Minesweeper, Part2: The Cascading Algorithm

SHAPE Summer 19, Computer Science

July 3, 2019

Update the minesweeper file you worked on yesterday.

In this part, we will work on two aspects of the Minesweeper game: Maintaining the player's view of the game board and uncovering areas of the board.

Modifying The Board Representation

So far, each cell on our board has only two different states. Either the cell is empty, or it contains a bomb. To use the map in a game, we also need to represent the view the user has of the board.

We will use special numeric values to represent the different states. The board contains two kinds of cells: those that have been discovered by the user and those that are still hidden. Hidden fields may contain either a bomb or are empty. For each discovered cell, the user can see how many adjacent bombs there are.

- `None` for an empty cell that is undiscovered by the user (i.e. the user does not know that it is empty).
- integer `-1` for a field that contains a bomb. Any bomb cell must be undiscovered (why?).
- integer `0` for an empty cell that has been discovered, but there are no adjacent Bombs.
- integer `1 to 8` for a discovered cell with the corresponding number of adjacent bombs.

Modify the `bury_bombs` function, so that it places the -1 symbol to indicate bombs. Then write a function `user_view(gameboard)` that displays the gameboard as seen by the users. In other words, if a cell contains 0, print out `.`. If it contains any other positive integer, print out that integer. If it contains None or a bomb, print out "?".

In the final game, we might to allow the user to mark undiscovered cells that they think do contain a bomb (red flags). How would you represent this information in the board data structure?

Uncovering the Board: The Cascading Algorithm

In the final game, areas will be uncovered if the user clicks on an undiscovered cell of the map. For now, we will have a function that takes coordinates as a parameter and uncovers cells. We will use *recursion* to implement that algorithm.

Write the function `def uncover_board(gameboard, x,y)` that uncovers the board starting at coordinate pair x/y. The function should modify the data structure passed to `gameboard`.
You can assume that x/y never specifies a bomb cell. The rules for uncovering are as follows:

- If the cell at position x/y is discovered, do nothing.
- If the cell at position x/y contain any adjacent bombs, the cell on the map hould be set to the number of bombs (use the `get_mine_count` function).
- If the cell at position x/y has not been discovered, but has no adjacent bombs:
  - Set the cell on the map to 0.
  - Proceed to uncover all adjacent cells. Make sure to pay attention to the boundaries of the board.

Think of this algorithm as a recursive algorithm. What is the sub-problem that is solved by each recursive call? What is the base case? How does each recursive call make progress towards the base case? How does the algorithm avoid re-computing overlapping subsolutions.

Testing the Cascading Algorithm

Write a function, that performs the following steps.

- Call the `create_board` function to create a new game board.
- Bury a number of mines on that board using the revised `bury_mines` function.
- Display the board using the `user_view` function.
- Repeatedly ask the user the enter x and y coordinates, modify the board by calling `uncover_board`, then re-display the board using the `user_view` function.

While this is not a complete Minesweeper game, this is getting close. What elements of the game are still missing (other than the GUI)?