Minesweeper SDK

1 Introduction

1.1 Development Strategy

I developed this project by splitting the presentation of the game (front-end) by the game logic (back-end). By using this strategy, I developed a sort of SDK (see SDK folder) and actually we can use it with different .Net environment: we don't are "locked" around UI elements or components.
With this SDK, we can implement the same game, for example, as WPF application, WCF service, ASP.NET MVC application and so on.
The integration of this "backend" is very easy!

1.2 Tech Specifications and Requirements

.NET Framework 4.5
Visual Studio 2013 Community
NUnit framework
lowerCamelCase coding style

1.3 Game logic

In this paragraph we are not going to discuss about the game instruction (http://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=29&page=show_problem&problem=1130), but we will talk only how the problem of the game can be solved.
I think that the main issue of his game is finding the adjacent mines and the adjacent empty cells around an item. In both cases we have to consider that at most each cell (item) could be surrounded by eight items (cells). If we consider a 4x4 grid (matrix) a cell C defined by a X,Y coordinates it will be surrounded by the following cells:

	x			
y	C1 (X-1,Y-1)	C2 (X,Y-1)	C3 (X+1,Y-1)	
	C4 (X-1,Y)	C(X,Y)	C5 (X+1,Y)	
	C6 (X-1,Y+1)	C7 (X,Y+1)	C8 (X+1,Y+1)	

In other words we can find the adjacent cells of an item by translating a point on the x and y axis.

2 SDK Classes and Structure

2.1 Introduction

Before we discuss of the classes and structure of this SDK, we can assert that we can see the board game as a Grid with N cols and N rows and where each item (cell) it will be uniquely defined by a pair values: a Row and a Col

2.2 SDK Classes

2.2.1 MinesweeperItemCellDefinition

It represents a cell, defined by a pair (int) values that indentify row and colum.
To create a new instance is required to pass row and col as parameters of the constructor:

```
public MinesweeperItemCellDefinition(int row, int col)
```

2.2.2 MinesweeperItem

It represents the "atomic" element of the grid game.
To create a new instance is required to pass a MinesweeperItemCellDefinition instance as parameter of the constructor.

```
public MinesweeperItem(MinesweeperItemCellDefinition cell)
```

Within the class is exposed an enum that defines the possible kind of an item.

```
public enum MinesweeperItemType{
        MinesweeperItemType_None = 0,
        MinesweeperItem_Empty = 1,
        MinesweeperItem_MineWarning = 2,
        MinesweeperItem_Mine = 3
    }
```

Each item can be defined as
- Mine
- Warning Mines Indicator
- Empty cell
- None (default)

MinesweeperItem has the following attributes:

`public Object tag { get; set;}`	useful to store user info data
`public MinesweeperItemType type { get; set;}`	allows to set the item type
`public Object value { get; set;}`	allows to store a value
`public MinesweeperItemCellDefinition cell { get; protected set;}`	allows to set cell coordinates

2.2.3 MinesweeperMineGenerator

This is an helper class and it helps us to generate a random number between 1 and number of cells (generally rows*cols or however a total elements of a matrix).
How to use:
1) create a new instance of the class by passing into the constructor the number of the

cells

```
public MinesweeperMineGenerator(int cells)
```

2) to make a random mine cell, call the method make

```
public int make()
```

2.2.3 MinesweeperGrid

This class represents the core of the SDK an it allows to make a grid game system with rows*cols items, and it helps, automatically, to define and dispose, randomly, mines on the grid.
To create a new instance you can use two constructors:

```
public MinesweeperGrid(int rows, int cols)
public MinesweeperGrid(int rows, int cols, int maxMines)
```

As you can see, in both cases is needed to pass rows and cols of the grid (matrix) and optionally you can also define how many mines you want to place on your grid:
If you don't pass the number of mines, the system will place rows-1 mines.
In a next version maybe we can suppose to implement a calculation algorithm, for instance, based on the number of the cell.

Class exposes the following behavior:

Properties

`public int rows { get; protected set; }`	returns the number of grid rows
`public int cols { get; protected set; }`	returns the number of grid columns
`public int maxMines { get; protected set; }`	returns then number of max mines for the games
`public List<MinesweeperItem> items { get; protected set; }`	returns a list of MinesweeperItem: it represents all items (cells) of the game grid

Public methods

`public void makeGrid()`	See 2.4 for further details, however it allows to make the game grid and it, automatically and randomly, will place the mines
`public void openAllBlocks()`	This method is useful when we are in a "game over" mode it will "open" all cells with a proper value (empty, mine number warning, mine)
`public void setAdjacentCells(MinesweeperItem item)`	With this method you can discover a) how many adjacent mines are around the input method item b) how many adjacent empty cells are around the input method item
`public MinesweeperItem findItemAt(MinesweeperItemCellDefinition cell)`	Allows to find an item by cell coordinates (row and col)
`public void evaluateItem(MinesweeperItem item)`	With this method you can implement all the game behavior If the item will be a mine, then will be raised a game over event and all cells will be opened,

	otherwise it will find all the item's adjacent cells

2.3 Event System

MinesweeperGrid works around a sort of messaging system implemented with a series of events exposed by the class.
MinesweeperGrid exposes the following events defined by delegates methods:

`itemAdded`
this event will be raised when a new item (cell) will be added.

```
public event MinesweeperItemAdded itemAdded;
public delegate void MinesweeperItemAdded(MinesweeperItem item);
```

`itemMineAdded`
this event will be raised when a "mine" item will be added.

```
public event MinesweeperItemMineAdded itemMineAdded;
public delegate void MinesweeperItemMineAdded(MinesweeperItem item);
```

`loadingCompleted`
this event will be raised when the grid game will be completely loaded.

```
public event MinesweeperLoadingCompleted loadingCompleted;
public delegate void MinesweeperLoadingCompleted(List<MinesweeperItem> items);
```

`cellOpeningCompleted`
this event will be raised when a new item will be open as empty or mine cell

```
public event MinesweeperCellOpeningCompleted cellOpeningCompleted;
public delegate void MinesweeperCellOpeningCompleted(MinesweeperItem item);
```

`gameOver`
this event will be raised when item is a mine

```
public event MinesweeperGameOver gameOver;
public delegate void MinesweeperGameOver(MinesweeperItem item);
```

`errorOccurred`
this event will be raised when an error occurs

```
public event MinesweeperError errorOccurred;
public delegate void MinesweeperError(Exception ex);
```

2.4 How to use

This SDK can be implemented as class set of a .NET project in different kind of solutions:
To implement the game, we suppose that you will make a grid system with some "clickabale" UI elements such as a Button.
However, to simplify the SDK comprehension, in this example we will talk about a Button of a WPF project (included as part of this SDK) and will make a buttons grid with an UniformGrid.
To implement the game you have to follow these step:

<u>1) Declare a MinesweeperGrid variable and Create an instance by using a proper constructor:</u>

```
private MinesweeperGrid gameGrid;
....
....
....
//we will create a grid 9x9 with 12 mines
gameGrid = new MinesweeperGrid(9, 9,12);
```

<u>2) Implement the event handler for the MinesweeperGrid   instance, at least for:</u>

```
public event MinesweeperCellOpeningCompleted cellOpeningCompleted;
public delegate void MinesweeperCellOpeningCompleted(MinesweeperItem item);

public event MinesweeperLoadingCompleted loadingCompleted;
public delegate void MinesweeperLoadingCompleted(List<MinesweeperItem> items);

public event MinesweeperError errorOccurred;
public delegate void MinesweeperError(Exception ex);

public event MinesweeperGameOver gameOver;
public delegate void MinesweeperGameOver(MinesweeperItem item);
```

example:

```
gameGrid.loadingCompleted+=gameGrid_loadingCompleted;
gameGrid.errorOccurred+=gameGrid_errorOccurred;
gameGrid.cellOpeningCompleted+=gameGrid_cellOpeningCompleted;
gameGrid.gameOver +=gameGrid_gameOver;
```

<u>3) Call the makeGrid method</u>

```
gameGrid.makeGrid();
```

<u>4) When the loadingCompleted event will be raised, you can load your grid by using the items grid:</u>

```
private void gameGrid_loadingCompleted(List<MinesweeperItem> items){
       //this event will be raised from the makeGrid method
       makeButtonsGrid();
}
```

```
private void makeButtonsGrid() {
       //for each grid item add a button on panel form
       List<MinesweeperItem> items= gameGrid.items;
       gamePanel.Columns = gameGrid.cols;
       gamePanel.Children.Clear();
       foreach (MinesweeperItem item in items){
              gamePanel.Children.Add(getGridButton(item));
       }

}

private Button getGridButton(MinesweeperItem item){
       //creates a button
       Button button = new Button();
       //stores the button on the item tag
       item.tag = button;
       //stores the item on the tag button
       button.Tag = item;
       button.Content = ".";
       button.Width = 35;
       button.Height = 35;
       button.Click += gridButton_Click;
       return button;
}
```

5) When an UI element will be clicked, you have just to call the evaluateItem method as follow:

```
   private void gridButton_Click(object sender, RoutedEventArgs e){
            Button button = (Button)sender;
            MinesweeperItem item = (MinesweeperItem)button.Tag;
            gameGrid.evaluateItem(item);
   }
```

6) Implement the presentation behavior of each UI element, within the cellOpeningCompleted and gameOver events

```
  private void gameGrid_gameOver(MinesweeperItem item) {
       MessageBox.Show(
                     "Oh no!\nI'm a mine :(", "GAME OVER",
                     MessageBoxButton.OK,
                     MessageBoxImage.Warning
                     );
}

private void gameGrid_cellOpeningCompleted(MinesweeperItem item){
       //gets the button from the item tag (see getGridButton method)
       Button button = (Button)item.tag;
       switch (item.type) {
              case MinesweeperItemType.MinesweeperItem_Empty:
                     button.Content = "";
                     break;
              case MinesweeperItemType.MinesweeperItem_Mine:
                     button.Content = "*";
                     break;
              case MinesweeperItemType.MinesweeperItem_MineWarning:
                     button.Content = item.value.ToString();
                     break;
              case MinesweeperItemType.MinesweeperItemType_None:
                     break;
             }

             //if items is opened, remove the click event handler from the button
             if (item.type != MinesweeperItemType.MinesweeperItemType_None) {
                 button.Click -= gridButton_Click;
             }

         }
```

2.5 Improvements

There are a lots of improvements that we can develop to gain the "gamification" and to give more fun!

Here some ideas:

1) Game difficulty:
- beginner: 9x9 with 10 mines
- intermediate:16x15 with 40 mines
- expert:30x16 with 99 mines
- custom

2) Game timing
3) Store data into database by user, to implement statistics
4) Sound and video effects
5) A "realistic" UI

2.6 Tests

Within the .NET solution, there is a Test project, that implement some basic Unit Test on the MinesweeperGrid.
Obviously tests could be developed much better, but at this stage and for this purpose they give an idea of how the project could be evolved.
All the unit tests are developed by using NUnit Framework.