Windows Forms

1 Introduction

1.1 Objectifs

During this TP, you will discover a new programming language, `C#`. You will be faced with a new coding style, a new programming language, in a new Workspace. The objective is not to kill you, for the moment. We will integrate some features related to the `C#` language and to *Visual Studio* to show you the right way.

This week, we will use *Winforms*, which allows the user to perform GUIs, and so simply and effectively. To set a specific goal you must perform a hanged game with image display, counters and text retrieval, etc.

1.2 The `C#` Language

`C#` (pronounced "see sharp") is an object-oriented programming language, created by Microsoft, by the development team of Anders HEJLSBERG, the `Delphi` creator.

The language is intended for use in developing software components suitable for deployment in distributed environments. The language is close to `Java` language, in that they have the same syntax.

Unlike `Caml`, which is a strongly typed functional programming language, `C#` is intended to be an object-oriented programming language. Basic rules will be specified later.

1.3 Visual Studio

Microsoft Visual Studio is an integrated development environment (IDE) from Microsoft. It is used to develop console and graphical user interface applications. This IDE supports different programming languages by means of language services, built-in languages include `C#` , `F#`, `Visual C++` and `Visual Basic`. With this IDE, you can develop some types of applications: graphical user interface applications, console applications, class libraries, Windows services and websites.

Epita students are allowed to download for free *Visual Studio* on the web site *Microsoft DreamSpark for Academic Institutions*[1] or to download for free the *express* version on the Microsoft web site.

[1] `https://www.dreamspark.com/Institution/Access.aspx` : Search EPITA.

2 Submission

Your submission must be a `zip` file.

2.1 `AUTHORS` file

This file contains your *login* by the following rules: an asterisk `*`, a space, your *login* (`login_x`) and a newline – which is represented by the `$` character in the following example:

```
* login_x$
```

2.2 Directory tree

```
rendu-tp0-login_x.zip
    | login_x/
        | AUTHORS
        | HelloWorld/
            | HelloWorld/
            | HelloWorld.sln
        | ImageViewer/
            | ImageViewer/
            | ImageViewer.sln
        | HangedGame
            | HangedGame/
            | HangedGame.sln
```

3 The basics of `C#`

`C#` has different programming mechanisms than `Caml`. These mechanisms will be explained in the next TP sessions. This first TP doesn't require a lot of programming. Its purpose is to let you discover *Visual Studio* and windows creations. `C#` documentation is available on *MSDN*:

http://msdn.microsoft.com/en-us/library/67ef8sbd.aspx

3.1 Basic knowledge

- **Comments**: To comment a single line use double slash `//`. To do it on multiple lines, use `/* My comment.*/`.
- **Block**: A statement block is enclosed in `{ }` brackets and can contain nested blocks.
- **Statement**: The actions that a program takes are expressed in `statements`. A statement can be a single line of code that end with a semi-colon '`;`', or multiple statements in another block. Statement examples:
  - variable declaration;
  - assignment of value;
  - method call;
  - iteration statements – loops;
  - branching to one or another block of code, depending on a given condition.
- **Expression**: An expression is a sequence of one or more operands that can be evaluated to a single value.

- **Operators**: Arithmetic operators are not *typed* as in `Caml` : `+ - * /`, *i-e* it is possible to compute by mixing integer and float numbers. There are lots of operators in `C#`, another one which will be useful for this TP is the dot '`.`' to access a member (when using objects for example).
- **Types**: `C#` define the following types : `int`, `float`, `double`, `char`, `string`, `objects`, etc.
- **Method**: A method *prototype* is composed of return type, method name, and some optional parameters with their type between parenthesis.

Example : factorial function

The recursive factorial function `C#` can be written as follows:

Source code
```
int fact(int i)
{
    if (i <= 1)
        return 1;
    else
        return i * fact(i - 1);
}
```

Notice that *block* brackets concerning `if` and `else` statements are optional if there is only one statement.

3.2 New project Windows Forms

- Launch Visual Studio – if any ask at start choose `C#` at launch.
- Click on **File**, **New** and **Project...**
- Choose **Windows Forms Application**

The Visual Studio IDE[2] is now your development tool. Do not hesitate to customize it to make it comfortable. To execute a project, push the keyboard key **F5**. Don't forget to indent your code, for which the shortcut is : `Ctrl + K + Ctrl + D`.

The solution explorer, which you can find on the right border of the IDE, shows you the project overview. It contains:

- `WindowsFormsApplication` (1 project)
  - WindowsFormsApplication: The application.
    - *Properties*: Project resources.
    - *References*: Loaded libraries in your application.
    - `Form1.cs`: Embed your code and draw the graphical editor.
      - `Form.Designer.cs`: Generated code by the graphical editor.
      - `Form1.resx`: **Your code**.
      - `Program.cs`: Entry point of the application.

[2]Integrated Development Environment

3.3 Windows Forms

Windows Forms permits the user to design graphical interface easily on Windows. Visual Studio has a *toolbox* where you find a list of available *WinForms*. To use it, just drag & drop.

When a *Form* is selected, we can customize it by modifying fields (size, color, position, content, etc.) in *properties* tab of VS.

A *lightning* tab is also available next to the *properties* tab. We can set multiple actions of the user, such as mouse click, key pressed on keyboard, etc. By doing a double-click on the field `Click` : an editor window appears. A *method* has been automatically created by VS. This *method* enables the creation of a *callback* to define the application behavior. The source code of the editor window contains different methods of the window `Form`.

4 Exercise 1 : `HelloWorld`

In the following exercises, you will enjoy a good deal of creative freedom. Experiment!

4.1 Handling

You must start by creating a new Windows Forms project, whose name is the one of the exercise : `HelloWorld`. Design a graphical user interface with the following elements:

- A *button* `button_say` whose text is `Say`.
  - Change button's name `(Name)` in *properties*.
  - Change the content of the *field* `Text` in *properties*.
- A *label* `label_say` with no text.
  - Change label's name `(Name)` in *properties*.
  - Delete the content of the *field* `Text` in *properties*.

The graphical user interface is now designed and configured. To make it work, you must connect *WinForms*. Double-click on button `button_say` of the graphical interface *or* in *lightning* tab double-click on `Click`. The source code editor appears with the *callback*.

Assign the text of label `label_say` :

Source code
```
private void button_say_Click(object sender, EventArgs e)
{
    label_say.Text = "Hello World !";
}
```

Push **F5** to launch the application with *debugger*.

4.2 Improvements

Creative freedom in this section.

- Say *welcome* in multiple language by using a `ComboBox`.
- Get the user's name and then say to him *welcome* by using a `TextBox`.
- Say *welcome* with colours by adding a `Button` and the dialog box `ColorDialog`.

ACDC 2016
IT'S DANGEROUS TO CODE ALONE

5 Exercise 2 : `ImageViewer`

5.1 Handling

The goal of this exercise is to make an image viewer. The graphical interface must contain:

- A *Button* : `button_open` with the text `Open`.
- A *PictureBox* : `pictureBox_viewer`
- A dialog box *OpenFileDialog* : `dialog_open`

Assign the *callback* of `button_open` :

Source code
```
private void button_open_Click(object sender, EventArgs e)
{
    dialog_open.ShowDialog();
    Image img = Image.FromFile(dialog_open.FileName);
    pictureBox_viewer.Image = img;
}
```

5.2 Improvements

Creative freedom in this section.

- Allow the image viewer to draw high resolution images by making a thumbnail.
- Show the image's filename and characteristics – height, width, filetype by retrieving the extension, etc.
- Set the graphical interface with colours.

6 Exercise 3 : `HangedGame`

Well, you now have some knowledge in `C#`, so you will experiment with a little project. You have learned to use **buttons**, **textboxes**, **image displaying**, etc. You will use all of these objects to make this hanged game.

6.1 The user interface

Your interface must have at least:

- 3 `button` s :
  - 'New game'
    - 'End of game'
    - 'Test'
- One `TextBox` to retrieve the letter given by the user.
- One `RichTextBox` to display the word with found letters.
- One `ProgressBar` to display the level of success.
- One `PictureBox` to see the hanged man.

- One Label to display the number of trials left.

With all these elements you can create your own hanged game. Download the resources to make it at :http://perso.epita.fr/~acdc/.

6.2 Goal

Everybody knows the hanged game, but we will give some directions to make it:

- 'New game' :
  - ProgressBar set to 0.
  - Number of trials set to 7. - *for example* -
  - First Image.
  - RichTextBox cleaned.
  - Button 'New Game' disabled.
  - Buttons 'End of game' and 'Test' activated.
- Button 'TEST' clicked :
  - If the letter is on the hidden word, increment the ProgressBar and display the letter on the RichTextBox
  - Else, the number of trials is decremented and the next image is displayed.
- You lose the game if the number of trials equals 0.
- You have to click on the button End of game to quit the game.

6.3 Bonuses

Some examples:

- Add a ListBox to display the history of tests.

Source code
```
// Add str in the string.
MyList.Items.Add(str);
// Clear the list.
MyList.Items.Clear();
```

- Add a dictionary with some words inside and take it randomly. *You have to search about the use of an array – add, get, etc.*

IT'S DANGEROUS TO CODE ALONE