UNIVERSITÉ ANTONINE
Faculté d'ingénieurs en Informatique,
Multimédia, Réseaux & Télécommunications

Fireworks

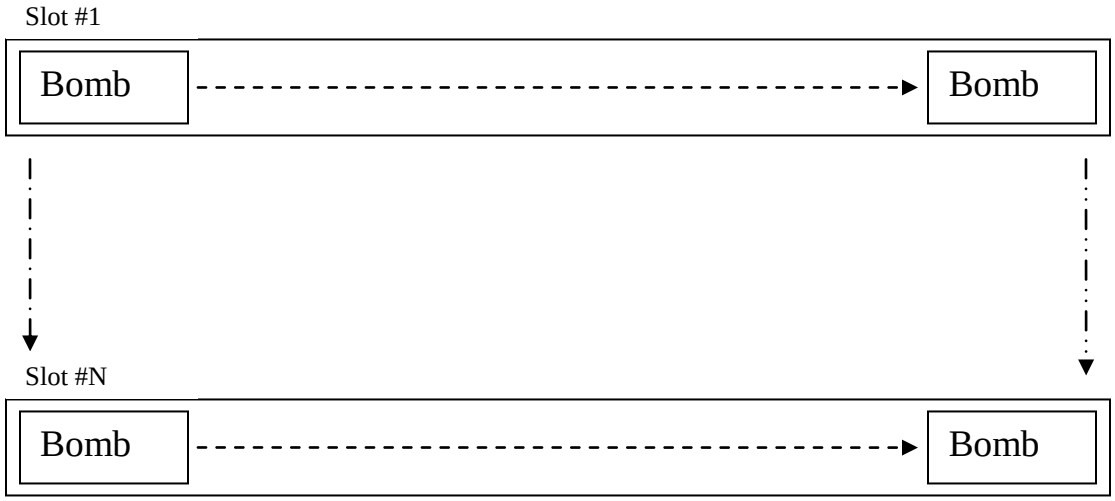
Matière : VB.NET et ASP.NET

Effectué par :	NOM Prénom	INF#
	MATTA Elie	Privacy
	HALLAGE Rabih	applied
	Et al.	

We separated our project into two main categories, the application category and the implementation category through the COM port.

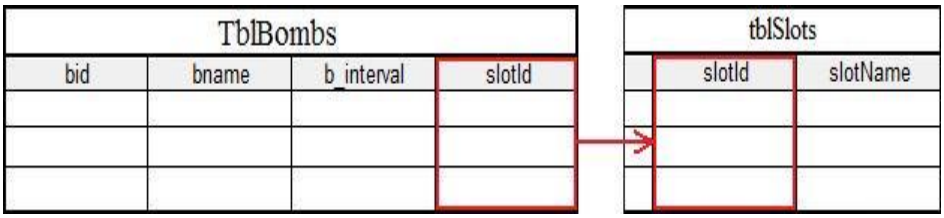
In this document we shall review the entire application category and its functionality through Visual studio 2005.

The project consist to create slots, this program can support 1→n slot(s);
each slot can contain 1→m bomb(s).

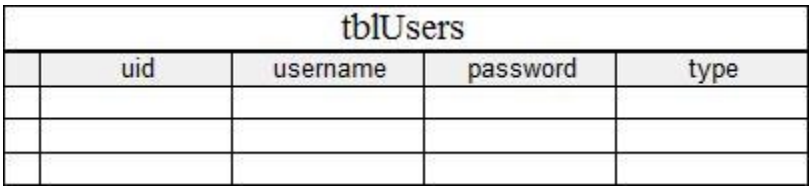


So, this is our vision and the "main idea" from which we started to develop our program.

0 – The Database (Class):



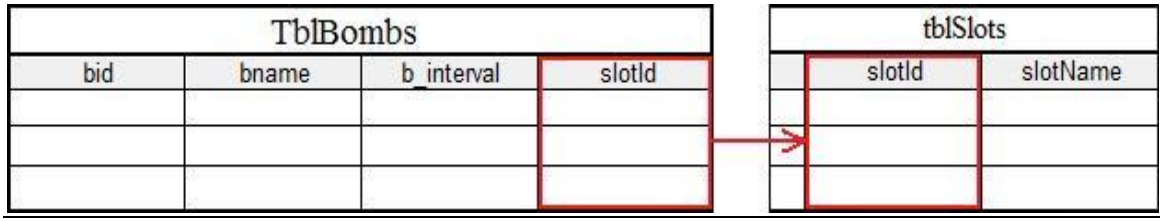
Our database consists of three tables,
tblUsers:



This is the users' tables, it is a simple structure it contains four (4) fields:

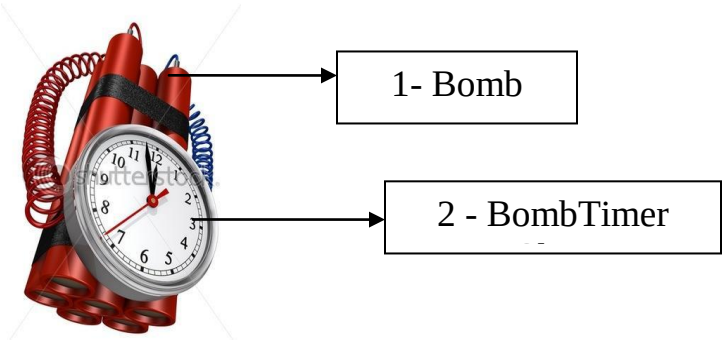
1. **uid** **:** Primary key
2. **username** **:** The user chosen UserId / Name
3. **password** **:** Password field
4. **type** **:** the type of the user, it can be either **regular** or **administrator**

TblBombs and tblSlots:



These are the two main tables in our application, as we mentioned before we kept our program "simple" as we can, we can notice from the snapshot above that these two tables are **linked by reference** using the **primary key (slotId)** from the table **tblSlots** and a **foreign key** in the **tblBombs** table.
Each slot can contains as many bombs as we want and we can determine each bomb to which slot belongs by the reference of the **slotId** in the both tables.

1 – The Bomb (Class):



Here we have the "**Bomb Class**" which contains the following parameters:

1. **p_bombInterval** (Integer) : time to explode
2. p_id (Integer) : each bomb have her own unique id
3. p_slot (Integer) : the id of the slot for which the bomb belongs.
4. p_name (String) : the name of the bomb
5. **timerObj** (bombTimer) : bombTimer Object(discussed later)

And we have added other parameters (non bolded) for controlling the application.

The "**Bomb Class**" is formed by the following methods:

1. `Public Function getTmr() As bombTimer`
   This method will return the TimerObj
2. `Public Sub setName(ByVal newName As String)`
   This method will set the name of the bomb
3. `Public Function getName() As String`
   This method will return the current name of the selected bomb
4. `Public Function getId() As Integer`
   This method will return the Id of the bomb
5. `Public Function getSlotId() As Integer`
   This method will retun the slot id assigned to the bomb
6. `Public Sub setSlotId(ByVal slotid As Integer)`
   In case we want to change the bomb's location and "transfer" it into another slot,this method will let us to assign a new slot id for the selected bomb.
7. `Public Function startTimer() As Boolean`
   This method will start the timer , once the timer is started it will return a "true" value.

`Public Function stopTimer() As Boolean`
Unlike the previous method, this one is to stop the timer of this bomb.

2 –BombTimer (Class) :

Inside the bomb class we have "Defined" the **BombTimer Class**, this class is defined by the following parameters :

- p_tmrInterval (Integer): timer interval
- p_isEnabled (Boolean): initial state
- p_currentImg (Windows.Forms.PictureBox):
  this parameter is a picture box component which will hold the images before and after explosion.
- p_tmrObj (System.Windows.Forms.Timer):
  The timer component.

The **BombTimer Class** is formed by the following methods:

- Public Sub New(ByVal tmrInterval As Integer, ByVal isEnabled As Boolean, ByRef currentImg As Windows.Forms.PictureBox) :
  →This is the default construct of this class,inside this construct we have define a "special line" of code :
  AddHandler p_tmrObj.Tick, AddressOf doAction
  this line is responsible of calling the "detonation action" called "doAction",this action is called once the timer's interval times up.
- Public Function getTmrObj() As System.Windows.Forms.Timer :
  returns **BombTimer** Object
- Public Function getInterval() As Integer :
  returns the interval of the current BombTimer Object
- Public Sub setInterval(ByVal newInt As Integer) :
  Sets a new interval value
- Public Function getImg() As Windows.Forms.PictureBox :
  return the image place holder from the "p_currentImg" attribute (default pic: before or after the explosion stored in My.Resources)
- Public Sub setImg(ByVal img As Windows.Forms.PictureBox) :
  set a new images in "p_currentImg" attribute
- Private Sub doAction(ByVal myObject As Object, ByVal myEventArgs As EventArgs) :
  the "detonation method" is called on the timer's time out.

3 –ClsConnection (Class):

This class holds the methods to manipulate the database, such as add, delete, update, etc…

```
Public Class ClsConnection
    Public p_conString As String
    Public Connection As OleDb.OleDbConnection
…
…
…
End Class
```

We will not discuss all the methods of this class;however we will list the methods used to manipulate the database:

```
1. Public Sub doUpdate(ByVal query As String)
2. Public Sub doInsert(ByVal query As String)
3. Public Sub doDelete(ByVal query As String)
```

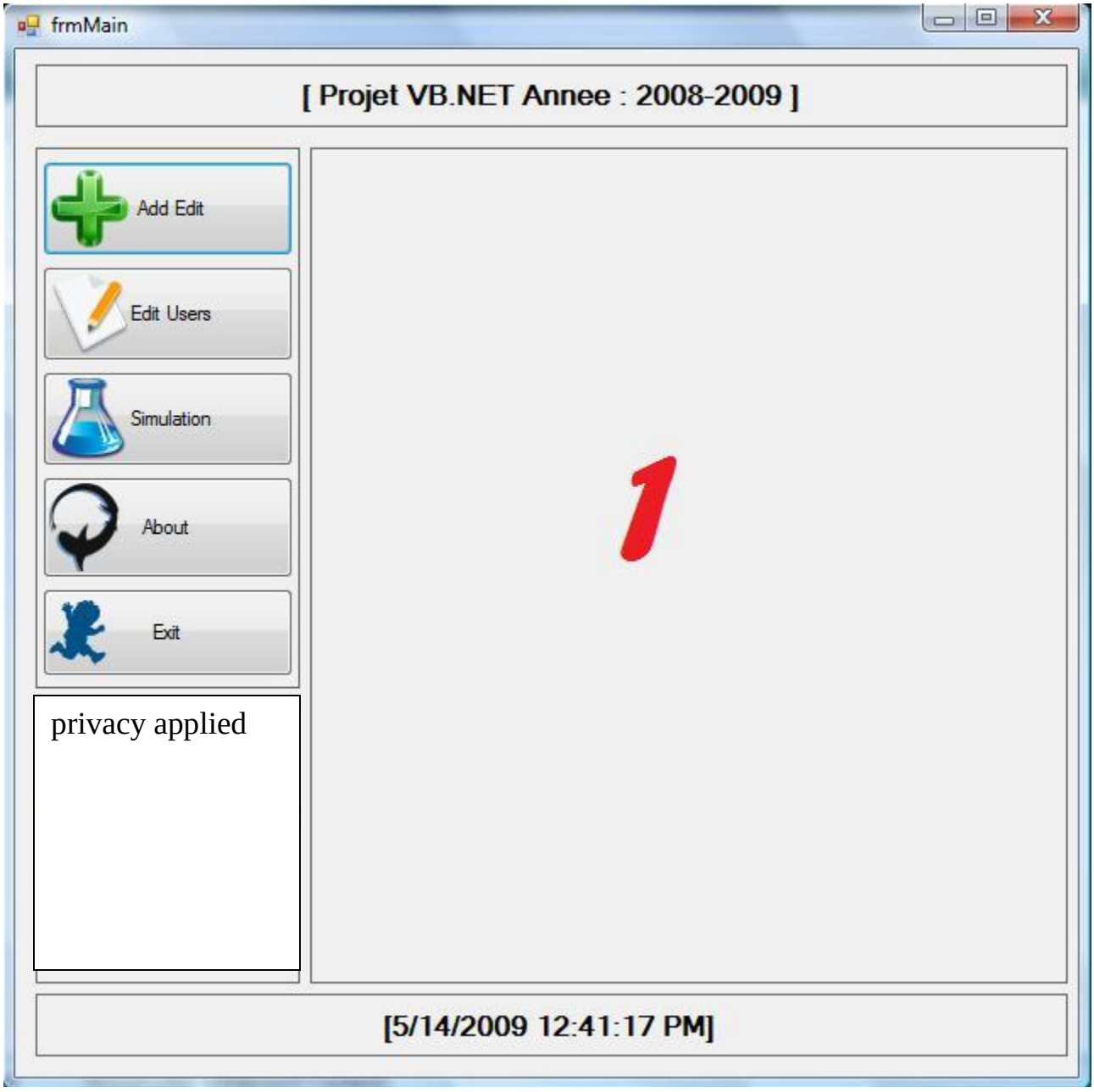
on the otherhand we will go in detail in some methods who have (for some levele) a main role in our application:

1. `Public Function getSlotsId() As ArrayList :`
   this method will return an ArrayList containing all the available **slotId**'s from the **tblSlots** table
2. `Public Function doLogin(ByVal t_username As String, ByVal t_password As String) As Boolean :`
   This method will preform the login to the application,it will return **true** in case the login was successful ,otherwise **false**.

4 –FrmLogin (Form):

A simple form login is included in this application to give the program a little bit of security.
How this form works?
Simply it compares the username and password entered by the user with the existing values inside the database, if the values are "equal" the login form must load the **frmMain** (the main form), otherwise it will display a login error message.

5 –FrmMain (Form):

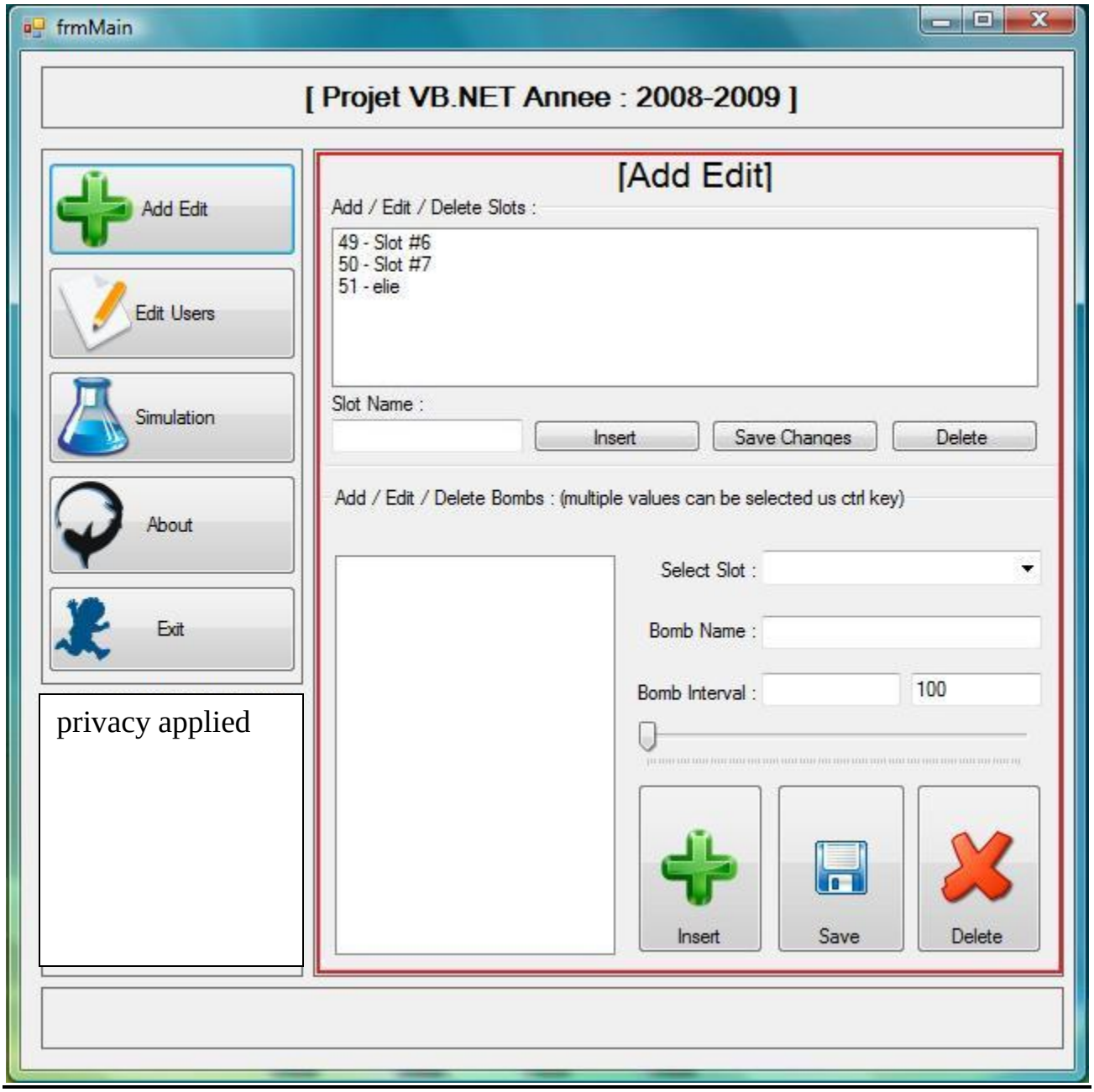


This is the main form (frmMain) of the application; this form contains the menu to the other forms.

- 5-1 Add Edit (button): it will load the **frmAddEdit** form.
  The frmAddEdit form will let us add/edit the bombs and slots.
- 5-2 Edit Users (button): it will load the **frmUserEdit**.
  This form can be accessed only by the users who have "admin" type accounts.
- 5-3 Simulation (button): will load the **frmViewExp** or in other words the simulation form, in this form we can view the "explosions" of the bombs added previously in their own respective slots.
- 5-4 About (button): loads the about form.
- 5-5 Exit (button): unload all forms and exit from application.

PS: all forms are loaded inside the zone #1 marked in the picture above.

6 –FrmAddEdit (Form):




```
Private arr_selectedItems As New ArrayList
Private arr_selectedBombItems As New ArrayList
Private LastIndexSelected As Integer
```

Inside the form declarations we can find the above attributes,these attributes are declared private and they are very important.

- `arr_selectedItems (ArrayList) :`
  `this arraylist will store the ids of the selected slots from the listbox,later we use these ids to save or delete many slots at the same time.`
- `arr_selectedBombItems (ArrayList):`
  `we will use it for the same reason as the above array,but this array will store the ids of the` **`bombs`**`.`
- `LastIndexSelected (Integer) :`
  `This variable will hold the last index of the slot/bomb listbox.`
- `Private Function getId(ByVal str As String) As String:`
  `this function will return the id of the selected slot(from the listbox).`
- `Private Function getSelectedId(ByVal listbx As Windows.Forms.ListBox) As Integer:`
  `this function will return (a single) id of the selected slot,this function will be used in the next function in a loop to fill an array.`
- `Private Function getGrpSelectedIds(ByVal lstbox As Windows.Forms.ListBox) As ArrayList:`
  `this function will fill the array with the ids of the selected slots from the listbox.`
- `Private Function getSelectedName(ByVal listbx As Windows.Forms.ListBox) As String:`
  `will return the name of the selected slot.`

As we mentioned previously, all the forms will be loaded into the zone #1. The add edit form will let us add one or more slots, after adding slots we are ready now to insert the bombs, each slot can contain 1 to N bombs, each bomb object has his own specifications (Name, Interval).

- **6-1 Insert (Slots button):** after typing the slot's name, we are ready to insert the current slot to the database, **what happens when we click the 'Insert' button?**
  →The Slot is inserted in the database and created automatically, then a custom sub is called [`Private Sub refreshSlotList(ByVal avSlots As Windows.Forms.ListBox, ByVal txtEdit As Windows.Forms.TextBox, ByVal cmb As Windows.Forms.ComboBox)`],this sub will update the contents of the available slots listbox.
- **6-2 Save (Slots button):** This button is clicked to save the modification of a given slot, **how we modify a slot in this app?**
  →in this application the only modification that can be done is changing

the slot name, we first select **one slot** from the available slots list box, then we change it's name by typing a new value inside it, finally we click on save.

6-3 Delete (Slots button): This button is used to delete slot(s).
This application offers a great feature; we can select more than one slot at a time to delete it.

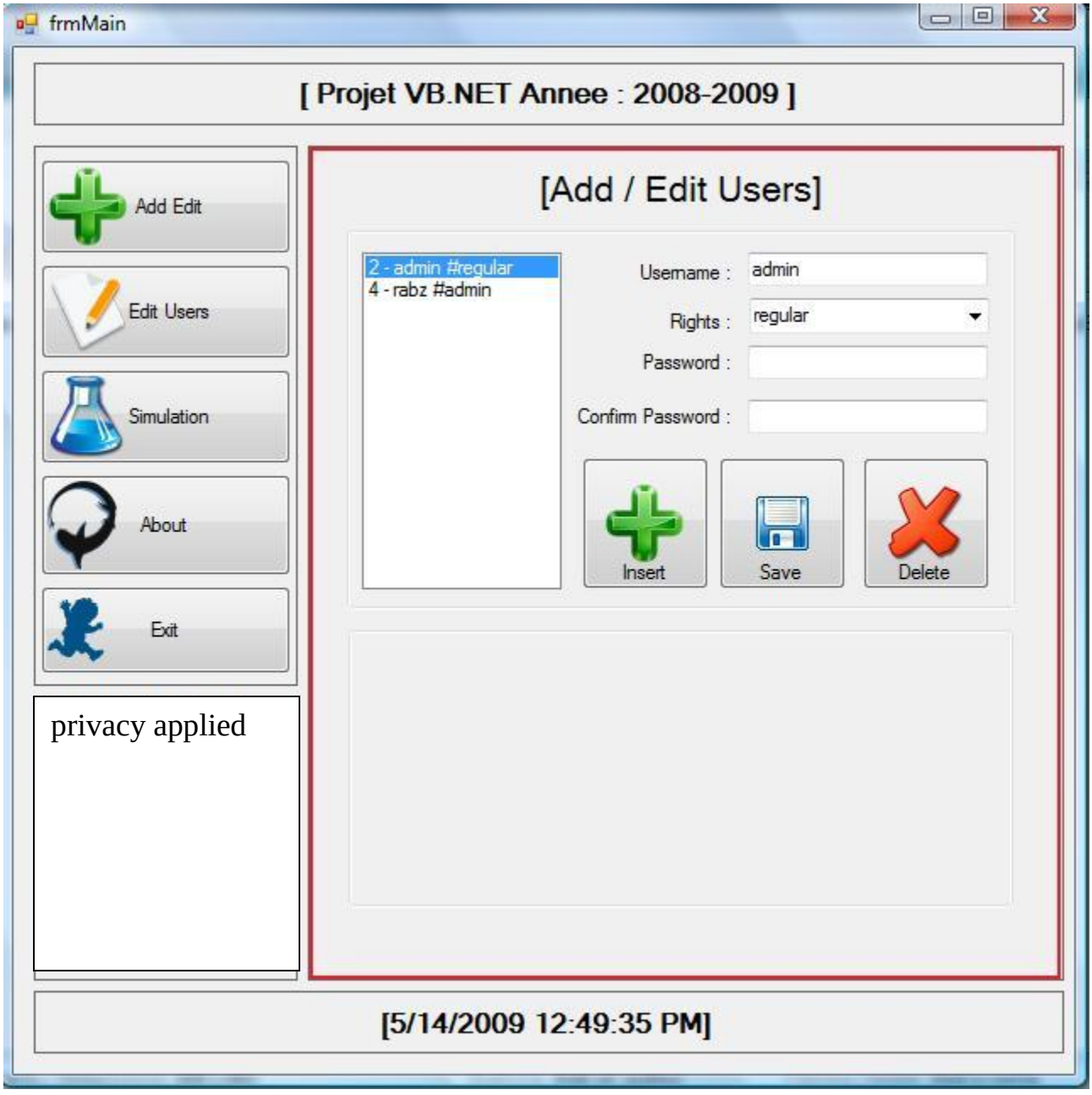
6-4 Insert (Bomb button): after adding the slots, we are ready to insert the bombs inside a specific slot, before clicking the **Insert button**; we have to define the **bomb characteristics (Name, Interval, Slot).**

6-5 Save (Bomb button): In case we had to change a detail concerning the bombs, we select a bomb change its values then we click on save.

6-6 Delete (Bomb button): similar to the previous delete button of the slots section, it is used to get rid off the unwanted bombs.

In these three buttons (6-4→6-6) every time a value is changed the listbox is update using a custom sub [`refreshBombList(ByVal avBomb As Windows.Forms.ListBox, ByVal cmbSlots As Windows.Forms.ComboBox)]`

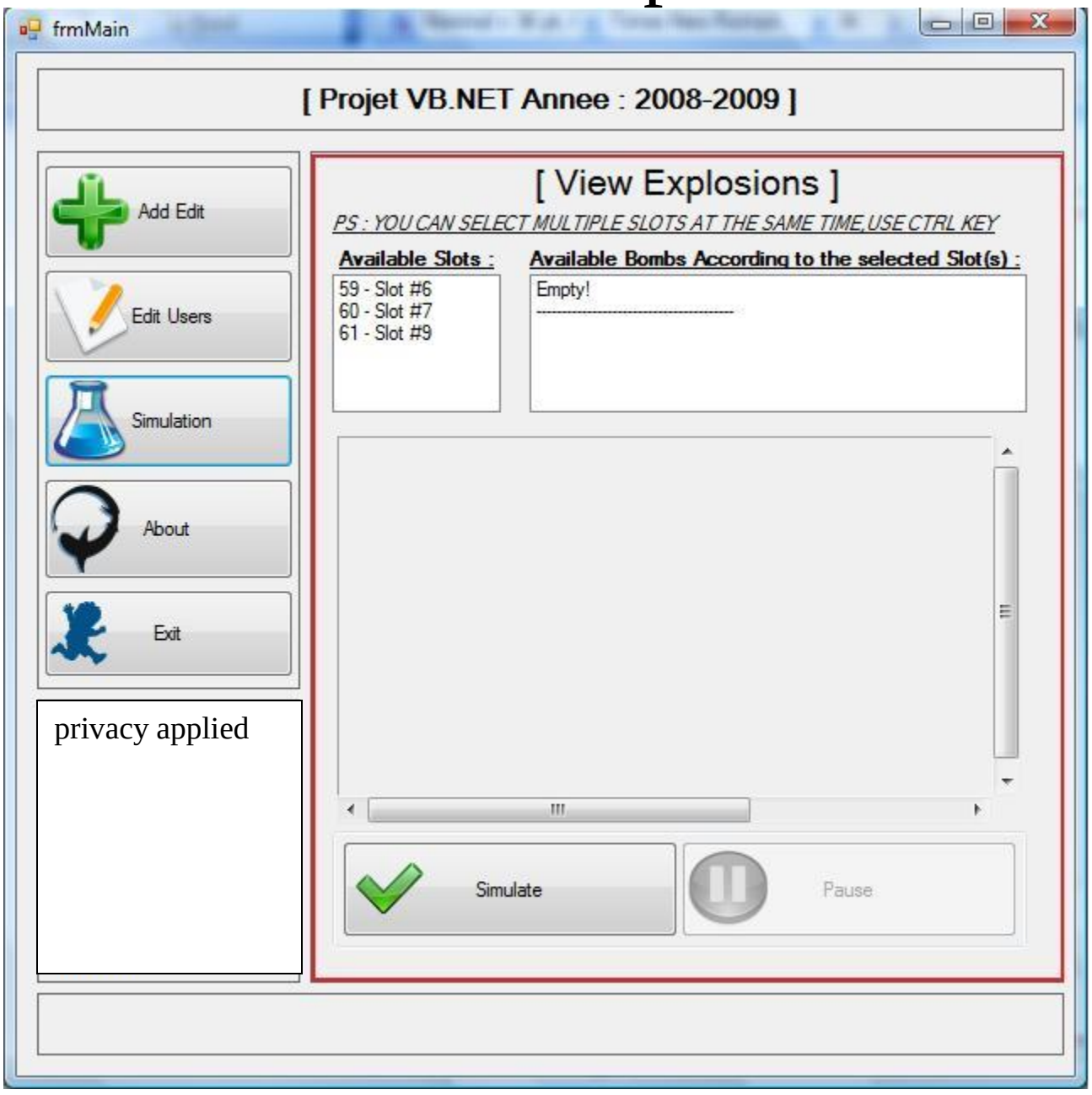
7 –FrmUserEdit (Form):



This form is the one who add/edit Users who access this application, as we mentioned before this form is inaccessible by users who don't own "**admin**" type account.

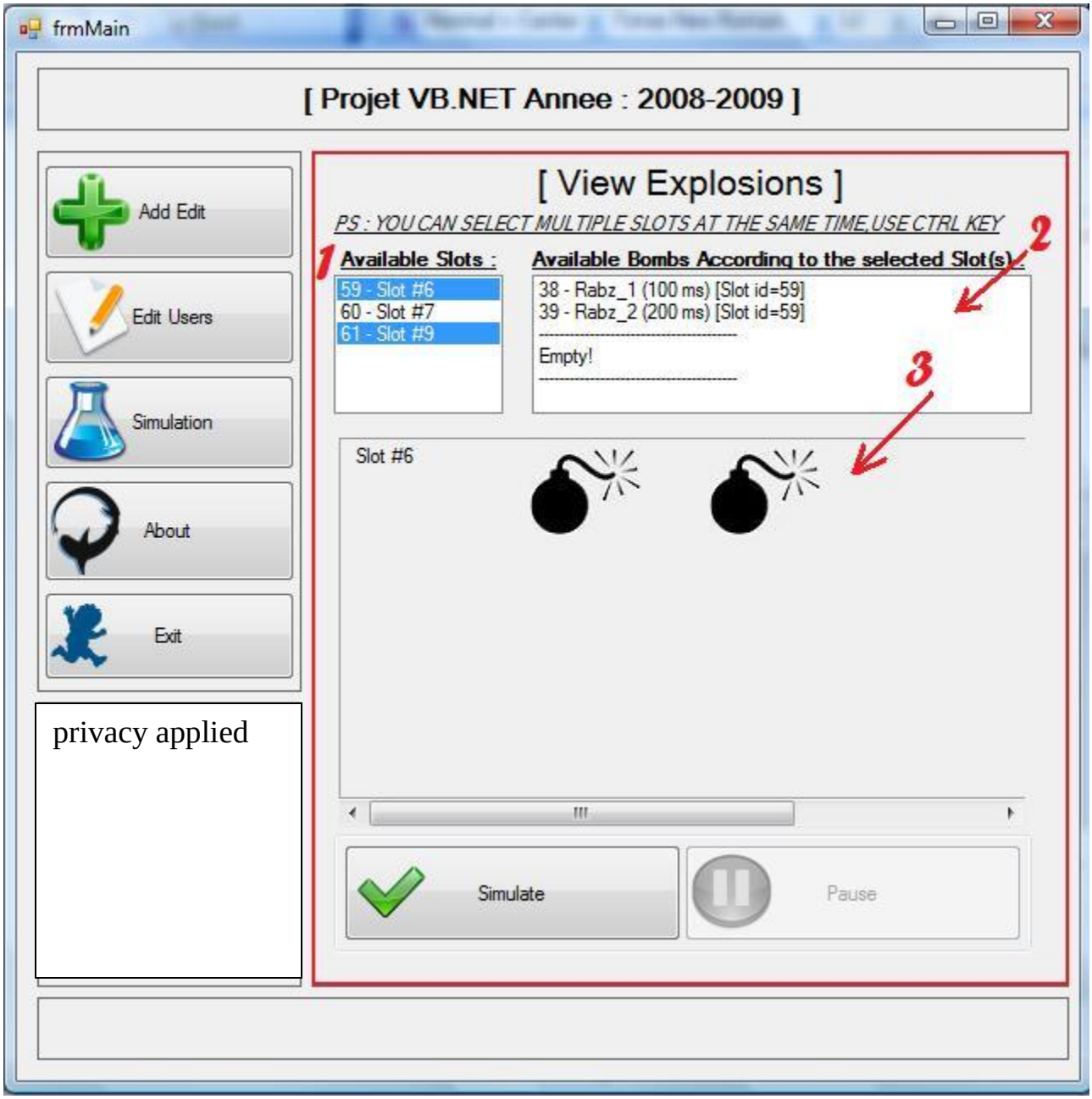
We won't talk in detail about this form because it's a simple login form.
It contains fields of the required information and three buttons (insert, add, delete) to manipulate the user accounts data.
Of course, like in all classes in this project all exceptions are treated (such as duplicated usernames, empty fields, etc . .

8–FrmViewExp (Form):



This form will **visualize** the available bombs and slots, and also will do a simulation of the explosions.
Once one or more slot(s) is selected from the left **slots listbox**, the **bombs listbox**

will automatically display inside it the bombs associated with the current selected slots and its information.



- **Zone #1** (in red) shows us that we can select multiple slots and the selected slots are not necessarily consecutive.

- Once we select the slots (see previous step), the information of the bombs contained by the slots are automatically listed in the **bombs listbox (Zone #2).**
- We can notice the **slot #6** contains **two bombs** and the **slot #9** is **empty**
- in the "simulator" section**(Zone #3)**, we can only see the bombs of the **slot #6**
- once we click simulate the timers countdown will start, and once the timers time out it will call the **"doAction"** function (mentioned before) and in our case the picture of the bomb will change into : 

Before "diving" in the form's functions and subs we have to explain the **"func__bomb" module.**

9–Func__bomb (Module):

This module contains Functions and Subs to manipulate the bomb objects; we added these functions and subs inside a module because these methods are object-independent.

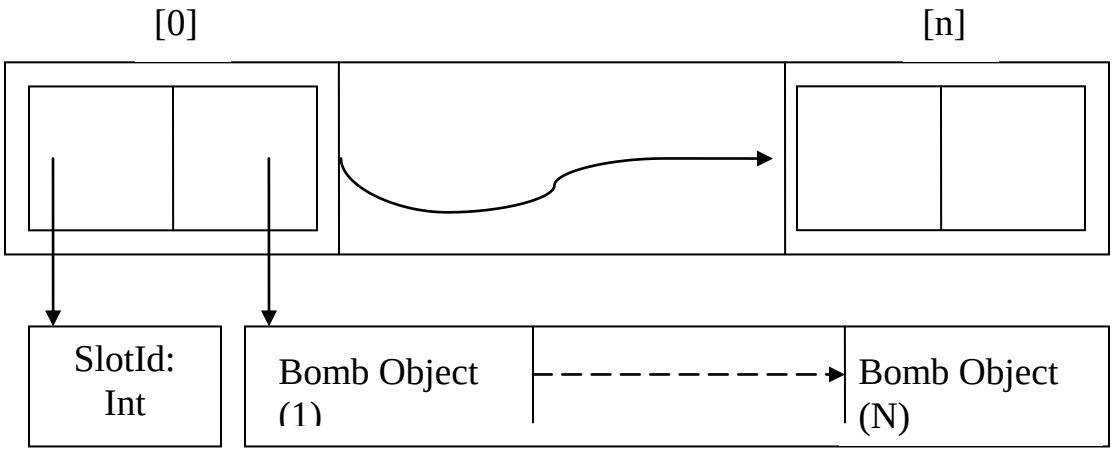
- `Public Function getBombs(ByVal slotId As Integer) As ArrayList:`
  This function will return an arrayList containing Bomb objects of the **same slotId**.
- `Function getBombById(ByVal id As Integer) As bomb :`
  This function will extract the bomb information from the database using the id provided by the argument, then these information will be used to create the bomb object and added back into the array.
- `Public Function getSlotNameById(ByVal id As Integer) As String:`
  This function will take the id of a given slot in its argument and return it's name.

After we have explained the func__bomb module now we can go back to the frmViewExp form and "dive" in the form's functions and subs…

8–FrmViewExp (Form): [Suite]

In the form declaration we can notice these two following attributes :

- `Private arr_selectedSlots As New ArrayList :`
  this array will hold the selected slotIds
- `Private arrayslotbombs As New ArrayList :`
  Every index of this arrayList contains another arrayList which contains:
  The first index [0] contains the slotId
  The Second index [1] contains an arrayList of bombs relative to the slotId:

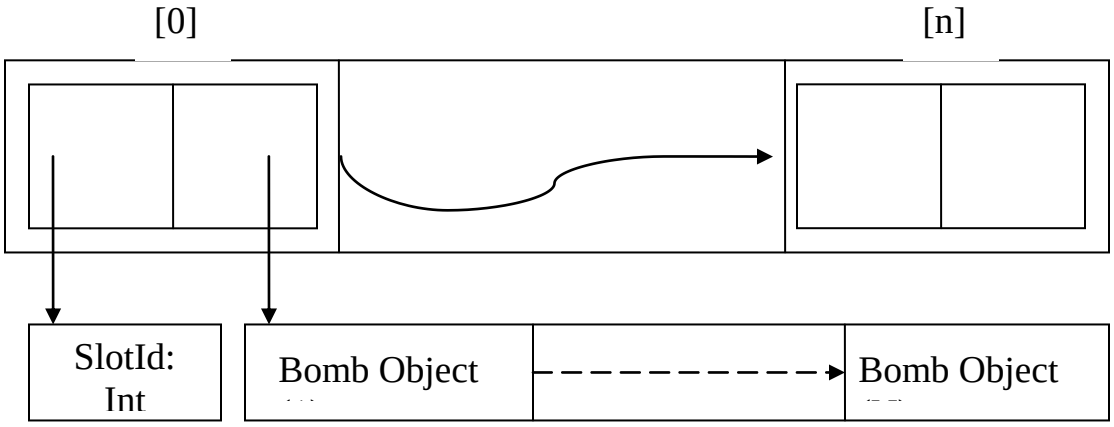


Once the form is loaded,the Slots and Bombs will be filled automatically using this sub :

- `Public Sub refreshSlots(ByVal avSlots As Windows.Forms.ListBox)`
  `'this sub is exceptionally public because we have to access it from the outside :`
  This sub will send an SQL query to the database and extract all the slots information then insert them into the corresponding listbox.

Once the Slots list box is filled with the available slots, we can select one or more slots. **How this application manage to select many slots?**
→we will reply on that question by going inside the code…

- Once we select one or more slot, the listbox ***SelectedIndexChanged*** event will call this function :
  `Private Function getGrpSelectedIds(ByVal lstbox As Windows.Forms.ListBox) As ArrayList`
  this function will get all the slot ids and place them inside an arrayList,and this arrayList is saved inside the variable "arr_selectedSlots" declared in the *global scope.*
- After we got all the selected **slot ids**, the same event will call another function :
  `Private Sub refreshBombList(ByVal avBomb As Windows.Forms.ListBox, ByVal ids As ArrayList)`
  This function will update the Bombs listbox in other words it will list the bombs using the ids obtained in the first step (arr_selectedSlots)
- `Private Function getArraySlotBombs(ByVal ids As ArrayList) As ArrayList:`
  This function will create our **"encapsulated model arrayList"**,this array list will hold the slotId and an array of bombs in each cell or index as the following image will show:



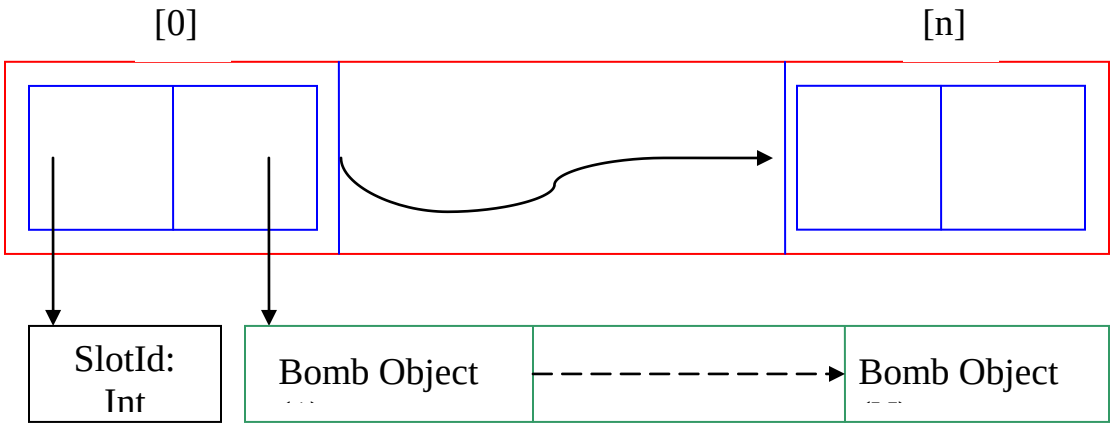
This function will be used to fill the array declared in the global scope

(arrayslotbombs),this array will be mainly used later to preform the simulation.

```
Private Sub DrawThem(ByVal tlp As Windows.Forms.TableLayoutPanel,
ByVal SlotAndBombs As ArrayList):
```

This function is the "Visualizer" function,it will draw the slots and the bombs inside a table layout panel, how this function works in a brief..?

```
This function uses in arguments a TableLayoutPanel, and an
arrayList (formated as shown in the previous step).
in this step the array will be "decapsulated".
More than one loop has been used to extract the data;
```



before explaining it, let's observe the picture above.
We notice that we have 3 objects in 3 different colors: red, blue, and green.
Let's call the red object: ***main_array***, blue object: ***second_array***, green object: ***bombs_array***.
First of all a **for loop** will be initiated to traverse the main_array (red):
for I as integer = 0 to length__of (main_array) -1
each **main_array (i)** contains a **second_array (blue)**
so another **for loop** is initiated to traverse the **second_array:**
for j as integer = 0 to length__of (second_array) -1
Now this is for the de-capsulation process. To visualize the bombs I will write the code in "pseudo-code" style :


- ```
  Private Sub DrawThem(ByVal tlp As Windows.Forms.TableLayoutPanel,
  ByVal SlotAndBombs As ArrayList):
  for I going from 0 to length_of(main_array)-1
  variable array_slotids_and_bombs = main_array(i)
  ```
  for j going from 0 to length_of(second_array)-1
  /*comment:
  j is going to start from 0 to 1 because it only contains to indexes
  */
  if(j=1) then /*comment: if the pointer is on the bombs array location*/
  if(second_array(i)→is_not_empty) then
  create bombs_array_list = second_array(1)
  create new_flowLayout_panel
  for k going from 0 to length_of(bombs_array)-1
  create image_bomb_before_explosion
  create new_bomb_object
  new_bomb_object→current_image = image_bomb_before_explosion
  new_flowLayout_panel→insert(new_bomb_object→current_image)
  next k
  end if
  end if
  next j
  next I
  end sub
  Basically it's a approach to the real code, I hope that this approach is enough to understand the real code.
  Okay,so now we have the slots and the bomb visualized inside a panel,fianally we have to start the detonations,to start them we select the slots then we click on simulate:
  ```
  Private Sub Simulate(ByRef arr As ArrayList, ByVal action As
  String)
  ```
  This Sub will decapsulate the array included in the argument(de-capsulation is explained in the previous paragraph),then the sub see the action if it is "start or stop".
  if it is a start it will walk through the array and start the timers of the bomb objects,
  otherwise it will stop them.


  ```

# 10 –Final Word (The Developers):

We as the developers of this project would like to say that it was an interesting project and we had fun while working on it, we are a little bit disappointed because we didn't make the com port circuit. In fact we faced some dead ends concerning this issue and we needed much time, a time cannot be found easily because we have many other projects.
We hope that this project is a **close approach to your vision.**
**Best Regards,**

***MATTA Elie, HALLAGE Rabih et al.***