# Algorithms and Data Structures

# Minesweeper Game

# Report

## I. Introduction

Our group include 2 members: Nguyễn Vĩnh Trí (ITDSIU19021) and Trương Công Trung (ITITIU19059). In this project, the instructor required each group to make a game with algorithms in it. And we decided to demo MineSweeper.

## II. Content

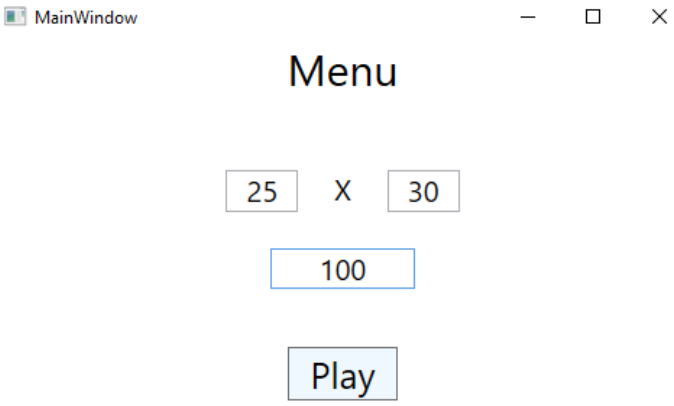
a. GUI

First, for the GUI, we used Winform from Microsoft to make the Menu and MineSweeper's board. We used "UniformGrid" which is a type of layout that has columns and rows. It made us easier to locate the tiles of the board by using x&y coordinate. For the tiles, we used button. Unfortunately, the button doesn't have any properties to locate itself. So, we made a new class, and extended it.

```
public class IButton : Button
{
    public int x { get; set; }
    public int y { get; set; }
    public bool IsRightClicked = false;
}
```

When the button is clicked by a right click, I will turn the color of the button into red, so players will know that they have checked for the bomb. We have a bool property to check if the player right clicked 2 times.

We also have a Reset button in case of players want to play the game again.
This is the menu of the game. In the menu part, we let players choose their options for the game (numbers of columns, rows, and bombs).



b. Data structures

We have a class “Pos” to save the position of the cell, return a position if we needed.

```
12 references | ssumti, 3 days ago | 1 author, 1 change
public class Pos
{
    private int x;
    private int y;
    14 references | ssumti, 3 days ago | 1 author, 1 change
    public int GetX()
    { return x; }
    14 references | ssumti, 3 days ago | 1 author, 1 change
    public int GetY()
    { return y; }
    0 references | ssumti, 3 days ago | 1 author, 1 change
    public void ReX(int t)
    { x = t; }
    0 references | ssumti, 3 days ago | 1 author, 1 change
    public void ReY(int t)
    { y = t; }
    3 references | ssumti, 3 days ago | 1 author, 1 change
    public Pos(int x, int y)
    {
        this.x = x;
        this.y = y;
    }
}
```

We also have a “Queues” class to implement BFS algorithms.

```
3 references | ssumti, 3 days ago | 1 author, 1 change
public class Queues
{
    private Pos[] ele;
    private int front;
    private int rear;
    private int max;
    1 reference | ssumti, 3 days ago | 1 author, 1 change
    public Queues(int size)
    {
        ele = new Pos[size];
        front = 0;
        rear = -1;
        max = size;
    }
    2 references | ssumti, 3 days ago | 1 author, 1 change
    public void Enqueue(Pos x)
    {

        if (rear != max)
        {
            ++rear;
            ele[rear] = x;
        }
    }
    1 reference | ssumti, 3 days ago | 1 author, 1 change
    public bool IsEmpty()
    {
        if (front > rear)
            return true;
        return false;
    }
```

```
    0 references | ssumti, 3 days ago | 1 author, 1 change
    public int GetMax()
    { return max; }
    0 references | ssumti, 3 days ago | 1 author, 1 change
    public int GetFront()
    { return front; }
    0 references | ssumti, 3 days ago | 1 author, 1 change
    public int GetRear()
    { return rear; }
    0 references | ssumti, 3 days ago | 1 author, 1 change
    public Pos GetPos(int x)
    {
        return ele[x];
    }
    1 reference | ssumti, 3 days ago | 1 author, 1 change
    public Pos Dequeue()
    {
        if ((rear == front - 1) || (rear == -1))
        {
            return null;
        }
        else
        {
            Pos pos = ele[front];
            ++front;                            // not empty
            return pos;
        }
    }
    0 references | ssumti, 3 days ago | 1 author, 1 change
    public void Clear()
    {
        for (int i = 0; i <= rear; i++)
        {
            ele[i] = null;
        }
        front = 0;
        rear = -1;
    }
```

c. Algorithms

We use “BFS” to handle the cells, which have not opened, for the spread action when the cell button is clicked.

By queuing the tiles, and spread them if the are qualified, we continue to spread if there are any tile left in the queue.

We prior to tiles that surround the clicked tile. And then, the next priority is tiles that have 1 distance to the clicked tile and so on.

d. Game process

At the beginning of the game, we random the number of bombs that players input.

```
private void RandomBomb(int x, int y, int b)
{
    Random random = new Random();

    int m = x * y;
    Pos[] temp = new Pos[m];
    for (int c = 0, i = 1; i <= x; ++i)
        for (int j = 1; j <= y; ++j)
            temp[c++] = new Pos(i, j);

    int cnt = 1;
    while (cnt <= b)
    {
        int p = random.Next(1, m);

        int h = temp[p].GetX();
        int w = temp[p].GetY();

        for (int i = p; i < m - 1; ++i) temp[i] = temp[i + 1];
        m--;

        ++cnt;
        a[h, w] = -1;

        for (int i = 0; i < 8; ++i)
            if (a[h + posX[i], w + posY[i]] != -1) a[h + posX[i], w + posY[i]]++;
    }
}
```

After setting the position of the bomb, as each bomb, we plus 1 to each tile around the bomb. And the bomb position is marked with -1.

Next, for each click, we check if the tile that the player clicked is the bomb or not.

If it is a bomb, then game over.

Else, if they hit a tile with number, then just spread 1 tile that they clicked.

If they hit an empty tile, then we use BFS to spread for new empty tiles.

```
if (a[button.x, button.y] == -1)
{
    TurnOnAllBomb();
    GameOver("lose");
}
else if (a[button.x, button.y] == 0)
{
    var t = new Pos(button.x, button.y);
    q.Enqueue(t);
    buts[t.GetX(), t.GetY()].IsEnabled = false;
    while (!q.IsEmpty())
    {
        var temp = q.Dequeue();
        for (int i = 0; i<=7; ++i)
        {
            if (check[temp.GetX() + posX[i], temp.GetY() + posY[i]] == true)
                if (a[temp.GetX()+posX[i],temp.GetY()+posY[i]] == 0)
                {
                    check[temp.GetX() + posX[i], temp.GetY() + posY[i]] = false;
                    buts[temp.GetX() + posX[i], temp.GetY() + posY[i]].IsEnabled = false;
                    var xtemp = new Pos(temp.GetX() + posX[i], temp.GetY() + posY[i]);
                    q.Enqueue(xtemp);
                }
                else
                if (a[temp.GetX() + posX[i], temp.GetY() + posY[i]] != 0 && (a[temp.GetX() + posX[i], temp.GetY() + posY[i]] != -1))
                {
                    buts[temp.GetX() + posX[i], temp.GetY() + posY[i]].Content = a[temp.GetX() + posX[i], temp.GetY() + posY[i]];
                    check[temp.GetX() + posX[i], temp.GetY() + posY[i]] = false;
                    buts[temp.GetX() + posX[i], temp.GetY() + posY[i]].Click += Button_Click; //disable
                    buts[temp.GetX() + posX[i], temp.GetY() + posY[i]].MouseRightButtonDown += MouseRightButtonDown;
                }
        }
    }
}
else
{
    button.Content = a[button.x, button.y];
    check[button.x, button.y] = false;
    button.Click += Button_Click;
}
```

The game continues until players hit bomb, or no tile is not clicked.

e. List of functions

- MainGame.BeginGameProcess.UI() : this function initialize the board of tiles in MineSweeper.

```
private void Ui(StackPanel layout, int x, int y)
{
    grid = new UniformGrid()
    {
        Columns = y,
        Rows = x,
    };
    for (int i = 1; i<=x; ++i)
        for (int j = 1; j<=y; ++j)
        {
            IButton b = new IButton()
            {
                Width = 20,
                Height = 20,

                y = j,
                x = i,
            };
            Grid.SetColumn(b, j);
            Grid.SetRow(b, i);
            b.Click += (object sender, RoutedEventArgs e) => B_Click(sender, e, b);
            b.MouseRightButtonDown += (object sender, MouseButtonEventArgs e) => B_MouseRightButtonDown(sender, e, b);
            buts[i, j] = b;
            grid.Children.Add(b);
        }
    layout.Children.Add(grid);

}
```

We created the Grid and the Button for the game.

- MainGame.BeginGameProcess.Reset() : this function used to reset all the properties like the array that we used to check the bomb and number the tile, and the array that we used to check if the tile is checked.

```
private void Reset(int x, int y)
{
    for (int i = 1; i <= x; ++i)
        for (int j = 1; j <= y; ++j)
        {
            a[i, j] = 0;
            check[i, j] = true;
        }
}
```

- MainGame.BeginGameProcess.RandomBomb() : this function used to random bombs. We had presented this above.

```
private void RandomBomb(int x, int y, int b)
{
    Random random = new Random();

    int m = x * y;
    Pos[] temp = new Pos[m];
    for (int c = 0, i = 1; i <= x; ++i)
        for (int j = 1; j <= y; ++j)
            temp[c++] = new Pos(i, j);

    int cnt = 1;
    while (cnt <= b)
    {
        int p = random.Next(1, m);

        int h = temp[p].GetX();
        int w = temp[p].GetY();

        for (int i = p; i < m - 1; ++i) temp[i] = temp[i + 1];
        m--;

        ++cnt;
        a[h, w] = -1;

        for (int i = 0; i < 8; ++i)
            if (a[h + posX[i], w + posY[i]] != -1) a[h + posX[i], w + posY[i]]++;
    }
}
```

- MainGame.BeginGameProcess.B_Clicked() : this function used to process when players clicked tiles. We also implement BFS in this function.

## III. Conclusion

In conclusion, the game MineSweeper is made. The algorithms that we used is BFS and queue data structures.

This is the simplest sample for implement the BFS algorithm in game developers.

There are many games using the graph for saving the data, item, or the processing some feature in game processing.

Otherwise, the graph theory like BFS, DFS is the base of the popular AI practice such as Chess bot, Chinese chess bot, MOBA game bot. Using DFS for the long-term prediction in game sense, game event.

## IV. Percentage of contribution