TP C#8 : Trap a Weasley

Consignes de rendu

À la fin de ce TP, vous devrez rendre un dépôt Git respectant l'architecture suivante :

```
csharp-tp08-firstname.lastname/
|-- README
|-- .gitignore
|-- Trap_a_Weasley/
    |-- Trap_a_Weasley.sln
    |-- Crypto/
        |-- Program.cs
        |-- Substitution.cs
        |-- Transposition.cs
        |-- Utils.cs
        |-- Vigenere.cs
        |-- Everything except bin/ and obj/
    |-- Steganography/
        |-- Embed.cs
        |-- Extract.cs
        |-- Program.cs
        |-- Steganography.csproj
        |-- Utils.cs
        |-- Everything except bin/ and obj/
```

Important

Pour ce TP, vous **DEVEZ** télécharger les fichiers donnés sur la page Moodle. Ces fichiers contiennent les images et les prototypes des fonctions que vous allez devoir implémenter. Si vous décidez de ne pas les utiliser, vous ne pourrez peut etre pas utiliser la classe `Bitmap`, et aucune aide ne sera donnée pas les ACDCs.

N'oubliez pas de vérifier les points suivants avant de rendre :

— Remplacez `prenom.nom` par votre propre login.
— Le fichier `README` est obligatoire.
— Pas de dossiers `bin` ou `obj` dans le projet.
— Respectez scrupuleusement les prototypes demandés.
— Retirez tous les tests de votre code.
— **Le code doit compiler !**

README

Vous devez écrire dans ce fichier tout commentaire sur le TP, votre travail, ou plus généralement vos forces / faiblesses, vous devez lister et expliquer tous les boni que vous aurez implémentés. Un `README` vide sera considéré comme une archive invalide (malus).

1 Introduction

1.1 Objectifs

A la fin de ce TP, vous serez familiers avec quelques concepts de cryptographie, et de stéganographie. Pour cela, vous devrez utiliser les classes `Bitmap` et `Color` avec lesquelles vous avez déjà travaillé au précédent TP.

2 Cours

2.1 Cryptographie

La pluspart d'entre vous avez déjà eu affaire à la cryptographie durant votre S1, grâce au projet d'AFIT. Durant ce TP, vous allez implémenter différents algorithmes de cryptographie dans le langage C#.

2.1.1 Substitution

La cryptographie par substitution est un technique simple : remplacer un caractère par un autre prédéterminé. Cette technique n'est pas très sécurisé car il suffit de trouver la correspondance entre les caractères pour décrypter le message.

2.1.2 Algorithme de Vigenere

L'algorithme de Vigenere est une technique qui associe chaque lettre du message à une lettre de la clé.

```
1     THIS TP IS THE BEST  // MESSAGE
2   + INFO IN FO INF OINF  // KEY : INFO
3   ----------------------
4     BUNG BC NG BUJ PMFY
```

2.1.3 Transposition matricielle

Cet algorithme est basé sur les transpositions matricielles. La première étape est de créer une matrice avec l'aide d'une règle de permutation. Cette règle est l'ordre alphabétique des lettres de la clé. La matrice doit être suffisament grande pour contenir le message et avoir une largeur égale à la taille de la clé.
L'étape suivante est de récupérer les lettres du messages dans la matrice. Regardons un exemple avec pour clé : "Ron" et pour message "be aware of the rat" :

Clé de permutation : (R, O, N) -> (3, 2, 1)

Taille du message : 15
Taille de la clé : 3
Taille de la matrice : 5 x 3

Clé de permutation	3	2	1
	B	E	A
	W	A	R
	E	O	F
	T	H	E
	R	A	T

On prend chaque colonne une par une dans l'ordre de la règle de permutation :

ARFET EAOHA BWETR

2.2 Stéganographie

2.2.1 Intéret

La stéganographie est un autre type d'encryption pour un message spécifique. Contrairement à la cryptographie, la stéganographie n'a pas pour but de rendre le message illisible. Le but de cette méthode est de cacher un message au sein d'un autre message. La stéganographie peut etre appliquée à tout type de message, en allant des vidéos à l'audio, cependant la méthode qui nous intéresse se concentre sur les images.

2.2.2 Rappels

Vous savez déjà heureusement utiliser les classes `Bitmap` et `Color` grace au TP précédent. Voici quelques rappels qui pourraient être utiles.

```
1  // Crée un nouvel objet Bitmap depuis un fichier .jpg.
2  Bitmap image = new Bitmap("harry.jpg");
3
4  // Donne la largeur de l'image.
5  int width = image.Width;
6  // Donne la hauteur de l'image.
7  int height = image.Height;
8
9  // Donne la couleur du pixel en haut à gauche de l'image.
10 Color pixel_color = image.GetPixel(0, 0);
11
12 // Crée la couleur blanche.
13 Color white_color = Color.FromArgb(255, 255, 255);
14
15 // Applique la couleur blanche au pixel en haut à gauche de l'image.
16 Color pixel_color = image.SetPixel(0, 0, white_color);
17
18 // Sauvegarde l'image dans le fichier nommé 'ron.jpg'.
19 image.Save("ron.jpg");
```

Ce code modifie la couleur du pixel en haut a gauche de l'image en blanc et la sauvegarde dans le fichier 'ron.jpg'.

Attention

Soyez surs d'être à l'aise avec les objets mentionnés dans l'objet ci-dessus. Tout ce qui a été utilisé dans l'exemple sera utile pour la partie exercice du TP.

2.3 La méthode de Pixel Value Differencing

2.3.1 Introduction

Le but de la deuxième partie du TP est d'implémenter une version simplifiée de la méthode de Pixel Value Differencing (PVD), étant une méthode de stéganographie servant à cacher du

texte dans une image.

Attention

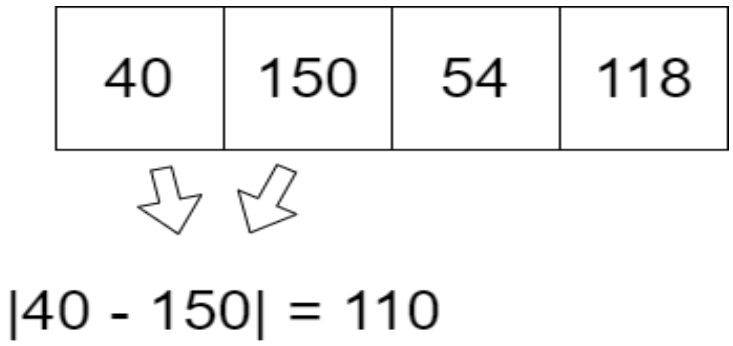
La version que vous allez implémenter est une version simplifiée. Vous DEVEZ suivre les directives du sujet et non ce que vous trouvez en ligne.

La méthode de PVD se base sur des images en nuances de gris. Cela signifie que nous allons travailler avec des images ou les valeurs RGB des pixels seront identiques. Le but de la méthode est de cacher l'information dans les bits de poids faible des pixels (LSB). Mais comment déterminer le nombre de bits que l'on peut changer sans altérer visuellement l'image ? C'est ici que la méthode prend tout son sens.

2.3.2 Intégration

Part. 1

La partie d'intégration du message est faite en prenant des paires de pixels, et en comparant leurs couleurs dans un premier temps (souvenez vous que nous travaillons sur des images en nuances de gris !). Nous pouvons donc calculer la différence de couleurs entre les deux pixels de la paire comme suit.

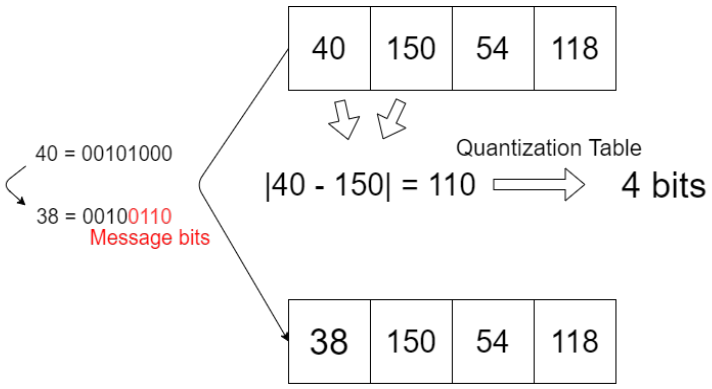


Part. 2

Maintenant que nous avons la différence de couleur entre les deux pixels de la paire, nous pouvons, grace à la table de quantification, déterminer le nombre maximum de bits que nous pouvons changer dans le premier pixel sans altérer l'image. Cette table de quantification est **très** simplifiée par rapport à celle de la méthode originale. La table de quantification est donnée dans la tarball, et voici à quoi elle ressemble.

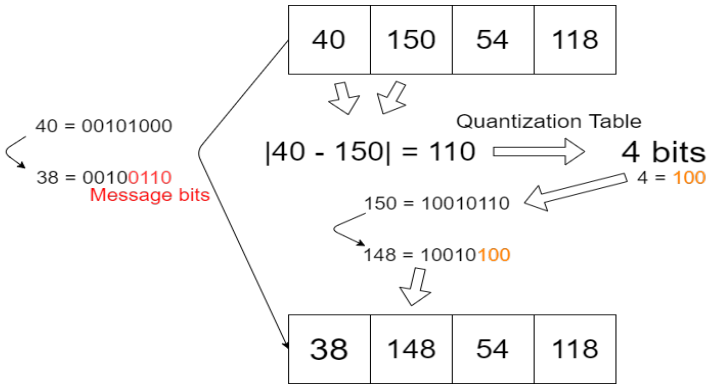
DIFFERENCE	BITS
[0, 1]	1
[2, 32]	2
[33, 64]	3
[65, 255]	4

Une fois le bon nombre de pixels déterminé, nous pouvons modifier les LSB du premier pixel de la paire. Voici ci-dessous un exemple ou nous supposons que nous voulons cacher les bits `0110`.



Part. 3

Mais une question se pose maintenant pour préparer l'extraction du message, comment savoir le nombre de bits qui a été modifié dans le premier pixel ? C'est ici que nous allons grandement simplifier la méthode originale, et placer le nombre de bits intégrés précédemment sur les 3 bits de poids les plus faibles du second pixel de la paire. Nous allons **TOUJOURS** sauvegarder cette valeur sur 3 bits (car elle est en effet contenue entre `001` = 1 et `100` = 4). Voici, en continuant l'exemple, comment cela se met en place.



Part. 4

Nous avons fini ! Nous pouvons maintenant aller sur la paire de pixels suivante et y intégrer la suite du message.

Attention

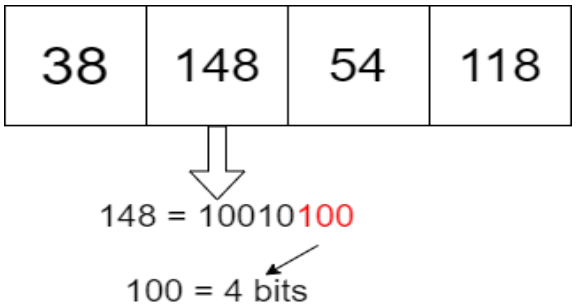
La partie d'intégration est la plus difficile. C'est la raison pour laquelle nous ne vous demandons pas de la coder entièrement, mais comprendre son fonctionnement est très important pour bien réussir le TP.

2.3.3 Extracting

Maintenant que nous avons intégrer notre message dans l'image avec succès, il est temps de l'extraire ! Pour le faire, nous allons également traverser l'image par paire de pixels. Voici les étapes d'extraction.

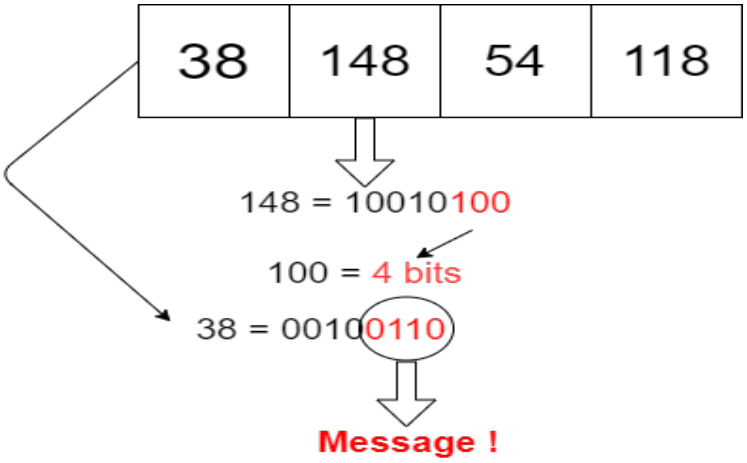
Part. 1

Nous devons dans un premier temps trouver le nombre de pixels cachés dans la paire que nous sommes en train de regarder. Cette information est cachée dans les 3 bits de poids faible du second pixel. Il est donc assez facile d'extraire cette information. En reprenant notre exemple, voici ce que nous obtenons.



Part. 2

Maintenant que nous avons trouvé le bon nombre de bits à extraire, il suffit de les extraire du premier pixel de la paire. Le message étant bien entendu situé sur les bits de poids faible du premier pixel. Voici l'extraction appliquée à notre exemple.



Part. 3

Une fois le message extrait correctement, nous pouvons itérer sur la paire de pixels suivant tant que le message entier n'a pas été décodé.

Tip

Cette explication de la méthode de PVD simplifiée n'est **PAS** simple. Nous vous conseillons de relire cette partie plusieurs fois et de ne pas hésiter à poser des questions si nécessaire.

3 Crypto

Bon retour à Poudlard jeune sorcier et félicitations pour votre promotion en tant que préfet ! Vous êtes désormais les yeux de Dumbledore et il en va de votre responsabilité de veiller à ce qu'aucun méfait ne se déroule en ces murs.

Après la cérémonie de répartition, vous voyez les jumeaux Weasley, les pires farceurs, faire une sortie discrète et décidez de les suivre. Vous courrez après eux mais ne trouvez qu'un morceau de papier avec votre nom et ce message :

" Cher préfet, en ce nouveau départ, nous sommes là pour vous proposer un petit accord, stoppez notre prochain coup et aucune farce ne sera faite contre vous ou votre maison durant une année entière. Mais si vous en parlez à qui que ce soit ou travaillez avec quelqu'un, nous ferons de vous notre unique cible.
Si vous êtes prêt à relever le défi, rendez-vous dans la salle de bain des préfets où le premier indice vous attend. Bonne chance, vous avez jusqu'à 19h ce soir !
F & G"

3.1 Substitution

Lorsque enfin vous arrivez dans la magnifique salle de bain des préfets, vous entendez un bruit : "toc toctoc toc..." répétant un cycle clair. Ce bruit provient d'un balai enchanté. Après quelques minutes de réflection, vous comprenez : ce bruit est l'indice, un message en Morse !

C'est à vous de trouvez le sort vous permettant de décoder ce message :

"–. — - — -.. .. ...- .. -. .- - .. — -. .-. — — –"

3.1.1 Morse décryption

Ecrivez une fonction qui décode un message en morse et retourne null si le message est incorrect. Utilisez la substitution.

```
1  public static string Morse_decode(string message);
2
3  Morse_decode(".- -...") // AB
```

Tip

Utilisez le Dictionnaire donné dans Utils.cs

3.1.2 Morse encryption

Maintenant que vous avez le premier message, personne d'autre ne doit le voir car sinon vous devrez faire face au danger d'avoir des jumeaux Weasley pour ennemis. Vous devez enchanter à nouveau ce balai !

Ecrivez une fonction qui encode un message en Morse et retourne null si le message est incorrect.

```
1  public static string Morse_encode(string message);
2
3  Morse_encode("ACDC"); // .- -.-. -.. -.-.
```

3.2 Vigenere

On dirait que les Weasley aiment vous faire courir, vous courez à travers Poudlard en suivant le dernier indice et arrivez à l'emplacement suivant. Dedans, une table avec une unique boule de cristal dessus. En vous approchant, vous pouvez y distinguer :

"BWP LVD ZZSNW BCM AEQB CAH QN QA TAM YVWKCMA WBBV GKM ZTSEL / La clé est le cours enseignée ici"

3.2.1 Vigenere décryption

Ecrivez une fonction qui décode un message encrypté grâce à l'algorithme de vigenere

```
public static string Vigenere_decode(string message, string key);

Vigenere_decode("BUNG BC NG BUJ PMFY", "INFO") // "THIS TP IS THE BEST"
//   BUNG BC NG BUJ PMFY
// - INFO IN FO INF OINF
```

3.2.2 Vigenere encryption

Ecrivez une fonction qui encode un message grâce à l'algorithme de Vigenere.

```
public static string Vigenere_encode(string message, string key);

Vigenere_encode("This TP is the best", "info") //  BUNG BC NG BUJ PMFY
//   This TP is the best
// + info in fo inf oinf
```

Tip

Vous pouvez utilisez la méthod ToUpper()

3.3 Transposition

L'endroit suivant est un très bon endroit pour faire une pause dans votre chasse pour les jumeaux mais..."tic toc tic toc" l'horloge tourne et les Weasley sont introuvables. Vous commencez vos recherches lorsqu'une petite voix s'élève derrière vous :
" Dobby est hereux de rencontrer une nouvelle personne ! Dobby a vu des amis aujourd'hui ! Ami de Dobby lui ont donné un cadeau ! "
Vous n'avez pas le temps de répondre avant que l'elfe n'agite une chaussette sous votre nez avec un message dessus :

"EWEAIETFLE NITSNPIDE TXLE,TIGMO HTBLFHNOBR ELHTDANUD" / La clé est le nom d'un ami proche de vous"

3.3.1 Règle de permutation

Ecrivez une fonction qui prend un mot et le transforme en une règle de permutation.

```
public static int[] Permutation_rule(string key);

Permutation_rule("GOAL"); // [2, 4, 1, 3]
Permutation_rule("Authority"); // [1, 8, 6, 2, 4, 5, 3, 7, 9]
```

Tip

Vous aurez probablement besoin d'une fonction pour récupérer l'index de chaque position pour la suite.

3.3.2 Décryptage

Créez la table de décryption

Ecrivez la fonction qui créer la matrice nécessaire pour décrypter le message. Le message aura un espace entre chaque colonne à l'exception du cas où le message remplit complètement le tableau.

```
public static char[,] Create_table_decrypt(string message, string key,
int size);

Create_table_decrypt("CTARMEYOGTRILISNPNOH", "GOAL", 5);
// [[E, N, C, R],
//  [Y, P, T, I],
//  [O, N, A, L],
//  [G, O, R, I],
//  [T, H, M, S]]
```

Tip

La matrice sera de taille : size_de_la_clé x size.

Décryption

Ecrire la fonction qui décrypte un message grâce à la transposition matricielle.

```
public static string Permutation_decrypt(string message, string key);

Permutation_decrypt("CTARMEYOGTRILISNPNOH", "GOAL");
// ENCRYPTIONALGORITHMS
```

Créez la table d'encryption

Ecrire une fonction qui crée la matrice nécessaire pour l'encryption du message.

```
public static char[,] Create_table_decrypt(string message, string key,
int size);

Create_table_decrypt("CTARMEYOGTRILISNPNOH", "GOAL", 5);
// [[E, N, C, R],
//  [Y, P, T, I],
//  [O, N, A, L],
//  [G, O, R, I],
//  [T, H, M, S]]
```

Encryption

Ecrire une fonction pour encrypter un message grâce à la permutation matricielle.

```
1 public static string Permutation_encrypt(string message, string key)
2
3 Permutation_encrypt("ALGE WETT RLER EAOH", "GOAL");
4 // WE ARE ALL TOGETHER
```

4 Stéganographie

Vous cherchez entièrement le château sans trouver la peinture mentionnée par le dernier indice. Où est-elle ? L'ultimatum approche lorsque vous vous souvenez enfin, dans les temps, où vous pouvez le trouver : le bureau de Dumbledore. Après vous être infiltré dedans, vous trouvez ce que vous cherchiez : décrypter l'image fournie.

Vous allez maintenant coder une partie de l'intégration et l'extraction de la méthode de PVD présentée dans le cours.

4.1 Utils.cs

4.1.1 Fonctions données

Le fichier `Utils.cs` est utilisé pour avoir toutes les fonctions nécessaires à l'implémentation de la méthode de PVD. Certaines fonctions sont déja données tandis que d'autres vous sont demandées.

```
1 static public Dictionary<int[] , int> QuantizationTable;
```

Cette table de quantification est utilisée pour obtenir le nombre de bits du message à intégrer à la paire de pixel. Elle est donnée dans la tarball.

```
1 public static int ClearLSB(int color, int nbBits);
```

La fonction `ClearLSB` est utilisée pour mettre à 0 les `nbBits` bits de poids faible de l'entier `color`. Comme cette fonction se rapproche grandement de ce que vous avez fait au TP précédent, elle est donnée dans la tarball.

```
1 public static int ReplaceLSB(int color, int nbBits, int newLSB);
```

La fonction `ReplaceLSB` est utilisée pour remplacer les `nbBits` bits de poids faible de l'entier `color` par les `nbBits` bits de poids faible de l'entier `newLSB`. Comme cette fonction se rapproche grandement de ce que vous avez fait au TP précédent, elle est donnée dans la tarball.

```
1 public static int SaveLSB(int color, int nbBits);
```

La fonction `SaveLSB` est utilisée pour garder uniquement les `nbBits` bits de poids faible de l'entier `color`. Comme cette fonction se rapproche grandement de ce que vous avez fait au TP précédent, elle est donnée dans la tarball.

```
1 // Donne un nouvel objet Bitmap depuis le fichier situé à path.
2 public static Bitmap OpenImage(string path);
3
4 // Ferme l'image après l'avoir sauvegardée sous le nom name.
5 public static void SaveImage(string name, Bitmap image);
```

Ces fonctions sont des fonctions de base d'image processing que vous avez déjà vu, elles sont données dans la tarball.

4.1.2 TextToBin

La fonction TextToBin permet de transformer un string en un tableau de bits, correspondant à la concaténation de chaque code ASCII de chaque caractère en binaire. Comme chaque caractère possède un code ASCII, il suffit de le récuprérer, le transformer en binaire, ce qui prendra toujours 8 bits dans ce cas, pour au final l'insérer dans le tableau. Voici son prototype avec quelques exemples.

```
1  public static int[] TextToBin(string secret);
2
3  TextToBin("");
4  // => []
5
6  TextToBin("a");
7  // => [0, 1, 1, 0, 0, 0, 0, 1] (Car a = 97 = 01100001 en binaire.)
8
9  TextToBin("ab");
10 // => [0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0]
11 // (Car a = 97 = 01100001 et b = 98 = 01100010 en binaire.)
```

Tip

Vous pouvez simplement faire les conversions à l'aide des opérateurs binaires « et/ou » !

4.1.3 BinToText

La fonction BinToText fait l'exact opposé de TextToBin. Elle crée un string a part d'un tableau de bits, qui, lorsque découpé en blocs de 8, donne une suite de code ASCII.

Tip

Le tableau donné en argument sera toujours un multiple de 8 ! Il est inutile de gérer les autres cas.

```
1  public static string BinToText(int[] bin);
```

4.1.4 ExctractBits

La fonction ExtractBits extrait `nbBits` bits en partant de la position index dans le tableau secret. La fonction convertit ensuite en décimal avant de renvoyer la valeur. Voici son prototype et quelques exemples.

```
1  public static int ExtractBits(int[] secret, int index, int nbBits);
2
3  int[] secret = {0, 1, 1, 0};
4  ExtractBits(secret, 0, 2); // = 01 en binaire = 1 en decimal.
5  ExtractBits(secret, 1, 3); // = 110 en binaire = 6 en decimal.
6  ExtractBits(secret, 2, 3); // = 100 en binaire = 4 en decimal.
```

Tip

Si le nombre de bits à extraire est trop grand, il faut agir comme si on rencontrait un 0 dans le tableau.

4.1.5 InsertBits

Cette fonction insère les `nbBits` bits de poids forts de l'entier value à la position index du tableau array. Voici son prototype et quelques exemples.

```
public static void InsertBits(int[] secret, int index, int nbBits,
    int value);

int[] secret = {0, 0, 0, 0};
InsertBits(secret, 0, 3, 7); // 7 = 111 in binary.
// => secret = [1, 1, 1, 0]

int[] secret2 = {0, 0, 0, 0};
InsertBits(secret2, 0, 4, 7); // 7 = 0111 in binary.
// => secret = [0, 1, 1, 1]

int[] secret3 = {0, 0, 0, 0};
InsertBits(secret3, 0, 1, 7); // 7 = 111 in binary.
// => secret = [1, 0, 0, 0]

int[] secret4 = {0, 0, 0, 0};
InsertBits(secret4, 3, 3, 7); // 7 = 111 in binary.
// => secret = [0, 0, 0, 1]
```

Tip

Si le nombre de bits à insérer est trop grand, il suffit d'insérer les premiers bits, les autres sont perdus. Cela est montré dans le dernier exemple.

4.1.6 GetDifference

Cette fonction calcule la différence de couleur entre deux pixels en nuances de gris comme décrit dans la partie Cours du TP sur la méthode de PVD. Voici son prototype et quelques examples.

```
public static int GetDifference(Color pixel1, Color pixel2);

Color c1 = FromArgb(120, 120, 120);
Color c2 = FromArgb(200, 200, 200);

GetDifference(c1, c2); // = 80
GetDifference(c2, c1); // = 80
```

Tip

Comme nous travaillons sur une image en nuance de gris, la couleur d'un pixel est égale à n'importe laquelle de ses composantes RGB de son objet Color. La couleur d'un pixel avec une couleur Color c1 sera donc c1.R = c1.G = c1.B !

4.2 Embed.cs

4.2.1 EmbedMsg

La fonction EmbedMsg est la fonction principale appelée afin d'intégrer un message dans une image. Cette fonction suit à la lettre ce qui est expliqué dans la partie Cours du TP. Voici son prototype.

```
1 public static void EmbedMsg(int[] secret, Bitmap image)
```

La fonction est donnée dans la tarball, mais vous être plus qu'invités à regarder la fonction et la comprendre. L'extraction ressemble sur plusieurs points à l'intégration.

4.2.2 GetBitsToEmbed

La fonction GetBitsToEmbed donne le nombre maximum de bits qu'il est possible de cacher dans le premier pixel de la paire grace à la différence passée en argument. Voici son prototype.

```
1 public static int GetBitsToEmbed(int difference);
```

Tip

Vous pouvez vous aider de la table de quantification donnée dans Utils.cs ainsi que de la partie Cours de ce TP.

4.3 Extract.cs

4.3.1 ExtractMsg

Voici la dernière partie de cet exercice. Vous devez maintenant coder la fonction qui extrait un message de taille length depuis l'image donnée en paramètre. Voici le prototype de la fonction que vous devez implémenter.

```
1 public static int[] ExtractMsg(Bitmap image, int length);
```

La fonction ExtractMsg extrait un message composé de length caractères. Pour implémenter avec succès cette fonction, il est conseillé de suivre ce qui est expliqué dans la partie extraction du Cours de ce TP. Une possible implémentation correcte de la fonction serait la suivante (pour chaque paire de pixels).

- **Etape 0** : Vous aurez probablement besoin de créer le tableau de bits qui va stocker le message extrait de l'image. (Ceci doit se faire avant d'itérer sur les paires !).
- **Etape 1** : Trouver le nombre de bits à extraire du premier pixel grace à l'information sur les 3 bits de poids faible du second pixel.
- **Etape 2** : Extraire le bon nombre de bits du premier pixel.
- **Etape 3** : Insérer ces bits dans le tableau.
- **Etape 4** : Itérer sur la paire de pixels suivante tant que le message n'a pas été totalement extrait (on peut le savoir avec la taille du message à extraire donnée en paramètre !).

4.4 Program.cs

4.4.1 Main

Le main est déjà donné dans les fichiers accessibles sur Moodle afin que vous puissiez tester votre projet. Le main ne fait pas partie du rendu.

4.5 Conclusion

Félicitations ! Vous êtes maintenant capables de cacher et d'extraire des données dans des images en nuances de gris grace à la méthode simplifiée de PVD. Pour tester votre projet, vous pouvez essayer de trouver le message caché dans l'image accessible sur Moodle.

5 Bonus

Trap a Weasley : Ecrire le message caché dans l'image donnée sur Moodle.

Tip

Afin de vous aider à résoudre cette énigme, voici quelques indices :
- Il faut avoir fait tous les exercices pour trouver le message.
- Il faut décrypter les messages donnés en examples dans les exercices.
- EX signifie Exercice.
- L signifie Lettre (cela inclue les espaces !).
- La taille du message caché dans l'image donnée est 94.

Algorithmes : Implémenter un algorithme de cryptographie de votre choix.

Help will always be given at EPITA to those who ask for it