Algorithm Design Project: MineSweeper Game and Auto-solver

Efrem Ştefan Hagiu Teodora

5th June 2018

Computer Science English Section First Year Group 1.1 A

Course Professor: Costin Badica
Laboratory Professor: Alex Becheru

1 Problem statement

The MineSweeper Game: a program that allows you to play the game yourself on a board of 16x16 spaces with 40 mines and the auto-solver for the game given the first move. The game consists in a board that has mines placed randomly on it and the goal is to find all the places that don't contain a mine.

2 Application Design

- The application uses one main function where we call the start game auxiliary function and 9 auxiliary functions called in the start game function and other functions in order to make the whole program more organized.

The main function is represented by :

- The use of auxiliary function start game.

The auxiliary functions for the game are:

- Build mines board function
  - It creates a char matrix and puts the mines randomly in it
  - After the mines have been placed, it puts numbers with how many mines are around a space in the matrix using the auxiliary function increment.
- Print mines board function.
  - This function prints the mines board matrix when you lose the game.
- Increment function
  - The function adds 1 at the space given.
- Build player board function
  - The function creates the player board matrix, the matrix that you see and use to see your moves.
- Print player board function
  - The function prints the player board matrix after every move you make, to see the changes done.
- Reveal spaces function
  - The function reveals the space you chose and if that space is a 0 on the mines board (with no mine around it) it reaveals all blank the spaces around it until it gets to the ones with mines around.

- Loss function
  - In case you hit a mine, this function shows you a loss message and asks you if you want to play again and in the case you do want, it calls the function start game again.
- Win function
  - In case you win the game this function shows a win message and asks you if you want to play again and in the case you do want, it calls the function start game again.
- Check win function
  - At every step you make, this function checks if you won the game (no spaces without a mine on them left) and returns TRUE (1) in case the game is finished and FALSE(0) otherwise.
- Start Game function
  - The build mines board and build player board are called.
  - The player board matrix is printed with the function print player board
  - A welcome message is printed along with the legend (the meaning of each character and number that appears).
  - With the help of variable decision (given by the player in the input) the function clears a space with the help of reveal spaces function or marks it with a M that stands for mine.
  - The player board with the new changes is printed.
  - With the use of check win function, at every step the win condition is checked, and if it is true the win function is called.

Besides the ones presented above, the auxiliary functions used in the auto-solver are:

- Build solver chance board function
  - It creates an int matrix where it stores the chances for a space to be a mine.This function is used in the start game function next to build player board and build mines board.
- Print solver chance board function
  - This function prints the solver chance board. It was used just for tests or to see how the program works.
- Look for mines function

  - Given a space on the board this function searches around to see how many mines and hidden spaces are. If the number of mines is equal to the number of the current space, it reveals all the other hidden spaces around. If the number of hidden spaces is equal to the number of the current space minus the number of mines found, it marks all the hidden spaces with M for mines.

- Automatic flagging function
  - This function goes through all the revealed numbers on the board and calls the look for mines function to reveal spaces around or mark the mines.

- Automatic chance placer function
  - The function goes through all the hidden spaces and calls the function place chance to see the chances.

- Place chance function
  - The function stores the chances for a space to be a mine in the solver chance board and adds the numbers of the revealed spaces around.

- Automatic decision taker function
  - The function goes through the spaces and searches for the one with the minimum chance (minimum number) and reveals it.

- Start Game function
  - The build mines board, build solver chance board and build player board functions are called.
  - The player board matrix is printed with the function print player board
  - A welcome message is printed along with the asking of the first move (the row and the column of the space you want to start from)
  - With the help of variable step (given by the player in the input) the program shows every step done to get to the final (the variable needs to be given after each step as a confirmation).
  - The player board with the new changes is printed
  - With the use of check win function, at every step the win condition is checked, and if it is true the win function is called.

2.1 Input Specification:

The input is given by the player and it is needed at every step. It consists in the decision, the row and the column of the space.

2.2 Output Specification.

The output consists in multiple prints of the player board matrix at every step made and finally the win ore lose messages along with the mines board matrix (in the case of a loss).

- The list of all files in the application and their description for the game:
  - minesweeper_game.c: the main file in which we create the application.Here we declare the libraries and headers.
  - pre_game_boards.c: the file in which the functions used for building and printing the matrices and the increment function are stored..
  - pre_game_boards.h : the header file in which the functions used for building and printing the matrices and the increment functions are declared.
  - in_game_functions_player.c : the file in which the functions reveal spaces, win, lose, check win and start game are stored.
  - in_game_functions_player.h : the header file in which the functions reveal spaces, win, lose, check win and start game are declared.
- The list of all files in the application and their description for the autosolver:
  - autosolvermsp.c: the main file in which we create the application.Here we declare the libraries and headers.
  - pre_game_boards.c: the file in which the functions used for building and printing the matrices and the increment function are stored.
  - pre_game_boards.h : the header file in which the functions used for building and printing the matrices and the increment function are declared.
  - pre_game_autosolver_boards.c : the file in which the functions used for building and printing the matrix that helps to keep the chances for a mine are stored.
  - pre_game_autosolver_boards.h : the file in which the functions used for building and printing the matrix that helps to keep the chances for a mine are declared.
  - in_game_functions_autosolver.c : the file in which the functions reveal spaces, win, lose, check win, look for mines, automatic flagging, automatic chance placer, place chance, automatic decision taker and start game are stored.
  - in_game_functions_autosolver.h : the header file in which the functions reveal spaces, win, lose, check win, look for mines, automatic flagging, automatic chance placer, place chance, automatic decision taker and start game are declared.

3 Conclusions

3.1 Achievements

Firstly, we achieved learning more about the game we play when we are bored and have some minutes. Secondly, we consider that this project was one of the most interesting ones, and that's why we decided to stick to it. We also have learned more about using LaTeX as a documentation method, and we are looking forward in new projects, interactive like this one where we learned how to think the game better and keep it simple at the same time.

3.2 Challenging parts

One of the interesting parts in the development process of the project was building the skeleton of the application, which needed a lot of patience and finally no one was harmed, even if code didn't work sometimes.It's the first time when we create a legit project with more than one .c file, so it was really interesting and fun and we also learned a lot.

Another interesting part of the project was learning how to use LaTeX as a documentation method, how to use DoxyGen as a comment extractor and see how the game we play so much works in the back along with learning about the different implementations and functions which were necessary for making the game actually work.

The most challenging part in this project was creating the functions as simple as possible. In programming time is money and things must be made in a short period of time. Being the first project, almost every task we had was challenging. We remember that we wanted the boards to look good and to be easy to see what space you want to choose, so we integrated numbering into matrix and we realised that, being a char matrix, the maximum number was 9 (too small for what we wanted) so we had to find another solution. Finally, we found that adding spaces at the print is really useful and everything became a lot more easier.

3.3 Future directions for extending the project

More options can be added for the boards and also a graphic part with unicorns instead of mines would be pretty awesome (UnicornSweeper - sounds pretty appealing to me).

4 Pseudocode

Figure 1: MineSweeper - the game

```
build_mines_board()
        if cell on the edges of the matrix
                cell <= ' '
        else
                cell <= '0'

        random cells become bombs '*'
        if cell = '*' increment(row, column) all adjacent cells

increment(row, column)
        mines_board[row][column] = mines_board[row][column] + 1

print_mines_board()
        print row and column of numbers on the outskirts of the board
        print mines board

build_player_board()
        if cell on the edges of the matrix
                cell <= ' '
        else
                cell <= '-'

print_player_board()
        print row and column of numbers on the outskirts of the board
        print player board

reveal_spaces (row, column)
        player_board[row][column] = mines_board[row][column]
        if player_board[row][column] = '0'
                reveal all adjacent cells

loss()
        you hit a mine
        type 'y' if you want to play again
        type 'n' otherwise

win()
        you won
        type 'y' if you want to play again
        type 'n' otherwise

checkwin()
        run through matrix
                if player_board_cell = '-' and mines_board_cell != '*'
                        FALSE
```

```
start_game()
        build_mines_board()
        build_player_board()
        print_player_board()

        type '1' if you want to flag a mine
        type '0' if you want to reveal a cell
        input: decision
        input: row and column

        if (decision = 1)
                if mines_board[row][column] == '*'
                        print_mines_board()
                        loss()

        else
                player_board[row][column] = 'M'

        if ( check_win = TRUE)
                win()
}

main()
        start_game()
```

Figure 2: MineSweeper - auto-solver

```
build_mines_board()
        if cell on the edges of the matrix
                cell <= ' '
        else
                cell <= '0'

        random cells become bombs '*'
        if cell = '*' increment(row, column) all adjacent cells

increment(row, column)
        mines_board[row][column] = mines_board[row][column] + 1

print_mines_board()
        print row and column of numbers on the outskirts of the board
                print mines board

build_player_board()
        if cell on the edges of the matrix
                cell <= ' '
        else
                cell <= '-'

print_player_board()
        print row and column of numbers on the outskirts of the board
                print player board

build_solver_chance_board
        if cell on the edges of the matrix
                cell <= '-1'
        else
                cell <= '60'

print_solver_chance_board()
        prints solver chance bvoard

reveal_spaces (row, column)

        player_board[row][column] = mines_board[row][column]
        if player_board[row][column] = '0'
                reveal all adjacent cells
```

```
look_for_mines(row, column)
        player_board_number_cell = the number that the cell contains on the player board
        if adjacent cells = '-'
                hidden_spaces <= hidden_spaces + 1
        if adjacent cells = 'M'
                mines <= mines + 1

        if hidden_spaces = 0
                return

        if mines = player_board_number_cell and hidden_spaces != 0
                reveal_spaces(row_adjacent_cell, column_adjacent_cell)

         if hidden_spaces = player_board_number_cell - mines
                player_board[row_adjacent_cell][column_adjacent_cell] <= 'M'

automatic_flagging()
        run through matrix
                if player_board_cell = digit
                        look_for_mines(row_player_board_cell, column_player_board_cell)

automatic_chance_placer()
        run through matrix
                if player_board_cell = '-'
                        place_chance(row_player_board_cell, column_player_board_cell)

place_chance(row, column)
        if adjacent_player_board_cell = digit
                confirm <= 1

        if confirm = 1
                solver_chance_board[row][column] <= 0

        if adjacent_player_board_cell = digit
                solver_chance_board[row][column] <= solver_chance_board[row][column] + adjacent_player_board_cell

automatic_decision_taker()
        minimum_chance <= 55;

        run through matrix
                if player_board_cell = '-' and solver_chance_cell < minimum_chance
                        row_minimum_chance <= player_board_cell_row
                        column_minimum_chance <= player_board_cell+column
                        minimum_chance_recorded <= solver_chance_cell

        reveal_space(row_minimum_chance, column_minimum_chance)
```

```
loss()
        you hit a mine
        type 'y' if you want to play again
        type 'n' otherwise

win()
        you won
        type 'y' if you want to play again
        type 'n' otherwise

checkwin()
        run through matrix
                if player_board_cell = '-' and mines_board_cell != '*'
                        FALSE

start_game()
        build_mines_board()
        build_player_board()
        print_player_board()

        first_move <= 1;
        input: row and column


        if (first_move <= 1)
                reveal_spaces(row, column)

        first_move <= 0

        type '1' to run a step of the program and see the progress of the computer
        input: 1

        if(input = 1)
                automatic_flagging()
                automatic_chance_placer()
                automatic_decision_taker()
                print_player_board()

        if ( check_win = TRUE)
                win()
}

main()
        start_game()
```

References

[1] https://drive.google.com/file/d/0B0qJvJ1d8F8Ea0JsZ2lYUVYwR1k/view

[2] https://www.youtube.com/watch?v=4aqUG5s-0Ec

[3] https://en.wikipedia.org/wiki/Microsoft_Minesweeper

[4] https://magnushoff.com/minesweeper/