Fichiers & Bits

1 Rendu

Le TP devra être rendu en réalisant une archive `zip`. Tout autre format de fichier sera systématiquement refusé.

1.1 Fichier `AUTHORS`

Ce fichier contient votre *login*, sous la forme suivante : une étoile `*`, un espace, votre *login* (`login_x`) et **un** retour à la ligne – représenté par le caractère `$` dans l'exemple ci-dessous.

```
* login_x$
```

1.2 Arborescence de l'archive `zip`

L'archive `zip` pour ce tp se nommera `rendu-tpcs6-login_x.zip` et disposera de l'arborescence suivante :

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

- Un fichier `README` peut s'avérer nécessaire dans le cas où vous réalisez des bonii.
- L'archive doit être exempt de tout fichiers *intermédiaires*, pensez à nettoyer chaque projet. N'incluez pas les dossiers `bin/` et `obj/`.
- Ne terminez pas le TP à la dernière minute.
- **Le code rendu doit impérativement compiler.** Vérifiez votre archive avant de rendre.

2 Introduction

Ce TP abordera la manipulation des fichiers (les images `BMP`), ainsi que la manipulation des bits pour pouvoir faire de la stéganographie.

3 Mot-clé `static`

Le modificateur `static` permet de déclarer un membre statique, *i-e* faire référence au *type* plutôt qu'à un objet spécifique. Le mot-clé `static` s'utilise avec des classes, champs, méthodes, propriétés, opérateurs, évènements et les constructeurs.

— Désigner une *classe statique*, signifie que tout les membres et méthodes de la classe sont statiques, par example la classe `Convert` n'a pas besoin d'instance pour être utilisée.
— Un autre example comme `Console.WriteLine` désigne une *méthode de classe* (ou méthode statique) que l'on peut utiliser sans avoir à instancier un objet de type `Console`.
— On peut aussi dans une méthode désigner une *variable statique*. Cela permet de conserver le resultat de cette variable au fil des appels à cette méthode.

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

Dans l'exemple, la méthode `UpdateCount` contient une variable de type `int` nommée `count`. Cette variable est initialisée à zéro, lors du premier appel à la méthode `UpdateCount`. Lors des prochains appels à la méthode `UpdateCount`, la variable `count` vaudra pas zéro mais la valeur résultant de l'appel précédant.

4 Arguments d'un programme

La méthode `Main` est le point d'entrée de tout programme, c'est ici que le programme commence et termine. Cette méthode a la particularité de pouvoir prendre des arguments qui sont ceux utilisés lors de l'appel du programme.

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

Dans l'exemple ci-dessus, les arguments seront situés dans le tableau de *string* en paramètre de la fonction `Main`. Ainsi on accèdera à `"arg0"` avec `args[0]` et ainsi de suite.

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

— La méthode `Main` peut également retourner en entier (souvent utilisé comme code d'erreur). Il suffit pour cela de remplacer `void` par `int`.
— Si le programme ne prend jamais d'argument, il n'est pas obligatoire de mettre `string[] args` dans les paramètres. Sachez aussi que le tableau d'arguments peut s'appeler autrement que `args`.
— La méthode `Main` doit toujours être `private` et `static`.

5 Mot-clé `struct` – Les structures

Les structures sont utilisées pour encapsuler un petit groupe de variables, par exemple les caractéristiques d'un objet dans un inventaire.

Code source

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

Une structure peut contenir des constructeurs, des constantes, des champs, des propriétés, des évènements, etc ... Mais dans ces cas là, il est préférable de faire une classe.

6 Fichiers binaires

Les fichiers ne contiennent pas tous du texte. Un fichier binaire (par exemple `BMP`, `EXE`, `MP3`, etc ...) sera illisible dans un editeur de texte comme `NotePad`. Ces fichiers comportent une structure (on dit qu'ils sont *formaté*) particulière selon leur type. La plupart des fichiers binaires commencent par un *champ* appelé *magic number*. Ce champ est une signature de quelques octets permettant d'identifier le type du fichier.

Par exemple, un fichier `PDF` débute par 4 octets au format `ASCII` : `%PDF`. Dans le cas d'un fichier `BMP`, le *magic number* est `BM`.

6.1 Format de fichier `BMP`

Le *bitmap* est un format de fichier non compressé d'image. Il est composé de 3 zones :
- l'en-tête ;
- la palette de couleurs ;
- les données relatives à l'image.

Détails complets sur le format `BMP` : http://en.wikipedia.org/wiki/BMP_file_format

Nom du champ	Taille en octet
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 de l'en-tête d'un fichier BMP.

Certaines informations contenues dans l'en-tête ne sont pas utiles pour ce TP.

Pour lire les informations contenues dans cette en-tête, on utilise la classe `BinaryReader` qui permet de lire les données dans un flux et de les stocker dans un type.

Code source
```
/* 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 Lire les fichiers BMP avec .NET

Le framework .NET fourni une classe permettant d'éditer des fichiers BMP. Cette classe est contenue dans le composant System.Drawing.

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

7 XNA Game Studio

Extrait de la documentation MSDN [1] :

> XNA Game Studio est un environnement de développement intégré conçu pour faciliter le développement de jeux pour Microsoft Windows, Xbox 360 et Windows Phone. XNA Game Studio étend Microsoft Visual Studio par la prise en charge de XNA Game Studio Framework et d'outils. Le XNA Game Studio Framework est une bibliothèque de classes de code managé qui contient des fonctions spécifiquement conçues pour le développement de jeux. De plus, XNA Game Studio inclut des outils pour ajouter des contenus graphiques et audio à votre jeu.

Documentation pas à pas d'un projet XNA : http://msdn.microsoft.com/fr-fr/library/bb203893.aspx

7.1 Construction d'un jeu

Un jeu vidéo débute par une phase d'initialisation où les images sont chargées en mémoire. Ensuite le jeu se déroule suivant un cycle (une boucle infinie) où se produisent successivement la mise à jour des éléments du jeu et l'affichage des images.

XNA met déjà en place les mécanismes de base du jeu. Dans le cadre de ce TP il est important de retenir :

— Les ressources graphiques sont chargées dans la méthode LoadContent.
— La configuration du jeu est mise à jour dans la méthode Update.
— Rendu de l'image dans la méthode Draw.

1. http://msdn.microsoft.com/fr-fr/library/bb203894.aspx

7.2 Ajout d'un *sprite*

Effectuez un simple glissé-déposé de l'image dans l'explorateur de solution au niveau du gestionnaire de contenu. Configurez (*Properties*) l'*asset name* en mettant un nom explicitant le contenu de l'image (si besoin). Notez également que l'on peut effectuer les réglages de la transparence (désigner une couleur à filtrer).

7.3 Chargez un *sprite*

Ajoutez un attribut `Texture2D` et un `Vector2` ou `Rectangle` à la classe `Game1` et éditez la méthode `LoadContent` :

Code source
```
// 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 Affichage d'un *sprite*

Éditez la méthode `Draw`. Il faut toujours préciser à quel moment on commence à dessiner et quand on a terminé. La méthode `Draw` est appelée à chaque génération de *frame*, il faut avant de dessiner *nettoyer* la surface de dessin.

Code source
```
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 Exercice 1 : `BMPReader`

Le but de cet exercice est de vous familariser avec la manipulation des images *bitmap*. Vous devez écrire un programme qui prend en paramètre un nom de fichier, et affiche les informations. **Vous n'avez pas le droit d'utiliser l'API `.NET` pour cet exercice**, vous devez lire le fichier avec `BinaryReader`.

8.1 Gestion des arguments

Vous devez gérer le cas où l'utilisateur appelle votre programme avec des arguments erronés. Le programme prend seulement un nom de fichier en argument.

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

— Si aucun ou plus de deux arguments sont utilisés affichez une aide :

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

— Si après vérification, le chemin du fichier s'avère incorrect :

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

8.2 Vérification du *magic number*

On vérifie dans un premier temps le *magic number*. Si celui ci ne correspond pas à un fichier image *bitmap*, le programme s'arrête et affiche un message d'erreur :
— Chemin complet vers le fichier.
— Indiquer à l'utilisateur que le fichier n'est pas un `BMP`.

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

8.3 Affichage des informations

Si aucune erreur ne s'est produite, vous devez afficher :
— La dimension de l'image
— Le nombre de bits pour coder un pixel
Affichez les informations de la manière suivante :

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

N'oubliez pas de mettre un caractère de retour à la ligne même si c'est la dernière ligne.

9 Exercice 2 : `DataHiding`

Le but de cet exercice est de vous faire découvrir un autre moyen de transférer un message à la manière de notre bon et vieil ami 007, à travers la stéganographie.

9.1 Petite histoire

Il y a peu, vous avez découvert un moyen d'envoyer un message de façon chiffrée, grâce à la cryptographie. Cette méthode consistait à changer le message originel pour cacher son contenu au personnes ne connaissant pas la clé pour le découvrir. Cette méthode, très utile, n'est pas vraiment discrète.

La stéganographie elle, consiste à cacher un message dans un autre message, comme dans une photo de vacances, ou dans une image de chat ! La stéganographie peut être réalisée dans le monde réel en utilisant par exemple de l'encre faite de jus de citron, visible seulement à la chaleur. La stéganographie permet de transmettre une donnée sans qu'elle soit trouvée par n'importe qui et sans qu'elle soit illisible.

Aujourd'hui, nous allons apprendre à cacher un message dans une image.

9.2 Stégano dans un bitmap 24 bits

Dans le format bitmap 24 bits que nous allons utiliser, chaque pixel est représenté par 24 bits. Dans ces 24 bits, chaque composante (rouge, vert et bleu) est codée sur 8 bits.

Pour cacher nos informations, nous allons exploiter le fait que 2 couleurs trop proches sont quasiment indiscernables par l'oeil humain. Nous allons donc utiliser le dernier bit de chaque octet pour cacher nos données. Il s'agit du bit de poids faible : le modifier revient à modifier l'octet de moins de 0,4%, tandis que modifier le premier bit, le bit de poids fort, aurait fait par exemple passer 50 à 178, soit de 50%, ce qui est très voyant comme différence.

Ainsi, si on veut cacher `5`, `101` en binaire, dans un pixel `(240, 65, 169)`, en binaire `(11110000, 00100001, 10101011)` on va prendre chacun des bits de `5` et les mettres au poids faible des composantes de ces pixels. Le premier est un `1`, la première composante prend donc un `1` en poids faible, devenant donc `241` (`11110001` en binaire). Le deuxième est un `0`, la deuxième composante prend donc un `0` en poids faible, devenant donc `64` (`00100000` en binaire). Le dernier est un `1`, la troisième composante prend donc un `1` en poids faible, ce qui ne change rien.

Nous, on va cacher de texte. Le texte, ce n'est rien d'autre qu'une suite de caractères `ASCII`, des `char`s, qui comme nous le savons bien, sont codés sur un octet. Il faut donc 8 composantes, soit presque trois pixels, pour cacher un unique caractère.

9.3 Let's go ! !

Pour cet exercice, vous êtes vraiment libre de choisir votre méthode, il va bientôt falloir voler de vos propres ailes !

Vous allez donc devoir écrire un programme, il devra permettre à l'utilisateur de choisir une image, de la charger, et de récupérer le message caché à l'intérieur. Ce programme aura bien sur une seconde fonction, celle de cacher un message dans une image. Bien sûr, vous serez tout de même guidé un minimum pour réussir cet exercice, on sait très bien que vous ne savez pas encore voler !

9.3.1 Bien démarrer

Cette partie sera consacrée à vous guider pour réaliser ce programme.

Premièrement, nous vous conseillons de créer un projet *Windows Form*, cela vous permettra d'avoir une interface rapide et un grand nombre de possibilités au niveau de l'aspect de votre programme.

Pour notre exemple, nous dirons que votre interface comporte trois bouton et une textBox. Un bouton pour charger l'image, un bouton (nommé `hideButton`) pour cacher le message écrit dans

la textBox dans une copie de l'image chargée appelée `result.bmp` et enfin un bouton (nommé `revealButton`) pour lire le message caché dans l'image chargée.

9.3.2 Image, image ... montre moi ton message !

Maintenant que l'interface est posée, vous allez coder la fonction suivante créée en double cliquant sur le bouton `revealButton` :

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

Cette fonction est appelée lorsque le bouton est cliqué, il faut donc vérifier au début de la fonction que l'image est bien chargée. Ensuite, il faut parcourir l'image ligne par ligne pour récupérer l'ensemble du message. On va définir qu'un message se terminera toujours par un `'\0'` ou caractère nul, correspondant au caractère en position 0 dans la table `ASCII`.

N'oubliez pas le TP des boucles, en particulier les doubles boucles `for`, elles vous seront utiles. Le message récupéré sera affiché dans la `textBox`.

Astuce : le bit de poids faible correspond à la parité d'un nombre. Pour 35, bit à 1, pour 20, bit à 0.

9.3.3 Votre mission si vous l'acceptez !

On sait maintenant récupérer un message caché, voyons comment cacher le nôtre.

Vous allez coder la fonction suivante créée en double cliquant sur le bouton `hideButton` :

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

Là encore, il faut vérifier qu'une image est chargée, ensuite il faut récupérer le message écrit dans la `textBox`, et pour chaque caractère, récupérer le code `ASCII` correspondant et le convertir en binaire. Une fois binarisé, le caractère peut être inséré dans l'image. Ne pas oublier le `'\0'` à la fin du message.

10 Exercice 3 : `GameMap`

Cet exercice consiste à modéliser une carte de jeu en utilisant le format `BMP`, ce qui permettra de créer très facilement des cartes sans avoir à concevoir un outil spécialisé dans l'édition de carte. On peut imaginer un système de carte avec plusieurs couches :

- une couche d'arrière-plan permettant de dessiner la carte (type de sprite à dessiner) ;
- une couche d'évènement indiquant en fonction de la couleur un évènement particulier (combat, dialogue, etc ...) ;
- une couche permettant de positionner des objets sur la carte.

On peut imaginer d'autres combinaisons de couche avec ce système (et peut rester valable dans un jeu en 3D). L'usage de cette technique permet bien évidemment d'alléger dans un premier temps la taille des ressources graphiques du jeu en préfèrant utiliser plusieurs petites images pour en constituer une plus grande.

Pour éviter d'avoir un résultat trop lourd, cet exercice va à l'essentiel. Le résultat demandé est d'afficher l'image `BMP` dans la fenêtre `XNA`.

FIGURE 1 – Image BMP de taille 10x10 représentant la carte.

10.1 Préparation

Créez un projet XNA [2] nommé GameMap dans Visual Studio 2010. Il faut ajouter ensuite le composant System.Drawing [3] au projet. Dans le *Solution explorer*, clic droit sur *References*, *Add Reference...* et cherchez dans l'onglet .NET. Pour terminer ajoutez au début du fichier Game1.cs :

Code source
```
using System.Drawing;
```

Nous allons afficher avec XNA des rectangles. XNA nécessite à chaque instruction d'affichage une Texture2D. Créez une image BMP avec un logiciel d'imagerie :

— Nom du fichier : blank.bmp
— Dimensions : 1x1
— Couleur du pixel : blanc

Ajoutez l'image dans Visual Studio, dans la section GameMapContent.

Créez également une image BMP pour représenter la carte. Ajoutez là également dans la section GameMapContent et configurez VS pour que l'image soit copiée dans le répertoire de l'exécutable : Properties et Copy always pour Copy to Output Directory.

10.2 Implémentation

Ajoutez des attributs à la classe Game1 :

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

L'image BMP représentant la carte se situe à l'emplacement suivant, la propriété RootDirectory désigne le chemin vers le dossier Content (situé au niveau de l'exécutable produit) :

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

Chargez l'image BMP et la Texture2D (*cf. le cours de ce TP*) dans la méthode :

Code source
```
protected override void LoadContent();
```

2. En cas de problème avec XNA, il s'agit probablement du pilote de la carte graphique de votre ordinateur qui n'est pas à jour. Sur un **RACK** récupérez le pilote sur le site d'Intel : https://downloadcenter.intel.com/default.aspx. Vous devriez trouver Intel® Iris™ and HD graphics Driver for Windows 7/8/8.1* 64. Essayez également de mettre le projet XNA en mode Reach dans les propriétés du projet.

3. Ce composant ajoute une autre référence à la classe Color, il faudra préciser le *namespace* adéquat, par exemple : System.Drawing.Color.

Pour finir, implémentez la méthode suivante dans la classe `Game1`, celle-ci lit le fichier `BMP` et affiche les rectangles :

Code source
```
/* Draw the map on the screen. */
private void DrawMap();
```

— Pensez à utiliser les propriétés `Width` et `Height` de la classe `Bitmap`.
— Créez des instances de la classe `Microsoft.Xna.Framework.Color` en récupérant la valeur des pixels de la classe `System.Drawing.Color`, utilisez les propriétés `R`, `G` et `B`.
— Récupérez les dimensions en pixel de la fenêtre `XNA` de cette manière (utile pour connaitre la largeur et la hauteur d'un rectangle dans la fenêtre) :

Code source
```
int screen_height = this.GraphicsDevice.Viewport.Height;
int screen_width = this.GraphicsDevice.Viewport.Width;
```

Ajoutez trois lignes dans la méthode suivante, pour dessiner la carte :

Code source
```
protected override void Draw(GameTime gameTime);
```

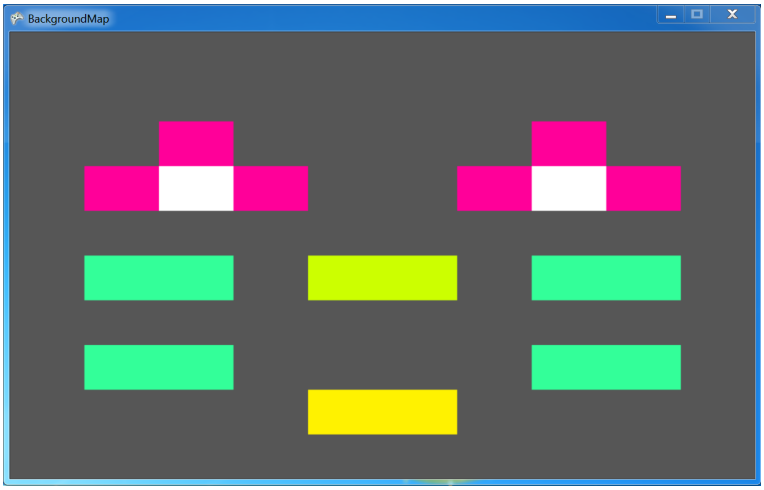

FIGURE 2 – Affichage de la carte dans XNA.