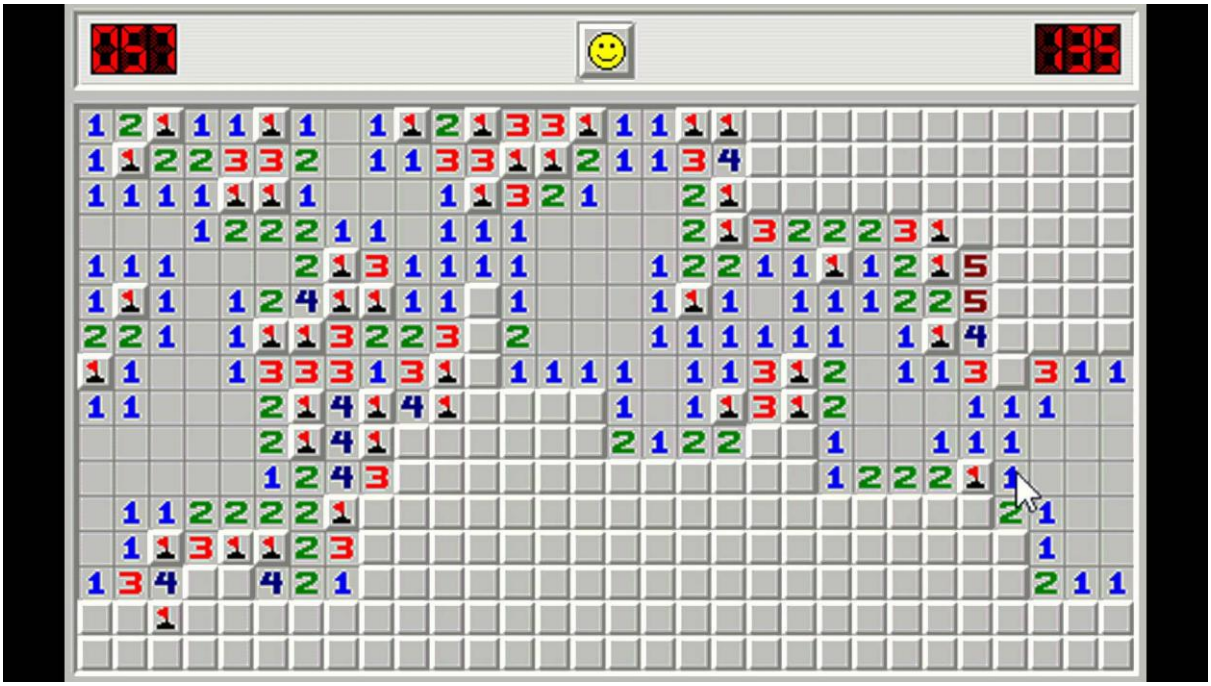
Program Name- Implementation of Minesweeper game

Project Category- Game Algorithm

Explanation-

Remember the old Minesweeper ?



Have you ever wondered how to implement it yourself ? Can we?

Yes we can.

Before proceeding on how to implement it, let us take a look at the rules of Minesweeper (for those who have forgotten this old yet addictive classic game).

We play on a square board and we have to click on the board on the cells which do not have a mine. And obviously we don't know where mines are. If a cell where a mine is present is clicked then we lose, else we are still in the game.

There are three levels for this game-

1) **BEGINNER-** 9 * 9 Board and 10 Mines
2) **INTERMEDIATE-** 16 * 16 Board and 40 Mines
3) **ADVANCED-** 24 * 24 Board and 99 Mines

If you calculate the probability of finding a mine then there is a probability of 10/81 (0.12) in the BEGINNER level and 40/256 (0.15) in the INTERMEDIATE level and 99 / 576 (0.17) in the ADVANCED level. Also the increasing number of tiles raises the difficulty bar. So the toughness level increases as we proceed to next levels.

It might seem like a complete luck-based game (you are lucky if you don't step over any mine over the whole game and unlucky if you have stepped over one).

But this is not a complete luck based game. Instead you can win almost every time if you follow the hints given by the game itself.

So what are the hints ?

When we click on a cell having adjacent mines in one or more of the surrounding eight cells, then we get to know how many adjacent cells have mines in them. So we can do some logical guesses to figure out which cells have mines.

Implementation-

There are two implementation of this game-
1) In the first implementation, the user's move is selected randomly using rand() function.
2) In the second implementation, the user himself select his moves using scanf() function.

Also there are two boards- realBoard and myBoard. We play our game in myBoard and realBoard stores the location of the mines. Throughout the game, realBoard remains unchanged whereas myBoard sees many changes according to the user's move.

We can choose any level among – BEGINNER, INTERMEDIATE and ADVANCED. This is done by passing one of the above in the function – chooseDifficultyLevel () [However in the user-input game this option is asked to the user before playing the game].

Once the level is chosen, the realBoard and myBoard are initialised accordingly and we place the mines in the realBoard randomly. We also assign the moves using the function assignMoves () before playing the game[However in the user-input game the user himself assign the moves during the whole game till the game ends].

We can cheat before playing (by knowing the positions of the mines) using the function – cheatMinesweepeer(). In the code this function is commented . So if you are afraid of losing then uncomment this function and then play !

Then the game is played till the user either wins (when the user never steps/clicks on a mine-containing cell) or lose (when the user steps/clicks on a mine-containing cell). This is represented in a while() loop. The while() loop terminates when the user either wins or lose.

The makeMove() function inside the while loop gets a move randomly from then randomly assigned moves. [However in the user-input game this function prompts the user to enter his own move].

Also to guarantee that the first move of the user is always safe (because the user can lose in the first step itself by stepping/clicking on a cell having a mine, and this would be very much unfair), we put a check by using the if statement – if (currentMoveIndex == 0)

The lifeline of this program is the recursive function – playMinesweeperUtil()

This function returns a true if the user steps/clicks on a mine and hence he loses else if he step/click on a safe cell, then we get the count of mines surrounding that cell. We use the function countAdjacentMines() to calculate the adjacent mines. Since there can be maximum 8 surrounding cells, so we check for all 8 surrounding cells.

If there are no adjacent mines to this cell, then we recursively click/step on all the safe adjacent cells (hence reducing the time of the game-play). And if there is atleast a single adjacent mine to this cell then that count is displayed on the current cell. This is given as a hint to the player so that he can avoid stepping/clicking on the cells having mines by logic.

Also if you click on a cell having no adjacent mines (in any of the surrounding eight cells) then all the adjacent cells are automatically cleared, thus saving our time.

So we can see that we don't always have to click on all the cells not having the mines (total number of cells – number of mines) to win. If we are lucky then we can win in very short time by clicking on the cells which don't have any adjacent cells having mines.

The user keeps on playing until he steps/clicks on a cell having a mine (in this case the user loses) or if he had clicked/stepped on all the safe cell (in this case the user wins).

Note-

There are two codes for this program-

1) *User's input chosen randomly*

2) *User's input given by himself*

References-

http://www.minesweeper.info/wiki/Windows_Minesweeper