Files & Bits

1 Submission

Your submission **must** be a `zip` file.

1.1 `AUTHORS` file

This file contains your *login* with 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$
```

1.2 Directory tree

The name of the `zip` file is `rendu-tpcs6-login_x.zip` and follows this directory tree:

```
rendu-tp6-login_x.zip
    | login_x/
        | AUTHORS
        | BMPReader/
            | BMPReader/*
            | BMPReader.sln
        | DataHiding
            | DataHiding/*
            | DataHiding.sln
        | GameMap/
            | GameMap/*
            | GameMap.sln
```

- The file `README` is advised in case of bonuses.
- No *dump* file is allowed. No `bin/` or `obj/` in the archive.
- Do not finish this pratical at the last minute.
- **The code has to compile.** Do not forget to check your archive.

2 Introduction

This TP will cover the basics about file manipulation (`BMP` images), and bits to do steganography.

3 `static` keyword

`Static` modifier is used to declare a static member, *i.e.* refer to the *type* rather than a specific object. The keyword `static` is used with classes, fields, methods, properties, operators, events and constructors.

- Designating a *static class* means that all members and class methods are static, for example the `Convert` class does not need to be used with an instance.
- Another example like `Console.WriteLine` means a *class method* (or static method) that can be used without having to instantiate an object such as `Console`.

- We can also refer to a method in a *static variable*. This keeps the result of this variable whenever calls are made to this method.

Source code
```
/* This is useless. */
static void UpdateCount()
{
    static int count = 0;
    count++;
}
```

In the example, the `UpdateCount` method contains a variable of type `int` named `count`. This variable is initialized to zero at the first call to the `UpdateCount` method. On subsequent calls to the `UpdateCount`, `count` variable is not worth zero, but rather the value resulting from the preceding call.

4 Program arguments

The `Main` method is the entry point of all programs.It is here that the program begins and ends. This method has the particularity of handling arguments of the program.

```
program.exe arg0 arg1 arg2
```

In the above example, arguments are placed in the *string* array that is one argument of the `Main` function. You can access to `"arg0"` by doing `args[0]`, etc ...

Source code
```
class TestClass
{
    static private void Main(string[] args)
    {
        /* Display the number of command line arguments. */
        System.Console.WriteLine(args.Length);
    }
}
```

- The `Main` method can return an integer (which is often use as error code). You just have to replace `void` with `int`.
- If the program never takes arguments, `string[] args` is optional. Notice that the arguments array can take names other than *args*.
- The `Main` method must always be `private` and `static`.

5 `struct` keyword – Structures

Structures are used to encapsulate a small group of variables, for example, characteristics of an object inventory.

Source code
```
public struct Book
{
    public decimal price;
    public string title;
    public string author;
}
```

A structure can contain constructors, constants, fields, properties, events, etc. But in these cases it is better to make a class.

6 Binary files

The files do not contain any text. A binary file (for example `BMP`, `EXE`, `MP3`, etc.) will be unreadable in a text editor such as `NotePad`. These files have a structure (they are said to be *formatted*) particular depending on the type. Most binary files begin with a *field* called *magic number*. This field is a signature of a few bytes to identify the file type.

For example, a `PDF` file starts with 4 bytes in `ASCII` format: `%PDF`. In the case of a `BMP` file, the *magic number* is `BM`.

6.1 `BMP` file format

The *bitmap* format is uncompressed image file. It is composed of 3 zones:

- the header;
- the color palette;
- data relating to the image.

Full details on the `BMP` format: http://en.wikipedia.org/wiki/BMP_file_format

Field name	Size in byte
Magic Number	2
File Size	4
Reserved1	2
Reserved2	2
File Offset to PixelArray	4
DIB Header Size	4
Image Width	4
Image Height	4
Planes	2
Bits per Pixel	2
...	...

Table 1: Structure of the BMP file header.

Certain information contained in the header are not useful for this practical.

To read the information contained in this header, we use the `BinaryReader` class that can read data from a stream and store it in a type.

Source code

```
/* Open the file. */
FileStream fs = new FileStream("filename.ext", FileMode.Open, FileAccess.Read);
BinaryReader br = new BinaryReader(fs);
/* Reads an 32 bits (4 bytes) signed integer. */
int i = br.ReadInt32();
/* Reads a character (1 byte). */
char c = br.ReadChar();
fs.Close();
```

6.2 Read BMP file with .NET

.NET framework provided a class for editing BMP files. This class is contained in the component System.Drawing.

Source code

```
using System.Drawing;
...
/* Loads a bitmap file. */
Bitmap img = new Bitmap("filename.bmp");
```

7 XNA Game Studio

Extract from MSDN[1]:

> XNA Game Studio is an integrated development environment designed to make it easier to develop games for Microsoft Windows, Xbox 360, and Windows Phone. XNA Game Studio extends Microsoft Visual Studio with support for the XNA Framework and tools. The XNA Framework is a managed-code class library that contains features targeted specifically at game development. In addition, XNA Game Studio includes tools for adding graphic and audio content to your game.

Step by step documentation: http://msdn.microsoft.com/en-us/library/bb203893.aspx

7.1 Construction of a game

A video game begins with an initialization phase in which images are loaded into memory. Then the game takes place in a cycle (an infinite loop) which successively produce the updated game elements and the display of images.

XNA sets up place the basic mechanisms of the game. In this pratical it is important to remember:

- Graphic resources are loaded in the `LoadContent` method.
- Set configuration is updated in the `Update` method.
- Rendering the image in the `Draw` method.

7.2 Add a *sprite*

Perform a simple drag and drop of the image in the Solution Explorer in Content Manager. Configure (*Properties*) the asset name by putting a name explaining the content of the image (if needed). Note also that you can make settings for the transparency (designate a color filter).

7.3 Load a *sprite*

Add a `Texture2D` attribute and `Vector2` or `Rectangle` in the `Game1` class and edit the `LoadContent` method:

[1] http://msdn.microsoft.com/en-us/library/bb203894.aspx

Source code

```
// This is a texture we can render.
Texture2D myTexture;

// Set the coordinates to draw the sprite at.
Vector2 spritePosition = Vector2.Zero;

protected override void LoadContent()
{
    // Create a new SpriteBatch, which can be used to draw textures.
    spriteBatch = new SpriteBatch(GraphicsDevice);
    myTexture = Content.Load<Texture2D>("mytexture");
}
```

7.4 Draw a *sprite*

Edit the `Draw` method. Always specify when you start drawing and when you have finished. The `Draw` method is called for each *frame* generation and we must *clean up* the drawing surface.

Source code

```
protected override void Draw(GameTime gameTime)
{
    graphics.GraphicsDevice.Clear(Color.CornflowerBlue);

    // Draw the sprite.
    spriteBatch.Begin();
    spriteBatch.Draw(myTexture, spritePosition, Color.White);
    spriteBatch.End();

    base.Draw(gameTime);
}
```

8 Exercise 1 : `BMPReader`

You have to write a program that uses, as a parameter a filename, and prints information. **You must not use the `.NET` API in this exercise** and you must read the file using `BinaryReader`.

8.1 Arguments handler

You have to handle the case in which the user is calling your program with wrong arguments. The program needs only one argument.

```
bmpreader.exe image.bmp
```

- If no arguments or more than two arguments are used, display a help message:

```
Error: This program needs only one argument.
Usage: bmpreader.exe <file.bmp>
```

- If after checking, the file path is incorrect:

```
Error: The file path is invalid.
Usage: bmpreader.exe <file.bmp>
```

8.2 *Magic number* checker

Before going any further, we check the *magic number*. If it does not match to a *bitmap* image file, the program exits and prints an error message:

- Full file path.
- Tell the user that the file is not a `BMP` file.

```
C:\path\file.txt : This file is not an bitmap (BMP) file.
```

8.3 Information printer

If no error appears, you must display the following information:

- Picture dimensions
- Number of bits per pixel

Display information by following the example:

```
C:\path\file.bmp file informations:
    - Magic number is valid.
    - Dimensions: 420x314
    - Number of bits per pixel: 24
```

Do not forget to put a newline character even if it is the last line.

9 Exercise 2 : `DataHiding`

The purpose of this exercise is to show you another way to transfer a message, like the way of our good old friend 007-through steganography.

9.1 Short History

A few weeks ago, you discovered a way to send a message in encrypted form, thanks to cryptography. This method consisted in changing the original message to hide it from people that don't know the private key. This method is very useful, but not really discreet.

The idea of the steganography system is to hide a message in another message. Steganography can be performed in the real world using ink made from lemon juice, visible only in a hot environment. Today we will learn how to hide a message in an image.

9.2 Stegano in a 24-bit bitmap

In the 24-bit bitmap that we will use, every pixel is represented by 24 bits. In these 24 bits, each component (red, green and blue) is coded on 8 bits.

To hide our information, we will exploit the fact that two colours that are too close are indistinguishable for humans. We will use the last bit of each byte to hide our data (because it is the least significant bit; the change is to alter the byte less than 0.4%, whereas changing the first bit, the MSB or most significant bit, would have made for example changed 50 to 178, which is a big difference).

Thus, if we want to hide "5" (101 in binary) in a pixel (240, 65, 169), in binary (11110000, 00100001, 10101011) is going to take each bit of "5" and put it on less significant bit components

of these pixels. The first is a 1, so the first component takes 1 on the less significant bit, thus becoming 241 (11110001 in binary). The second is a 0, so the second component takes a 0 on the less significant bit, thus becoming 64 (00100000 in binary). The last is a 1, so the third component takes 1 on the less significant bit, which changes nothing.

We will hide text. The text is nothing more than a series of ASCII characters, which as we know, are bytes. So it takes 8 components, meaning almost three pixels, to hide a single character.

9.3 Let's go !

For this exercise, you are truly free to choose your method, as you'll soon have to stand on your own two feet !

So you have to write a program, that will allow the user to choose an image, then load and retrieve a hidden message inside. This program will of course have a second function, that of hiding a message in an image. Of course, you will still need some guidance this year, we know very well that you can't fly yet!

9.3.1 Getting Started

This part will show you how to make this program.

First, we recommend that you to create a Windows Form project, wich will give you a fast interface and many possibilities for your program.

In our example, we will say that your interface has three button and a textBox. A button to load the image, a button (named `hideButton`) to hide the text in the textBox in a copy of the picture called `result.bmp`, and finally a button (named `revealButton`) to read the message hidden in the loaded image.

9.3.2 Image , Image ... show me your message!

Now that the interface is raised, you will code the following function created by double clicking on the button `revealButton` :

Source code

```
private void revealButton_Click (object sender, EventArgs e);
```

This function is called when the button is clicked, so check at the beginning of the function that the image is correctly loaded. Then you have to browse the image line by line to get the whole message. We will define that a message will always end with a '\0' or null character, corresponding to the character in position 0 in the `ASCII` table.

Do not forget the loops TP, especially dual loop `for`, they will be useful. Restore the message on the textBox.

`Tip:` the least significant bit is the parity of a number.

9.3.3 Your mission, should you decide to accept it

We know now how to recover a hidden message. Let us see how to hide ours.

You will code the following function created by a double click on the button `hideButton` :

Source code

```
private void hideButton_Click (object sender, EventArgs e);
```


Again, check that an image is loaded. Then we need to get back the message written TextBox and for each character, recover corresponding `ASCII` code and convert it to binary. Once binarized, the character can be inserted in the image, byte by byte. Do not forget the '\0' at the end of the message.

10 Exercise 3 : `GameMap`

In this exercise you will model a game map using the `BMP` format. This will allow you to easily create maps without using any map editing tool. You can go further by creating for example a map with multiple layers:

- a background layer
- an event layer to make actions depending on the color (fight, dialogue, etc.)
- an object layer to draw special objects on the map.

In this exercise you must draw, the image using XNA.

Figure 1: `BMP` picture of size `10x10` representing the map.

10.1 Preparation

Create a new XNA project named `GameMap` in Visual Studio 2010. Then add the following component `System.Drawing`[2]. In *Solution explorer*, right-click on *References*, *Add Reference...* and search in `.NET`. To finish, add at the beginning of the `Game1.cs` file:

Source code

```
using System.Drawing;
```

We will draw rectangles with `XNA`. For each drawing instruction, `XNA` needs a `Texture2D`. Create a `BMP` picture:

- File name : `blank.bmp`
- Dimensions : `1x1`
- Pixel color : white

Add the picture in `GameMapContent` section.

Create another `BMP` picture to design the map. Add it in the `GameMapContent` and setup VS to copy the image in the output directory (where there is the executable) : `Properties` and `Copy always` set to `Copy to Output Directory`.

[2]Beware! This component adds another refererence to the class `Color`. You will have to specify the namespace, for example: `System.Drawing.Color`.

10.2 Implementation

Add the following fields in `Game1` class:

Source code
```
/* BMP file that contains our map. */
Bitmap map;
/* Blank texture to draw Rectangle. */
Texture2D blank;
```

The `BMP` picture representing the map is located in the `Content` directory whose path is the property `RootDirectory`:

```
this.Content.RootDirectory + "\\map.bmp"
```

Load the `BMP` picture and the `Texture2D` (*cf. course notes for this TP*) in this method:

Source code
```
protected override void LoadContent();
```

To finish, implement the following method in `Game1` class. This method reads the `BMP` file and draw rectangles (representing pixels).

Source code
```
/* Draw the map on the screen. */
private void DrawMap();
```

- Use properties `Width` and `Height` of `Bitmap` class.
- Create instances of class `Microsoft.Xna.Framework.Color` by using pixel value of `System.Drawing.Color` use properties `R`, `G` and `B`.
- Get dimensions of `XNA`'s window:

Source code
```
int screen_height = this.GraphicsDevice.Viewport.Height;
int screen_width = this.GraphicsDevice.Viewport.Width;
```

Now, add three lines in this method to draw the map:

Source code
```
protected override void Draw(GameTime gameTime);
```

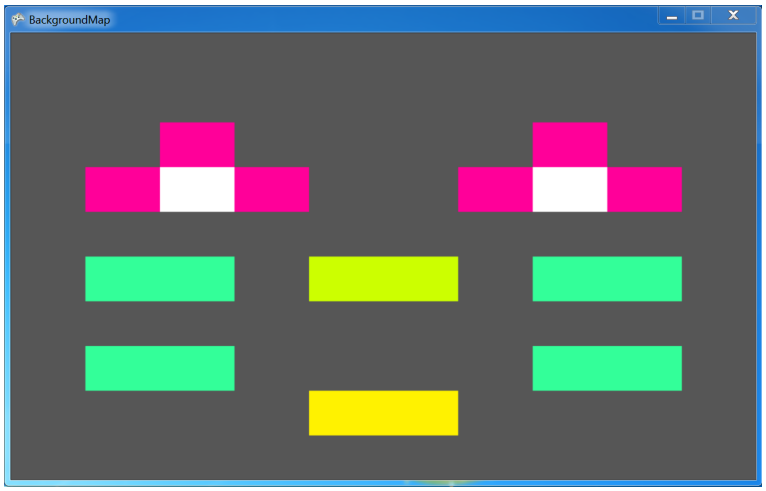

Figure 2: Map drawing in XNA.