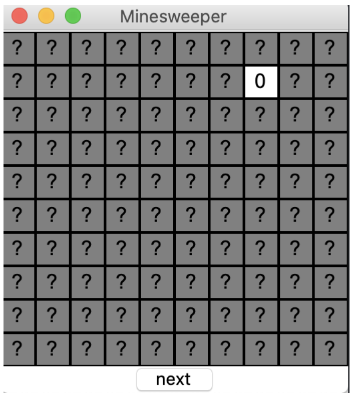
   (d)

(a) 1



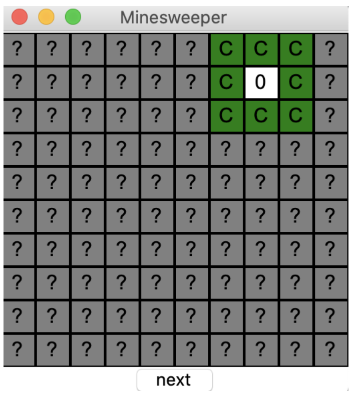
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?

(b) 2



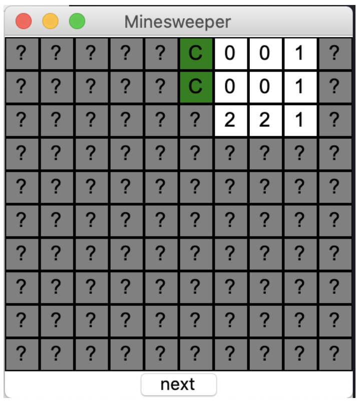
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	0	?	?
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?

(c) 3



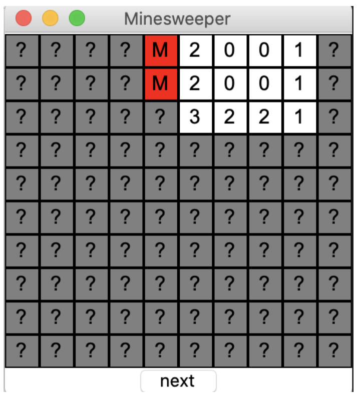
?	?	?	?	?	?	C	C	C	?
?	?	?	?	?	?	C	0	C	?
?	?	?	?	?	?	C	C	C	?
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?

(d) 4



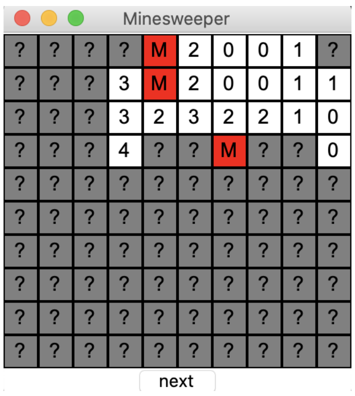
?	?	?	?	?	C	0	0	1	?
?	?	?	?	?	C	0	0	1	?
?	?	?	?	?	?	2	2	1	?
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?

(e) 5



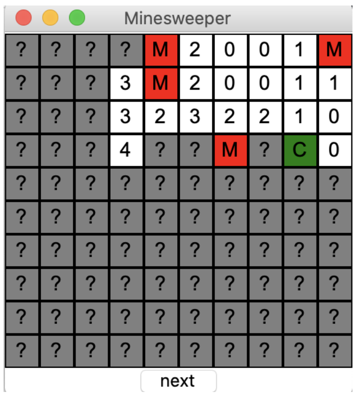
?	?	?	?	M	2	0	0	1	?
?	?	?	?	M	2	0	0	1	?
?	?	?	?	?	3	2	2	1	?
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?

(f) 6



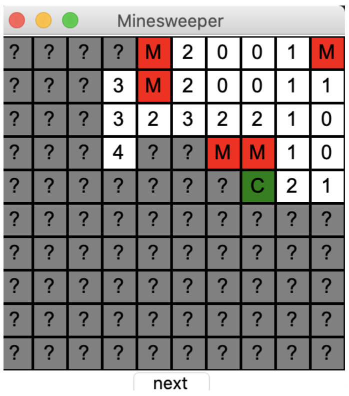
?	?	?	?	M	2	0	0	1	?
?	?	?	3	M	2	0	0	1	1
?	?	?	3	2	3	2	2	1	0
?	?	?	4	?	?	M	?	?	0
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?

(g) 7



?	?	?	?	M	2	0	0	1	M
?	?	?	3	M	2	0	0	1	1
?	?	?	3	2	3	2	2	1	0
?	?	?	4	?	?	M	?	C	0
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?

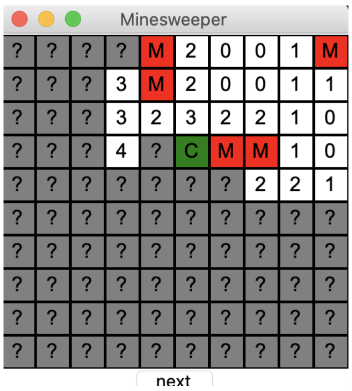
(h) 8



?	?	?	?	M	2	0	0	1	M
?	?	?	3	M	2	0	0	1	1
?	?	?	3	2	3	2	2	1	0
?	?	?	4	?	?	M	M	1	0
?	?	?	?	?	?	?	C	2	1
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?

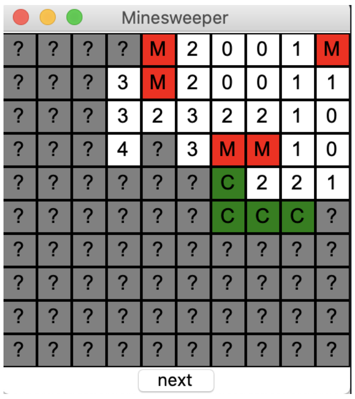
Figure 1: Game Example

(a) 9



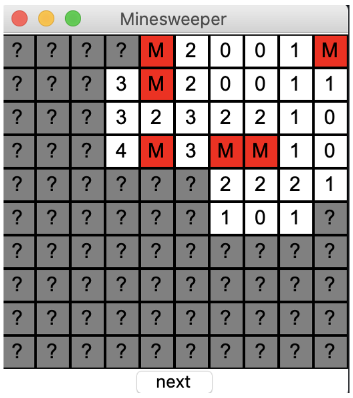
?	?	?	?	M	2	0	0	1	M
?	?	?	3	M	2	0	0	1	1
?	?	?	3	2	3	2	2	1	0
?	?	?	4	?	C	M	M	1	0
?	?	?	?	?	?	?	2	2	1
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?

(b) 10



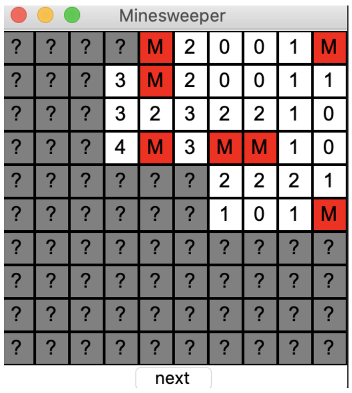
?	?	?	?	M	2	0	0	1	M
?	?	?	3	M	2	0	0	1	1
?	?	?	3	2	3	2	2	1	0
?	?	?	4	?	3	M	M	1	0
?	?	?	?	?	?	C	2	2	1
?	?	?	?	?	?	C	C	C	?
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?

(c) 11



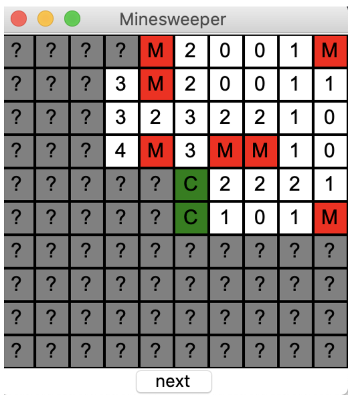
?	?	?	?	M	2	0	0	1	M
?	?	?	3	M	2	0	0	1	1
?	?	?	3	2	3	2	2	1	0
?	?	?	4	M	3	M	M	1	0
?	?	?	?	?	?	2	2	2	1
?	?	?	?	?	?	1	0	1	?
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?

(d) 12



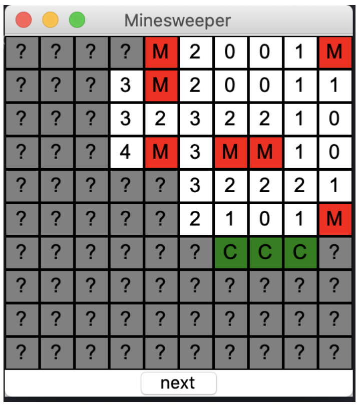
?	?	?	?	M	2	0	0	1	M
?	?	?	3	M	2	0	0	1	1
?	?	?	3	2	3	2	2	1	0
?	?	?	4	M	3	M	M	1	0
?	?	?	?	?	?	2	2	2	1
?	?	?	?	?	?	1	0	1	M
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?

(e) 13



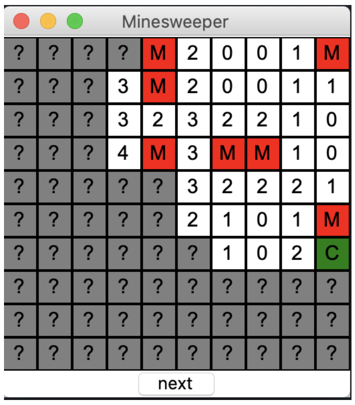
?	?	?	?	M	2	0	0	1	M
?	?	?	3	M	2	0	0	1	1
?	?	?	3	2	3	2	2	1	0
?	?	?	4	M	3	M	M	1	0
?	?	?	?	?	C	2	2	2	1
?	?	?	?	?	C	1	0	1	M
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?

(f) 14



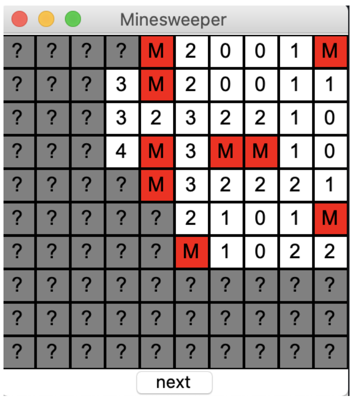
?	?	?	?	M	2	0	0	1	M
?	?	?	3	M	2	0	0	1	1
?	?	?	3	2	3	2	2	1	0
?	?	?	4	M	3	M	M	1	0
?	?	?	?	?	3	2	2	2	1
?	?	?	?	?	2	1	0	1	M
?	?	?	?	?	?	C	C	C	?
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?

(g) 15



?	?	?	?	M	2	0	0	1	M
?	?	?	3	M	2	0	0	1	1
?	?	?	3	2	3	2	2	1	0
?	?	?	4	M	3	M	M	1	0
?	?	?	?	?	3	2	2	2	1
?	?	?	?	?	2	1	0	1	M
?	?	?	?	?	?	1	0	2	C
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?

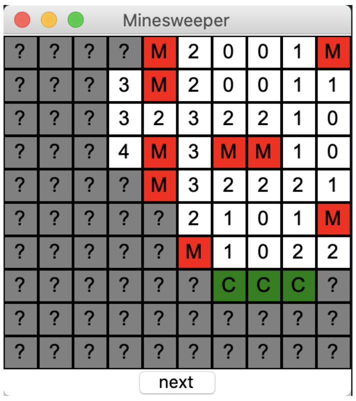
(h) 16



?	?	?	?	M	2	0	0	1	M
?	?	?	3	M	2	0	0	1	1
?	?	?	3	2	3	2	2	1	0
?	?	?	4	M	3	M	M	1	0
?	?	?	?	M	3	2	2	2	1
?	?	?	?	?	2	1	0	1	M
?	?	?	?	?	M	1	0	2	2
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?

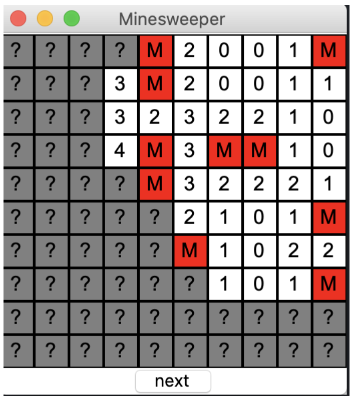
Figure 2: Game Example continued...(2)

(a) 17



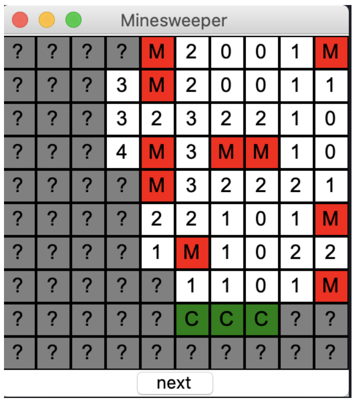
?	?	?	?	M	2	0	0	1	M
?	?	?	3	M	2	0	0	1	1
?	?	?	3	2	3	2	2	1	0
?	?	?	4	M	3	M	M	1	0
?	?	?	?	M	3	2	2	2	1
?	?	?	?	?	2	1	0	1	M
?	?	?	?	?	M	1	0	2	2
?	?	?	?	?	?	C	C	C	?
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?

(b) 18



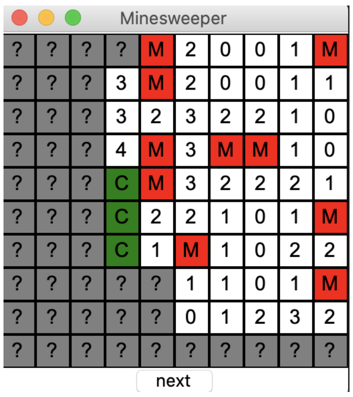
?	?	?	?	M	2	0	0	1	M
?	?	?	3	M	2	0	0	1	1
?	?	?	3	2	3	2	2	1	0
?	?	?	4	M	3	M	M	1	0
?	?	?	?	M	3	2	2	2	1
?	?	?	?	?	2	1	0	1	M
?	?	?	?	?	M	1	0	2	2
?	?	?	?	?	?	1	0	1	M
?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?

(c) 19



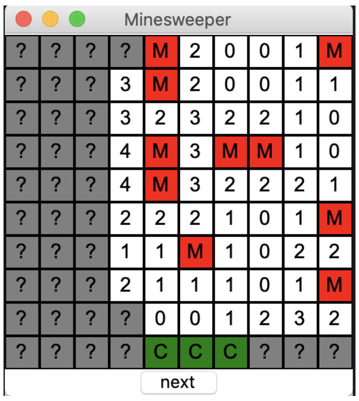
?	?	?	?	M	2	0	0	1	M
?	?	?	3	M	2	0	0	1	1
?	?	?	3	2	3	2	2	1	0
?	?	?	4	M	3	M	M	1	0
?	?	?	?	M	3	2	2	2	1
?	?	?	?	2	2	1	0	1	M
?	?	?	?	1	M	1	0	2	2
?	?	?	?	?	1	1	0	1	M
?	?	?	?	?	C	C	C	?	?
?	?	?	?	?	?	?	?	?	?

(d) 20



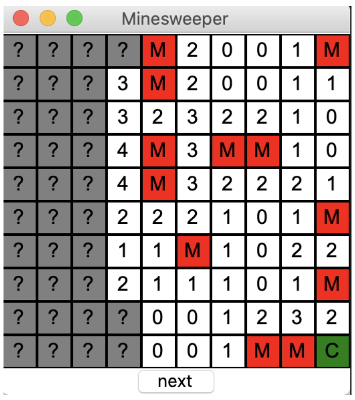
?	?	?	?	M	2	0	0	1	M
?	?	?	3	M	2	0	0	1	1
?	?	?	3	2	3	2	2	1	0
?	?	?	4	M	3	M	M	1	0
?	?	?	C	M	3	2	2	2	1
?	?	?	C	2	2	1	0	1	M
?	?	?	C	1	M	1	0	2	2
?	?	?	?	?	1	1	0	1	M
?	?	?	?	?	0	1	2	3	2
?	?	?	?	?	?	?	?	?	?

(e) 21



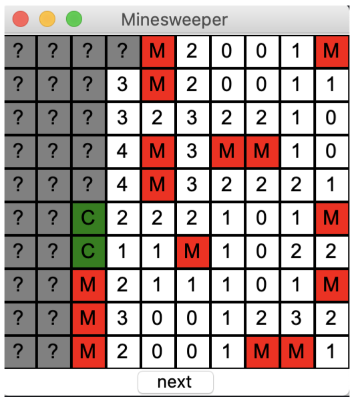
?	?	?	?	M	2	0	0	1	M
?	?	?	3	M	2	0	0	1	1
?	?	?	3	2	3	2	2	1	0
?	?	?	4	M	3	M	M	1	0
?	?	?	4	M	3	2	2	2	1
?	?	?	2	2	2	1	0	1	M
?	?	?	1	1	M	1	0	2	2
?	?	?	2	1	1	1	0	1	M
?	?	?	?	0	0	1	2	3	2
?	?	?	?	C	C	C	?	?	?

(f) 22



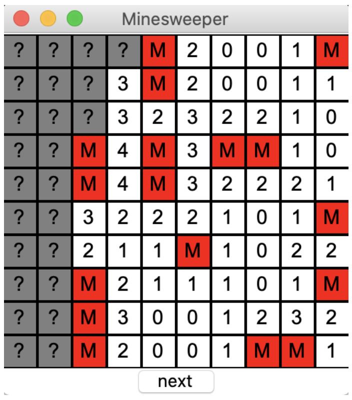
?	?	?	?	M	2	0	0	1	M
?	?	?	3	M	2	0	0	1	1
?	?	?	3	2	3	2	2	1	0
?	?	?	4	M	3	M	M	1	0
?	?	?	4	M	3	2	2	2	1
?	?	?	2	2	2	1	0	1	M
?	?	?	1	1	M	1	0	2	2
?	?	?	2	1	1	1	0	1	M
?	?	?	?	0	0	1	2	3	2
?	?	?	?	0	0	1	M	M	C

(g) 23



?	?	?	?	M	2	0	0	1	M
?	?	?	3	M	2	0	0	1	1
?	?	?	3	2	3	2	2	1	0
?	?	?	4	M	3	M	M	1	0
?	?	?	4	M	3	2	2	2	1
?	?	C	2	2	2	1	0	1	M
?	?	C	1	1	M	1	0	2	2
?	?	M	2	1	1	1	0	1	M
?	?	M	3	0	0	1	2	3	2
?	?	M	2	0	0	1	M	M	1

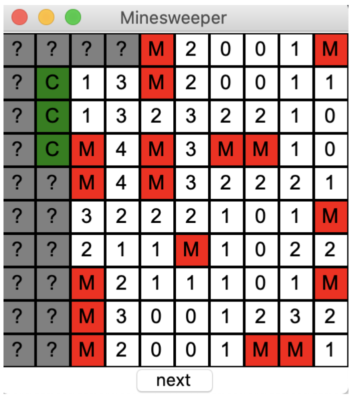
(h) 24



?	?	?	?	M	2	0	0	1	M
?	?	?	3	M	2	0	0	1	1
?	?	?	3	2	3	2	2	1	0
?	?	M	4	M	3	M	M	1	0
?	?	M	4	M	3	2	2	2	1
?	?	3	2	2	2	1	0	1	M
?	?	2	1	1	M	1	0	2	2
?	?	M	2	1	1	1	0	1	M
?	?	M	3	0	0	1	2	3	2
?	?	M	2	0	0	1	M	M	1

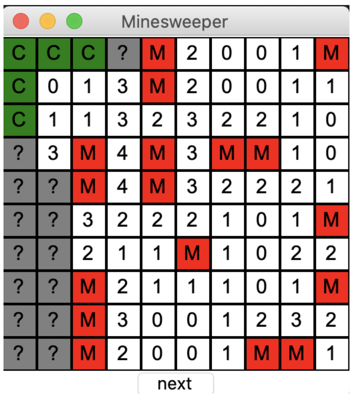
Figure 3: Game Example continued...(3)

(a) 25



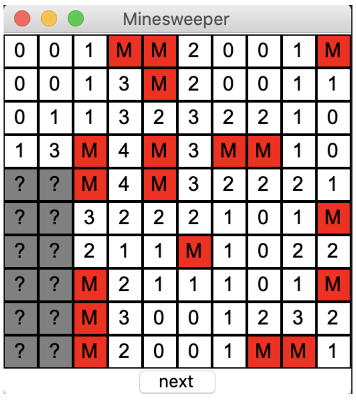
?	?	?	?	M	2	0	0	1	M
?	C	1	3	M	2	0	0	1	1
?	C	1	3	2	3	2	2	1	0
?	C	M	4	M	3	M	M	1	0
?	?	M	4	M	3	2	2	2	1
?	?	3	2	2	2	1	0	1	M
?	?	2	1	1	M	1	0	2	2
?	?	M	2	1	1	1	0	1	M
?	?	M	3	0	0	1	2	3	2
?	?	M	2	0	0	1	M	M	1

(b) 26



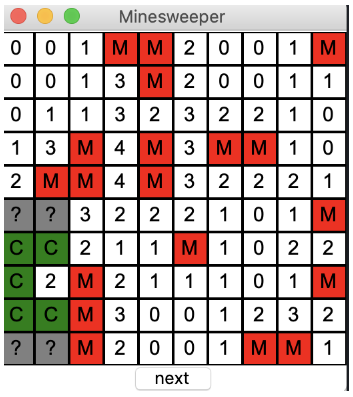
C	C	C	?	M	2	0	0	1	M
C	0	1	3	M	2	0	0	1	1
C	1	1	3	2	3	2	2	1	0
?	3	M	4	M	3	M	M	1	0
?	?	M	4	M	3	2	2	2	1
?	?	3	2	2	2	1	0	1	M
?	?	2	1	1	M	1	0	2	2
?	?	M	2	1	1	1	0	1	M
?	?	M	3	0	0	1	2	3	2
?	?	M	2	0	0	1	M	M	1

(c) 27



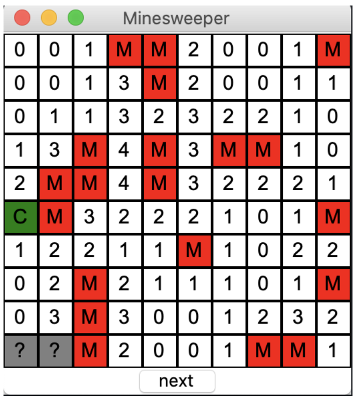
0	0	1	M	M	2	0	0	1	M
0	0	1	3	M	2	0	0	1	1
0	1	1	3	2	3	2	2	1	0
1	3	M	4	M	3	M	M	1	0
?	?	M	4	M	3	2	2	2	1
?	?	3	2	2	2	1	0	1	M
?	?	2	1	1	M	1	0	2	2
?	?	M	2	1	1	1	0	1	M
?	?	M	3	0	0	1	2	3	2
?	?	M	2	0	0	1	M	M	1

(d) 28



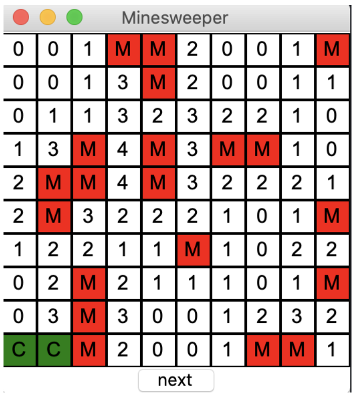
0	0	1	M	M	2	0	0	1	M
0	0	1	3	M	2	0	0	1	1
0	1	1	3	2	3	2	2	1	0
1	3	M	4	M	3	M	M	1	0
2	M	M	4	M	3	2	2	2	1
?	?	3	2	2	2	1	0	1	M
C	C	2	1	1	M	1	0	2	2
C	2	M	2	1	1	1	0	1	M
C	C	M	3	0	0	1	2	3	2
?	?	M	2	0	0	1	M	M	1

(e) 29



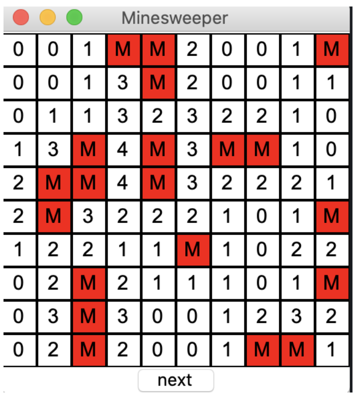
0	0	1	M	M	2	0	0	1	M
0	0	1	3	M	2	0	0	1	1
0	1	1	3	2	3	2	2	1	0
1	3	M	4	M	3	M	M	1	0
2	M	M	4	M	3	2	2	2	1
C	M	3	2	2	2	1	0	1	M
1	2	2	1	1	M	1	0	2	2
0	2	M	2	1	1	1	0	1	M
0	3	M	3	0	0	1	2	3	2
?	?	M	2	0	0	1	M	M	1

(f) 30



0	0	1	M	M	2	0	0	1	M
0	0	1	3	M	2	0	0	1	1
0	1	1	3	2	3	2	2	1	0
1	3	M	4	M	3	M	M	1	0
2	M	M	4	M	3	2	2	2	1
2	M	3	2	2	2	1	0	1	M
1	2	2	1	1	M	1	0	2	2
0	2	M	2	1	1	1	0	1	M
0	3	M	3	0	0	1	2	3	2
C	C	M	2	0	0	1	M	M	1

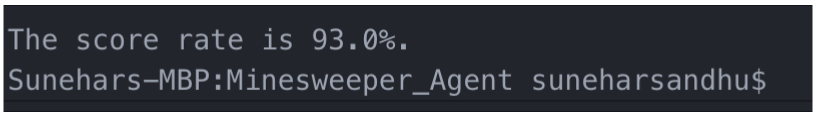
(g) 31



0	0	1	M	M	2	0	0	1	M
0	0	1	3	M	2	0	0	1	1
0	1	1	3	2	3	2	2	1	0
1	3	M	4	M	3	M	M	1	0
2	M	M	4	M	3	2	2	2	1
2	M	3	2	2	2	1	0	1	M
1	2	2	1	1	M	1	0	2	2
0	2	M	2	1	1	1	0	1	M
0	3	M	3	0	0	1	2	3	2
0	2	M	2	0	0	1	M	M	1

Figure 4: Game Example continued...(4)

Figure 5: Results



```
The score rate is 93.0%.
Sunehars-MBP:Minesweeper_Agent suneharsandhu$
```

Contributions

Jae: All code for following files: *GameSetting.py, ImprovedAgent.py, ImprovedGamesetting.py, ImprovedVisualization.py, MinesweeperAgent.py*)
Cesar: *MinesweeperEnvironment.py*, creation of LaTex doc
Sunehar: Questions and Writeup