CSE 102 Assigment VII

In this project, I implemented a simplified version of the classic Minesweeper game. The game is designed to be played in a console environment and includes functionality for both normal gameplay and undoing previous moves.

At the start of the program, a random board size is selected between a minimum and maximum value (2 to 10). Two boards are generated: one for displaying the player's view (board), and another hidden board containing the mine locations (mined_board). The boardGeneration() function initializes these boards by filling the player's board with hidden cells ('#') and the mine board with empty cells ('.').

Mines are randomly placed on the mine board using the mineGeneration() function. Approximately one-third of the board is filled with mines ('*'). Care is taken to avoid placing multiple mines on the same cell.

The main gameplay loop begins by printing the current state of the board to the console. The user is prompted either to input a move (row and column) or to type 'undo' to revert their last move. Moves are processed by checking whether the selected cell is within bounds and has not been opened already. If the cell contains a mine, the game immediately ends with a loss message. Otherwise, if the cell is empty, the floodFill() function is called to reveal the surrounding area, automatically opening adjacent non-mined cells. Each move is recorded with the push() function into move history arrays, and the number of flood-filled cells is separately tracked using push_flood_fill().

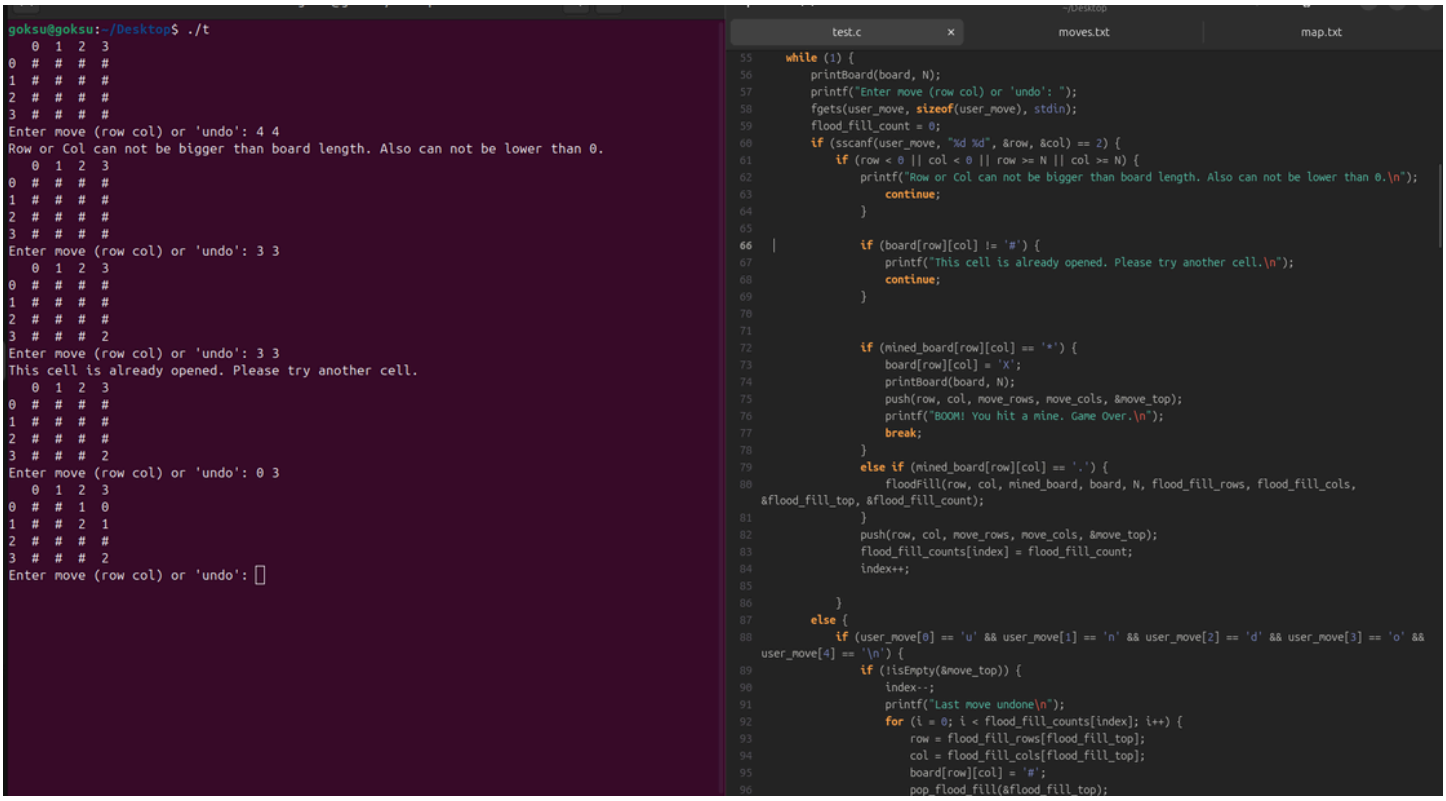
The floodFill() function recursively reveals connected empty cells. It uses a helper function, checkMineCount(), to determine the number of neighboring mines for each cell. If a cell has no adjacent mines, it continues expanding; if it has neighboring mines, it reveals the number and stops.

If the player chooses to undo a move, the program reverts the previous action by resetting the affected cells to the hidden state ('#'). This is managed by popping entries from the flood fill history with pop_flood_fill() and adjusting the move stack with pop(). The undo functionality ensures that both the selected cell and all automatically revealed cells are correctly hidden again.

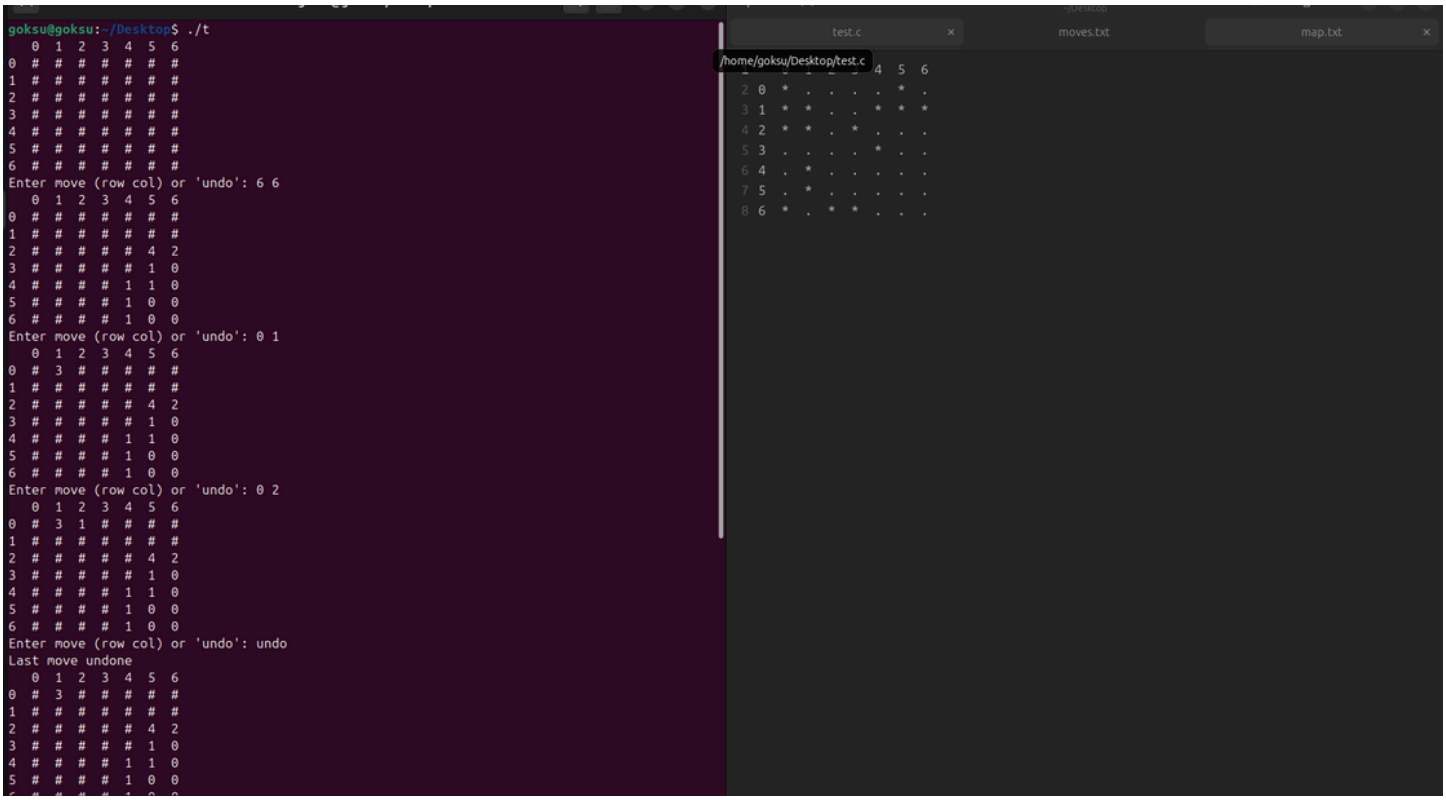
The game continues until the player either opens a mine or successfully opens all non-mined cells. If the player wins, a congratulatory message is displayed. Regardless of the outcome, all moves are saved into a file called moves.txt through the saveMoves() function, providing a record of the entire game session.

Additionally, at the beginning of the game, the initial mined board (map) is saved into a file named map.txt for reference. All file operations include proper error checking to ensure safe opening and writing.

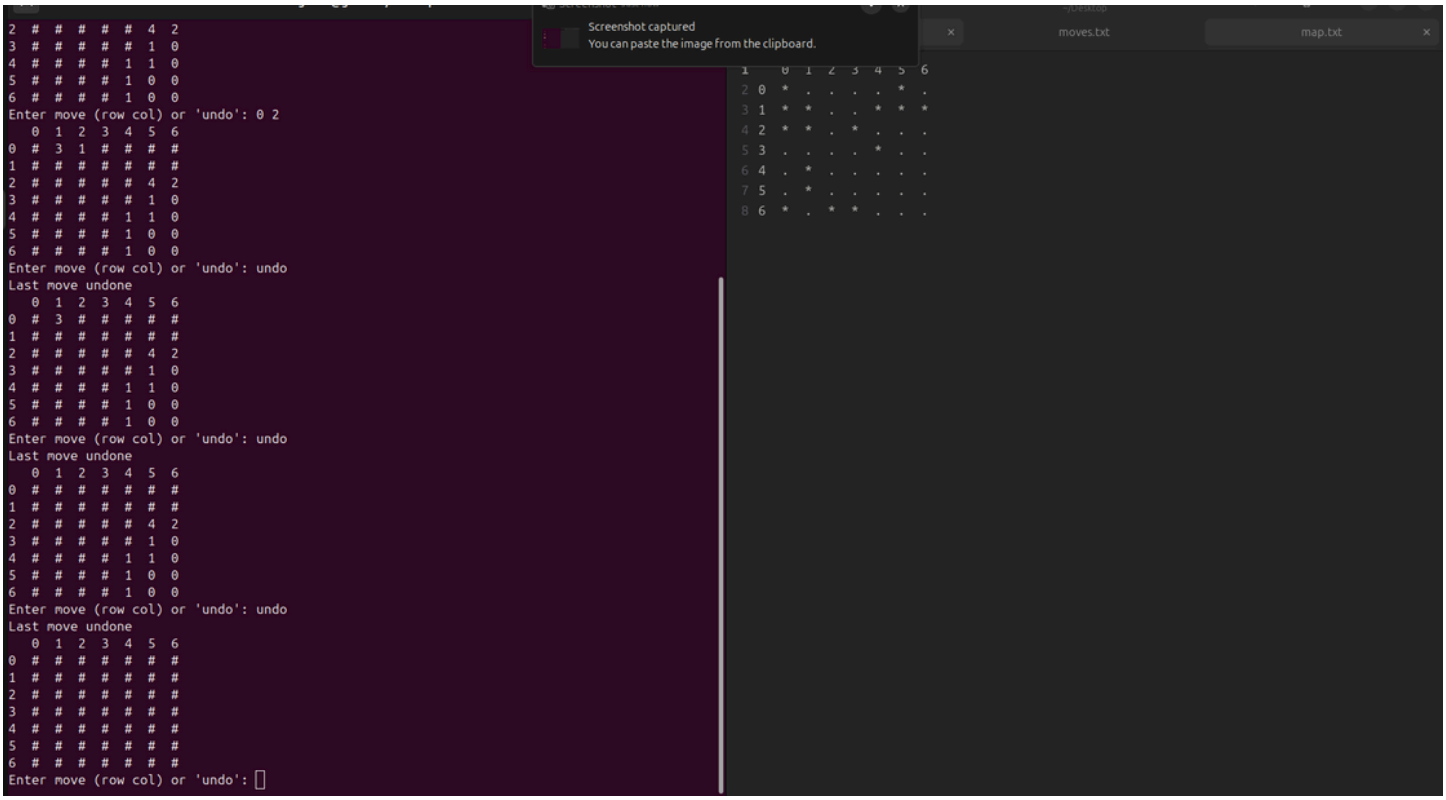
- **Some Input checks**



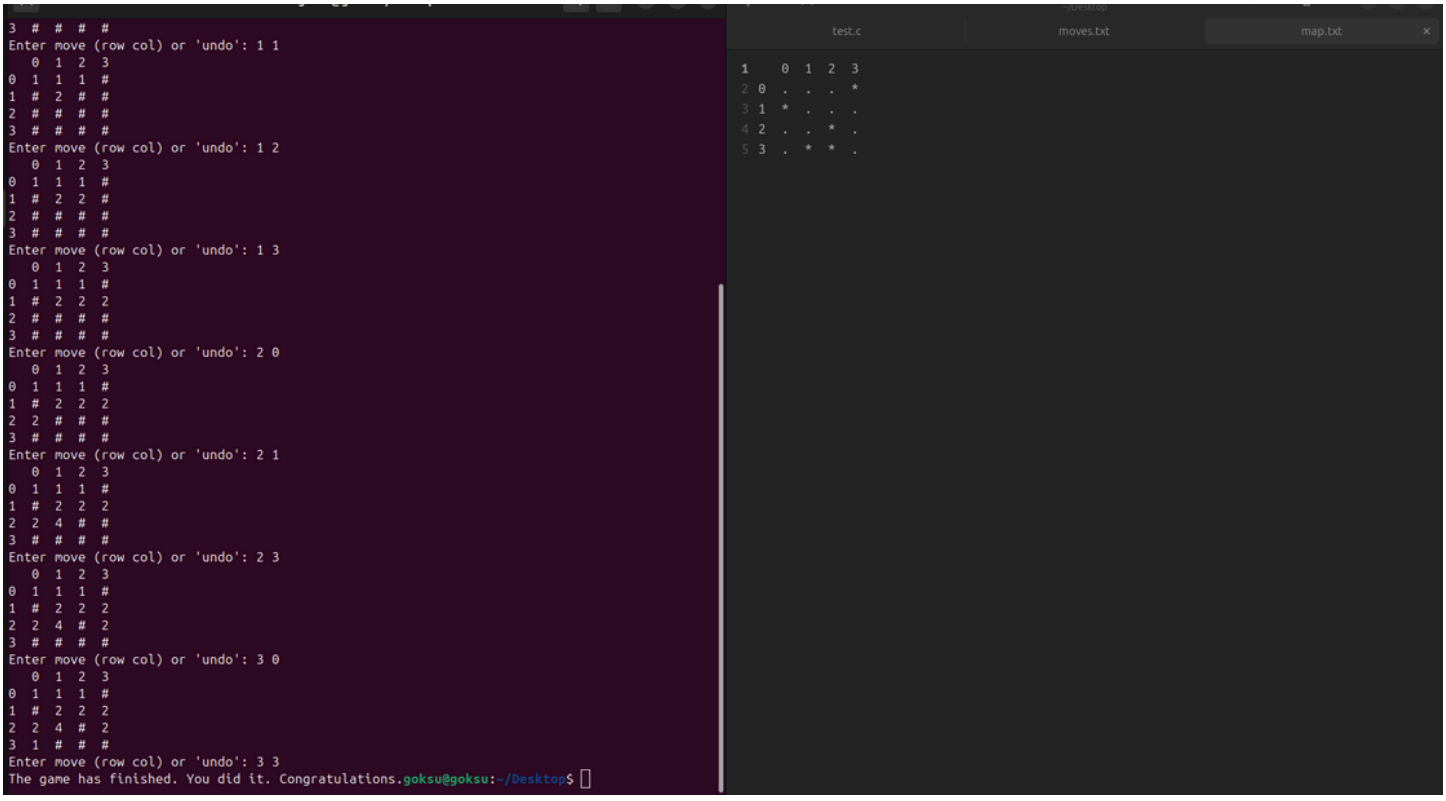
- **Opening Cell and Undo.**



- **Multiple Undo.**



- **Win the game**



- **Lose the game**

```
goksu@goksu:~/Desktop$ ./t
   0  1  2  3  4  5
0  #  #  #  #  #  #
1  #  #  #  #  #  #
2  #  #  #  #  #  #
3  #  #  #  #  #  #
4  #  #  #  #  #  #
5  #  #  #  #  #  #
Enter move (row col) or 'undo': 0 0
   0  1  2  3  4  5
0  1  #  #  #  #  #
1  #  #  #  #  #  #
2  #  #  #  #  #  #
3  #  #  #  #  #  #
4  #  #  #  #  #  #
5  #  #  #  #  #  #
Enter move (row col) or 'undo': 4 4
   0  1  2  3  4  5
0  1  #  #  #  #  #
1  #  #  #  #  #  #
2  #  #  #  #  #  #
3  #  #  #  #  #  #
4  #  #  #  #  2  #
5  #  #  #  #  #  #
Enter move (row col) or 'undo': 1 0
BOOM! You hit a mine. Game Over.   0  1  2  3  4  5
0  1  #  #  #  #  #
1  X  #  #  #  #  #
2  #  #  #  #  #  #
3  #  #  #  #  #  #
4  #  #  #  #  2  #
5  #  #  #  #  #  #
goksu@goksu:~/Desktop$
```

test.c | moves.txt | map.txt

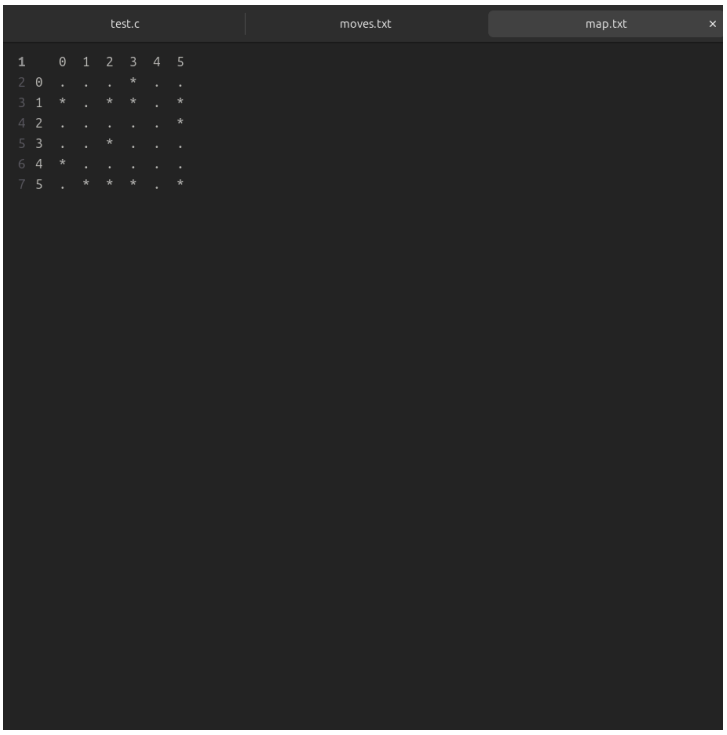
```c
    while (1) {
        printBoard(board, N);
        printf("Enter move (row col) or 'undo': ");
        fgets(user_move, sizeof(user_move), stdin);
        flood_fill_count = 0;
        if (sscanf(user_move, "%d %d", &row, &col) == 2) {
            if (row < 0 || col < 0 || row >= N || col >= N) {
                printf("Row or Col can not be bigger than board length. Also can not be lower than 0.\n");
                continue;
            }

            if (board[row][col] != '#') {
                printf("This cell is already opened. Please try another cell.\n");
                continue;
            }


            if (mined_board[row][col] == '*') {
                board[row][col] = 'X';
                printBoard(board, N);
                push(row, col, move_rows, move_cols, &move_top);
                printf("BOOM! You hit a mine. Game Over.\n");
                break;
            }
            else if (mined_board[row][col] == '.') {
                floodFill(row, col, mined_board, board, N, flood_fill_rows, flood_fill_cols,
&flood_fill_top, &flood_fill_count);
            }
            push(row, col, move_rows, move_cols, &move_top);
            flood_fill_counts[index] = flood_fill_count;
            index++;

        }
        else {
            if (user_move[0] == 'u' && user_move[1] == 'n' && user_move[2] == 'd' && user_move[3] == 'o' &&
user_move[4] == '\n') {
                if (!isEmpty(&move_top)) {
                    index--;
                    printf("Last move undone\n");
                    for (i = 0; i < flood_fill_counts[index]; i++) {
                        row = flood_fill_rows[flood_fill_top];
                        col = flood_fill_cols[flood_fill_top];
```

- **moves.txt.**

test.c | moves.txt | map.txt

```
1 --- Game Moves ---
2 Move 1:  (Row 0, Col 0)
3 Move 2:  (Row 4, Col 4)
4 Move 3:  (Row 1, Col 0)
5
6
7 Total Moves: 3
```

- map.txt


test.c
moves.txt
map.txt
1   0 1 2 3 4 5
2 0 . . . * . .
3 1 * . * * . *
4 2 . . . . . *
5 3 . . * . . .
6 4 * . . . . .
7 5 . * * * . *