Quiz 2 Report
Design & Analysis of Algorithms (E)

Class : E

Fadhlan Aulia	05111640000125
Rifqi Mukti W	05111640000076
Abraham Wong	05111740000065

Informatics Department
Faculty of Information Technology and Communication
Institut Teknologi Sepuluh Nopember (ITS)
Surabaya

2019

Quiz 2: Minesweeper.cpp, implementation of BFS

Link : https://github.com/rifqimw/Quiz-2-PAA

Opening

Minesweeper is a game that played by one player, played by opening all the field while evading the bombs by marking them with flags and clicking the safe field. The player will click on a square in the field to open the square. The square's contents are a bomb, a number, or a blank square.

a. If a player opens a square that contains a bomb, the player loses.
b. If the square contains a number, the number represents how many bombs are there in the square's current location on a 3 x 3 spaces.
c. If a player opens a square that contains a blank square, that is a safe zone, there is no bomb near there.

The winning conditions are when all the bombs are evaded and all other fields are opened.

Problem

In Mr. Done's field, there is a field with the area of **n** x **n**, filed with **k** number of bombs. To clean up the field, Danny, as the (not so) famous bomb defuser must defuse all the bombs, but he doesn't know where is the bomb, so he asked you to make a program to locate the bombs, replicating the system of a minesweeper. Create a program to simulate the conditions and clean the field.

Analysis

In this program of MineSweeper game, the algorithm we use is Breadth First Search, which implementation uses Queue data structure, where it'd be activated when a user input a coordinate point of a position wished to be revealed (in order to know whether there's a mine in such position or not, the algorithm will then iterate from the coordinate point to any other points which neighbor is a mine). We use Queue as its concept of First In First Out (FIFO) has a similarity towards the process of Breadth First Search. On the start of the BFS process, the coordinate which user has submitted will be added to a Queue. Then an iteration will be done while the Queue is not empty. The front of Queue will be submitted to a variable of pair of int **coord**, in case that **coord** has been accessed before, or has a mine, or is out of bounds from **board**, it'll be popped out of Queue and the iteration will restart with the next member of Queue. If **coord** doesn't meet the "if" statement in the sentence before, the process will check all surrounding neighbors of **coord**, if there is a mine, the content of **board** which has the position of **coord** will be changed to a marker of how many mines in the surrounding neighbors. If there is no mine, the content of **board** will be changed to '.' instead, indicating that there is no mine(s) surrounding it and it

has been visited. Then every neighbor on top of, on the right of, below and on the left of **coord** will be pushed into Queue if the corresponding neighbor hasn't been visited before, or it does not contain a mine. Then for the position of **coord** in **board**, the same position in array **isVisited** will be changed to '1', indicating that it has been visited, and the Queue will be popped. This process will keep happening until no visitable neighbors are found.

Pseudo-code of the function described above is :

```
void BFS (int x, int y):

    Queue.push(x,y)
    while(Queue not empty):

        coord = Queue.front()
        if coord is visited OR coord in board is a mine OR coord out of bounds:
            Queue.pop()
            continue to next iteration
        counter = 0

        //checking whether all neighbours is a mine

        if left neighbor is a mine : add to counter
        if upper left neighbor is a mine: add to counter
        if top neighbor is a mine: add to counter
        if upper right neighbor is a mine: add to counter
        if right neighbor is a mine: add to counter
        if bottom right neighbor is a mine: add to counter
        if bottom neighbor is a mine: add to counter
        if bottom left neighbor is a mine: add to counter
        if counter > 0 : board[coord.first][coord.second] = (char)counter
        else board[coord.first][coord.second] = '.'



        if (left neighbor is not visited AND is not a mine) Queue.push(left neighbor)
        if (top neighbor is not visited AND is not a mine) Queue.push(left neighbor)
        if (right neighbor is not visited AND is not a mine) Queue.push(left neighbor)
        if (bottom neighbor is not visited AND is not a mine) Queue.push(left neighbor)

        isVisited[coord.first][coord.second] = 1

        Queue.pop()      //pop coord
```

Implementation

1. **Universal variables**

```
#include <cstdio>
#include <cstdlib>
#include <queue>
using namespace std;

char board[30][30];
int isVisited[30][30];
int size;
queue <pair <int, int> > Queue;
```

There are 4 variables located outside of other functions, so that it can be accessed and modified easily by all functions, those are : **char board[30][30]** for minesweeper board, **int isVisited[30][30]** to indicate whether the corresponding position of board has been visited or not, **int size** to indicate size of board submitted later by the user, **queue <pair <int, int> > Queue** for the implementation of Breadth First Search.

2. **Int main() or Main Function**

```
int main() {
    int mines;
    printf("MINESWEEPER SEDERHANA\nInput nilai n untuk papan ukuran n x n yang diinginkan (maks 30): ");
    scanf("%d", &size);
    printf("Mau berapa mines (ranjau)? : ");
    scanf("%d", &mines);
    queue <pair <int,int> > minePost;
    while(mines--) {
        pair <int,int> minesLoc;
        minesLoc.first = rand() % size;
        minesLoc.second = rand() % size;
        minePost.push(minesLoc);
    }
    for(int i=0; i<size; i++) for(int j=0; j<size; j++) { board[i][j] = 'o'; isVisited[i][j] = 0; }
    while(!minePost.empty()) {
        board[minePost.front().first][minePost.front().second] = 'X';
        minePost.pop();
    }
    doMineSweep();
    printf("End! :)");
    return 0;
}
```

In the main function, the user will input the desired size of minesweeper board, how many mines that the user wants, and then an iteration will be done to determine the random location of each mine and it will be modded by “size” to avoid the number from growing out of bounds, it will then be pushed to a queue “minePost”, to later be used to change the content of board which position is in the minePost queue to ‘X’, indicating a mine. Initialization of the minesweeper board and isVisited array is also done in this main function. It will then call doMineSweep() function to perform the rest of its functions.

3. doMineSweep()

```
void doMineSweep() {
    int x, y;
    while(true) {
        print();
        if(isFinished()) return;
        printf("Mau mencoba klik di koordinat (x,y) yang mana? : ");
        scanf("%d %d", &x, &y);
        BFS(x, y);
    }
}
```

This function has an iteration to implement the BFS algorithm for as long as it hasn't been indicated as finished by isFinished() function. The iteration includes a printing function of the board and an input of the position that the user wishes to try accessing.

4. BFS(x,y)

```
void BFS(int x, int y) {
    pair <int,int> coord = make_pair(x,y);
    Queue.push(coord);

    while(!Queue.empty()) {
        coord = Queue.front();
        if(isVisited[coord.first][coord.second] || board[coord.first][coord.second]=='X') {
            Queue.pop();
            continue;
        } else if(coord.first > size-1 || coord.second > size-1 || coord.first < 0 || coord.second < 0) {
            Queue.pop();
            continue;
        }
        int counter = 0; char tempcount;

        if(board[coord.first][coord.second-1]=='X') counter+=1;
        if(board[coord.first-1][coord.second-1]=='X') counter+=1;
        if(board[coord.first-1][coord.second]=='X') counter+=1;
        if(board[coord.first-1][coord.second+1]=='X') counter+=1;
        if(board[coord.first][coord.second+1]=='X') counter+=1;
        if(board[coord.first+1][coord.second+1]=='X') counter+=1;
        if(board[coord.first+1][coord.second]=='X') counter+=1;
        if(board[coord.first+1][coord.second-1]=='X') counter+=1;

        if(counter>0) {
            tempcount = counter + '0';
            board[coord.first][coord.second] = tempcount;
        } else {
            board[coord.first][coord.second] = '.';
        }

        if(!isVisited[coord.first][coord.second-1] && board[coord.first][coord.second-1]!='X') Queue.push(make_pair(coord.first,coord.second-1));
        if(!isVisited[coord.first-1][coord.second] && board[coord.first-1][coord.second]!='X') Queue.push(make_pair(coord.first-1,coord.second));
```

The functionality of this function has been explained in the **"Explanation of Breadth First Search Usage".**

5. isFinished()

```
bool isFinished() {
    int status=0;
    for(int i=0; i<size; i++) for(int j=0; j<size; j++) if(board[i][j]=='o') status-=1;
    if(status==0) return 1;
    else return 0;
}
```

The function to determine whether the process should be ended or not. It will check through the board whether it still finds a visitable position or not, if it does not find more, it'll return '1' indicating the process should be finished.

6. print()

```
void print() {
    for(int i=0; i<size; i++)
    { for(int j=0; j<size; j++) {
        if(board[i][j]=='X') printf("o ");
        else printf("%c ", board[i][j]); }
    printf("\n"); }
}
```

The function to print all the contents of the minesweeper board while the game is still ongoing. In other words, since the user is not supposed to know the locations of the mines just yet, if the iteration encounters a mine, it'll print as 'o'. Other than that, it will iterate through the first index of the array and the second index of the array to get its content and then print it as a character type.

7. printall()

```
void printall() {
    for(int i=0; i<size; i++) {
        for(int j=0; j<size; j++) printf("%c ", board[i][j]);
        printf("\n");
    }
}
```

The function to print all the contents of the minesweeper boards when the game reaches its end. When the user has clicked on a mine or finished clicking all empty tiles aside from the mine, the program will print every content of minesweeper boards in **board** array, so the user can check the locations of the mines once more after the game is finished.

Solution

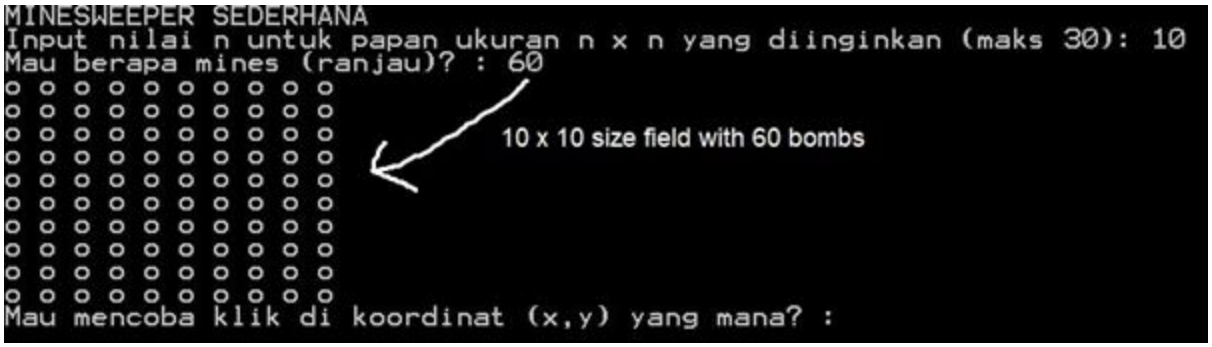
By the explanation above, we can make a code to help Danny. Here is the code that we suggest.

```cpp
#include <cstdio>
#include <cstdlib>
#include <queue>
using namespace std;

char board[30][30];
int isVisited[30][30];
int size;
queue <pair <int, int> > Queue;

void printall() {
    for(int i=0; i<size; i++) {
        for(int j=0; j<size; j++) printf("%c ", board[i][j]);
        printf("\n");
    }
}

void print() {
    for(int i=0; i<size; i++)
    { for(int j=0; j<size; j++) {
        if(board[i][j]=='X') printf("Q ");
        else printf("%c ", board[i][j]); }
    printf("\n"); }
}

void BFS(int x, int y) {
    pair <int,int> coord = make_pair(x,y);
    Queue.push(coord);

    while(!Queue.empty()) {
        coord = Queue.front();
        if(isVisited[coord.first][coord.second] || board[coord.first][coord.second]=='X') {
            Queue.pop();
            continue;
        } else if(coord.first > size-1 || coord.second > size-1 || coord.first < 0 || coord.second < 0) {
            Queue.pop();
            continue;
        }
        int counter = 0; char tempcount;

        if(board[coord.first][coord.second-1]=='X') counter+=1;
        if(board[coord.first-1][coord.second-1]=='X') counter+=1;
        if(board[coord.first-1][coord.second]=='X') counter+=1;
        if(board[coord.first-1][coord.second+1]=='X') counter+=1;
        if(board[coord.first][coord.second+1]=='X') counter+=1;
        if(board[coord.first+1][coord.second+1]=='X') counter+=1;
        if(board[coord.first+1][coord.second]=='X') counter+=1;
        if(board[coord.first+1][coord.second-1]=='X') counter+=1;

        if(counter>0) {
            tempcount = counter + '0';
            board[coord.first][coord.second] = tempcount;
        } else {
            board[coord.first][coord.second] = '.';
        }

        if(!isVisited[coord.first][coord.second-1] && board[coord.first][coord.second-1]!='X') Queue.push(make_pair(coord.first,coord.second-1));
```

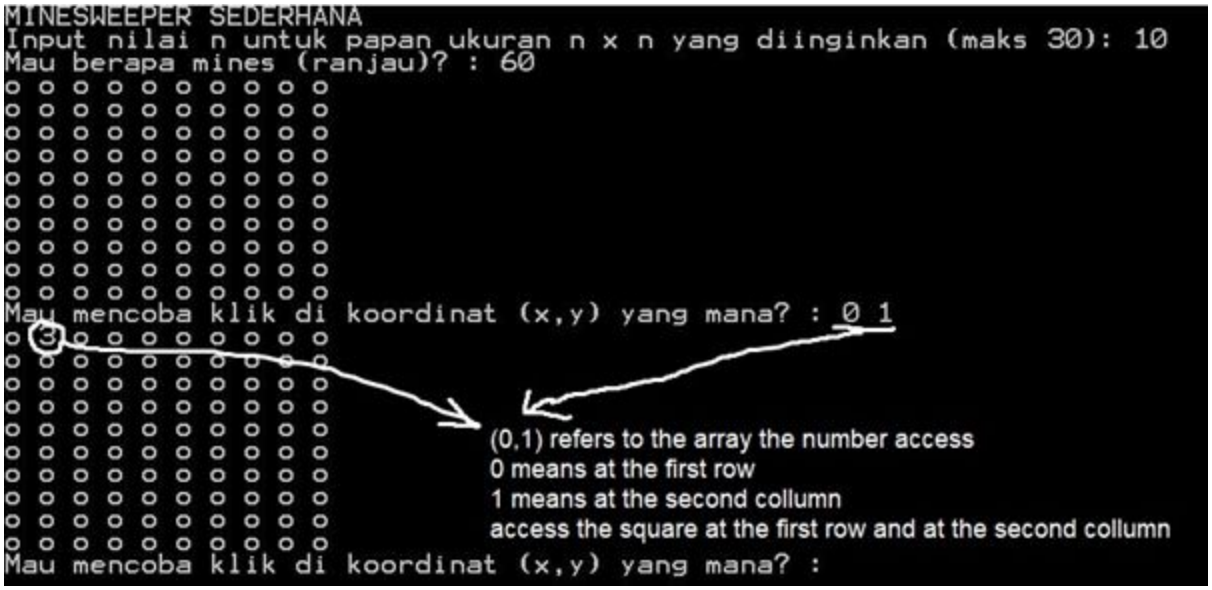
```
        if(!isVisited[coord.first-1][coord.second] && board[coord.first-1][coord.second]!='X') Queue.push(make_pair(coord.first-1,coord.second));
        if(!isVisited[coord.first][coord.second+1] && board[coord.first][coord.second+1]!='X') Queue.push(make_pair(coord.first,coord.second+1));
        if(!isVisited[coord.first+1][coord.second] && board[coord.first+1][coord.second]!='X' ) Queue.push(make_pair(coord.first+1,coord.second));

        isVisited[coord.first][coord.second] = 1;
        Queue.pop();
    }
}

bool isFinished() {
    int status=0;
    for(int i=0; i<size; i++) for(int j=0; j<size; j++) if(board[i][j]=='o') status-=1;
    if(status==0) return 1;
    else return 0;
}

void doMineSweep() {
    int x, y;
    print();
    while(true) {
        if(isFinished()) {
            printf("You win ?\n");
            return;
        }
        printf("Mau mencoba klik di koordinat (x,y) yang mana? : ");
        scanf("%d %d", &x, &y);
        if(board[x][y]=='X') {
            printf("You lose!\n");
            return;
        }
        BFS(x, y);
        if(!isFinished())print();
    }
}

int main() {
    int mines;
    printf("MINESWEEPER SEDERHANA\nInput nilai n untuk papan ukuran n x n yang diinginkan (maks 30): ");
    scanf("%d", &size);
    printf("Mau berapa mines (ranjau)? : ");
    scanf("%d", &mines);
    queue <pair <int,int> > minePost;
    while(mines--) {
        pair <int,int> minesLoc;
        minesLoc.first = rand() % size;
        minesLoc.second = rand() % size;
        minePost.push(minesLoc);
    }
    for(int i=0; i<size; i++) for(int j=0; j<size; j++) { board[i][j] = 'o'; isVisited[i][j] = 0; }
    while(!minePost.empty()) {
        board[minePost.front().first][minePost.front().second] = 'X';
        minePost.pop();
    }
    doMineSweep();
    printall();
    printf("End! :)");
    return 0;
}
```

How to Use

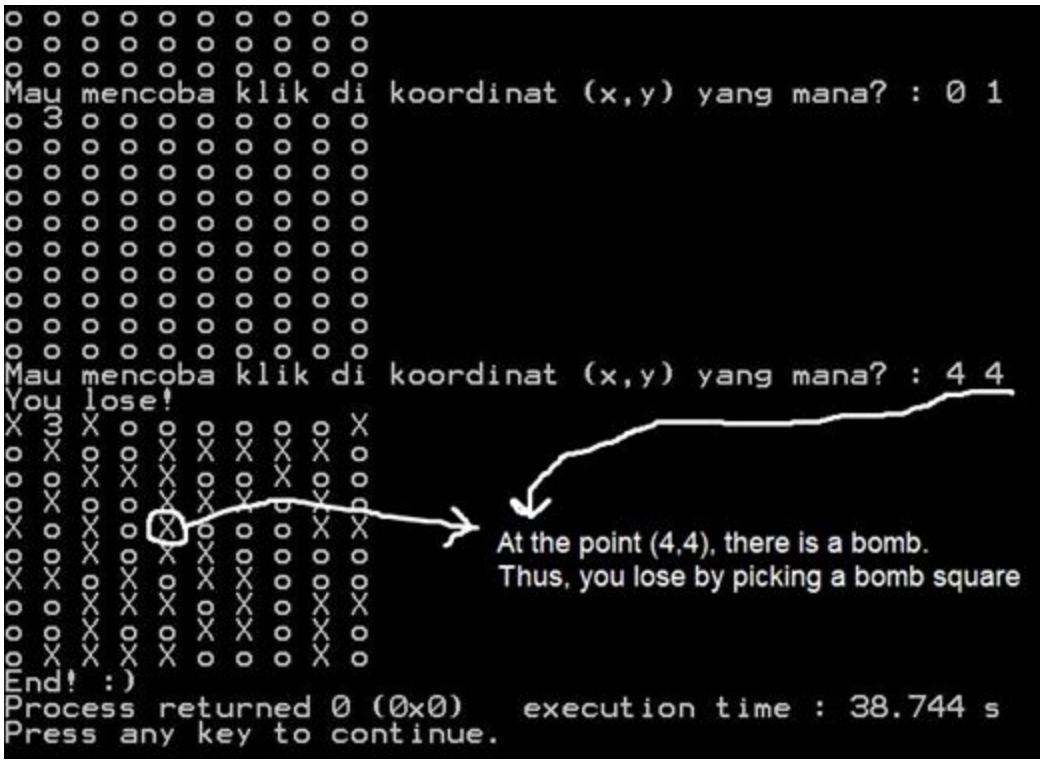
The user asked to give inputs **n** to state the area of the field, and **ranjau** to state the number of bombs planted in the field.



The program then generates **n x n area** field with **ranjau** bombs. To check a box, insert two numbers to check the square of that box.



In this case, the user inputs a zero and a one. Imagine that the value given is (0,1). Now, (0,1) doesn't refer to the cartesian places, but more to which column and row is a square. Add each value by one, we get (1,2), as in:
1 → the first row
2 → the second column
Check the first row and second column of the field, open the square. We now get a number '3', which means that there are 3 bombs near the 3 x 3 area of the square.



In case of losing, the map then will be revealed, the bombs and the numbers. However, if you win, there only will state that you win (because you evade all the bombs).
Let's look at this directory (4,4) → increment to (5,5)
5 → the 5th row
5 → the 5th column
Check the fifth row and the fifth column, open the square. That happens to be a bomb. Then, you lose the game.

PLEDGE DECLARATION

“By the name of Allah (God) Almighty, herewith we pledge and truly declare that we have solved quiz 2 by ourselves, didn’t do any cheating by any means, didn’t do any plagiarism, and didn’t accept anybody’s help by any means. We are going to accept all of the consequences by any means if it has proven that we have been done any cheating and/or plagiarism.”

Surabaya, 30 March 2019

Rifqi Mukti Wicaksana	**Fadhlan Aulia**	**Abraham Wong**
05111640000076	**05111640000125**	**05111740000065**