SIM-SysIn : Test Final, partie machine

Seuls documents autorisés : ceux fournis avec la clef USB du test

Durée 1h20

Lire attentivement le sujet...

Le travail sur machine est à faire sur la **clef USB** remise par l'enseignant, que vous devrez rendre en fin de session après une extraction sécurisée. L'évaluation de votre travail sera basée sur l'examen des fichiers que vous aurez travaillés sur la clef USB.

Vous devez réaliser, dans l'ordre, les actions suivantes :

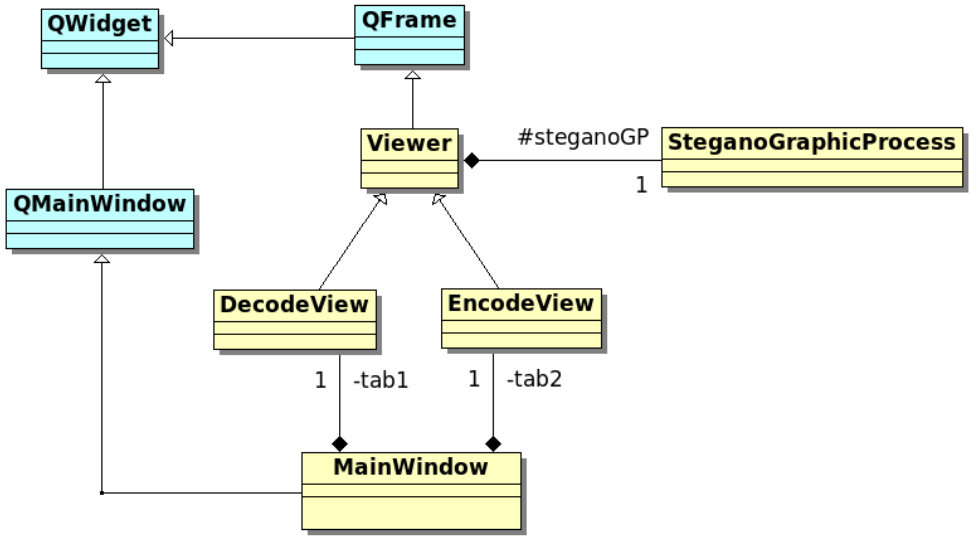
1 Renommage

Renommer le répertoire `TestFinalSysIn` de la clef USB en utilisant votre `Nom.prenom` comme nom de répertoire (pas d'accent, pas d'espace, pas de caractère de ponctuation !).

Le répertoire maintenant renommé `Nom.prenom` contient une application PyQT de traitement stéganographique. La stéganographie est l'art de dissimuler un texte dans une image, sans modification apparente de l'image.

L'application PyQt est constituée de plusieurs fichiers *Python* :
- les fichiers `main.py`, `Viewer.py`, `EncodeView.py` et `DecodeView.py`, pour la définition de l'interface graphique,
- le fichier `SteganoGraphic.py` pour la définition des traitements stéganographiques permettant de cacher un texte dans une image ou, à l'inverse, de retrouver un texte caché dans une image.

Le diagramme UML simplifié des relations entre les classes de l'application est le suivant :



2 Visite rapide de la classe `DecodeView`

1. Ouvrir le fichier `DecodeView.py` avec l'éditeur IDLE. Balayer rapidement le code, lire la méthode `Decode()`.
2. Vérifier l'intégrité du fichier en l'exécutant avec la touche F5 : aucune erreur ne doit apparaître dans le shell Python.

3 Visite rapide de la classe `EncodeView`

1. Ouvrir le fichier `EncodeView.py` avec l'éditeur IDLE. Balayer rapidement le code, lire la méthode `Encode()`.
2. Vérifier l'intégrité du fichier en l'exécutant avec la touche F5 : aucune erreur ne doit apparaître dans le shell Python.

4 La Classe `SteganoGraphicProcess`

Le principe du texte caché dans une image

Considérons un texte simple comme 'Bonjour' (nous l'appellerons par la suite `textToHide`). Il est possible de cacher les caractères de ce texte dans une image bitmap (tableau de pixels à `nb_lignes` et `nb_colonnes`) en suivant les étapes suivantes :

1. On convient de cacher `textToHide` au début du tableau des couleurs rouges des pixels de l'image bitmap (tableau `Red`, à 2 dimensions, contenant des valeurs comprises entre 0 et 255).
   Pour faciliter les écritures, on utilise un tableau de travail à 1 seule dimension contenant les lignes du tableau `Red` mises bout à bout. Ce tableau est nommé `Pix`, il contient `N = nb_lignes`×`nb_colonnes` valeurs entières comprises entre 0 et 255 (ce tableau est défini à la fin de la méthode `LoadQImage()` de la classe `SteganoGraphicProcess`);
2. On convient de représenter le nombre de caractères `N` de `textToHide` par un octet, permettant de considérer jusqu'à 255 caractères au maximum. Cet octet sera caché au tout début de l'image (8 premiers pixels), puis les caractères de `textToHide` seront également cachés chacun à leur tour dans les pixels suivants ;
3. La première étape du procédé est un **pré-traitement** consistant à remplacer chacune des valeurs du tableau `Pix` par la valeur de l'entier pair inférieur le plus proche (35 est remplacé par 34, 112 reste 112...) ;
4. On peut alors dissimuler les 8 bits d'un octet quelconque (représentant l'entier `N` ou un caractère du texte) dans une suite de 8 éléments consécutifs de `Pix` : on rajoute la valeur de chaque bit (0 ou 1) à la valeur de l'élément correspondant de Pix. Ainsi, les valeurs du tableau `Pix` sont "à peine modifiées" (± 1 pour des valeurs comprises entre 0 et 255).

À la fin du traitement, le tableau `Pix` est re-versé dans le tableau des rouges, et l'image est ré-enregistrée (ce travail est déjà fait au début de la méthode privée `ConvertToQImage()` de la classe `SteganoGraphicProcess`).

Travail à faire

1. Ouvrir le fichier `SteganoGraphic.py` avec l'éditeur IDLE. Balayer rapidement le code.
2. **Vérifier l'intégrité du fichier en l'exécutant une première fois avec la touche F5.**
3. Fermer les fichiers `DecodeView.py` et `EncodeView.py`.

Recherche du texte caché dans une Image

4.1 Programmation dans la classe `SteganoGraphicProcess`

Compléter la méthode privée `FindByteIn8Pixels()`

Cette méthode retrouve un octet caché dans une suite de 8 valeurs entières comprises entre 0 et 255.

1. **Programmation** - L'algorithme reconstitue bit par bit la représentation binaire de l'octet caché dans les 8 éléments du tableau `pixels` à partir du rang `atRank` (si un élément est pair, le bit est `'0'`, si il est impair le bit est `'1'`), puis il retourne la valeur entière correspondant aux 8 bits trouvés :

```
01 FindByteIn8Pixels(pixels, atRank):
02   bits <- chaîne vide
02   Pour n allant de 0 à 7, Faire:
03     bits <- concaténation de bits et de str(pixels[atRank + n] % 2)
04   Fin pour
05 retourner (atRank+8, bits converti en base 10)
```

   indications Python :
   - `bits` est une chaîne de caractères faite de `'0'` et de `'1'`, une chaîne vide est définie avec 2 guillemets consécutifs
   - `str()` est la fonction Python qui convertit un nombre en chiffres (`0` devient `'0'` et `1` devient `'1'`),
   - la concaténation des caractères s'écrit simplement avec l'opérateur `+`,
   - la méthode retourne `atRank + 8` de façon à pouvoir continuer de balayer le tableau `Pix`,
   - `int(a,2)` convertit en base 10 la chaîne de '0' et '1' qui est dans `a`.
2. **Test** - Vérifier avec F5 l'absence d'erreur de syntaxe.

Compléter la méthode privée `FindCharIn8Pixels()`

Cette méthode retrouve un caractère caché dans une suite de 8 valeurs entières comprises entre 0 et 255.

1. **Programmation** - Utiliser la méthode `FindByteIn8Pixels()` et convertir en caractère la valeur retournée :

```
01 FindCharIn8Pixels(pixels, atRank):
02   (atRank, byte) <- FindByteIn8Pixels(pixels, atRank)
02   char <- conversion de byte en caractère
03 retourner atRank, char
```

indications Python :

- `chr()` est la fonction Python qui convertit un octet en caractère
- attention à la syntaxe pour **appeler une méthode de la classe** : voir le code Python déjà écrit (par exemple voir comment `LoadQImage()` appelle la méthode privée `Clear()`).

2. **Tests** - Vérifier avec F5 l'absence d'erreur de syntaxe.

Compléter la méthode publique `FindTextInImage()`

`Méthode` finale qui retrouve un texte caché dans une image en se servant des méthodes privées `FindByteIn8Pixels()`, `FindCharIn8Pixels()` et de la méthode publique `LoadQImage()`. Elle prend un argument, `image`, qui représente un objet de type `QImage`.

1. **Programmation** - Séquence des opérations à coder :

```
01 FindTextInImage(image):
02   exécuter LoadQImage(image)
03   atRank <- 0
04   (atRank, N) <- valeurs retournées par l'appel à FindByteIn8Pixels(Pix, atRank)
05   text <- chaîne vide
06   Pour n allant de 0 à N-1:
07     (atRank, char) <- valeurs retournées par l'appel à FindCharIn8Pixels(Pix, atRank)
08     text <- concaténation de text et de char
09   Fin Pour
10 Retourner text
```

Algorithme 1

indications Python :

- dans le programme, `Pix` s'écrit `self.__Pix` (attribut privé `Pix` de la classe `SteganoGraphicProcess`).

2. **Tests** - Exécuter le fichier avec F5 et vérifier la sortie écran dans le shell Python ; mettre au point si besoin.

Dissimulation d'un texte dans une image

4.2 Programmation dans la classe `SteganoGraphicProcess`

Compléter la méthode publique `PreProcessImage()`

1. **Programmation** - Le pré-traitement a pour but de rendre paires toutes les valeurs du tableau `Pix` :

```
01  PreprocessImage():
02    Pour i allant de 0 à nombre d'éléments de Pix -1 :
02      Si Pix[i] % 2 == 1:
03        Pix[i] <- Pix[i] - 1
04      Fin Si
05    Fin Pour
06  Fin PreprocessImage
```

indications Python :

- dans le programme, `Pix` s'écrit `self.__Pix`,
- le nombre d'éléments de `self.__Pix` est `self.__Pix.size` (`self.__Pix` est un objet de type `ndarray`).

2. **Test** - Exécuter le fichier avec F5 et vérifier la sortie écran dans le shell Python ; mettre au point si besoin.

Compléter la méthode privée `HideByteIn8Pixels()`

Cette méthode (à utiliser après le pré-traitement) cache les 8 bits d'un octet dans une suite de 8 valeurs entières comprises entre 0 et 255.

1. **Programmation** - L'algorithme utilise la décomposition en 8 bits de l'octet à cacher dans les 8 éléments du tableau `Pix` à partir du rang `atRank` (bit `'0'` : l'élément est laissé pair, bit `'1'` : l'élément est rendu impair en lui ajoutant 1) :

```
01 HideByteIn8Pixels(byteToHide, pixels, atRank):
02   binDigit <- valeur retournée par l'appel à bin8bits(byteToHide)
02   Pour n allant de 0 à 7, Faire:
03     Si binDigit[n] == '1':
04       pixels[atRank+n] += 1
05     Fin Si
04   Fin pour
05 retourner atRank+8
```

indications Python :

- `binDigit` est une chaîne de caractères constituée de `'0'` et de `'1'` obtenue en appelant la méthode publique `bin8bits()` de la classe `SteganoGraphicProcess`, avec l'argument `byteToHide` pour le convertir en sa représentation binaire sur 8 bits,
- la fonction retourne `atRank + 8` de façon à pouvoir continuer à balayer le tableau `pixels` si besoin.

2. **Test** - Vérifier avec F5 l'absence d'erreur de syntaxe.

Compléter la méthode privée `HideCharIn8Pixels()`

Cette méthode (à utiliser après le pré-traitement) cache un caractère dans une suite de 8 valeurs entières comprises entre 0 et 255.

1. **Programmation** - Utiliser la méthode `HideByteIn8Pixels()` et convertir sa valeur de retour en caractère :

```
01 HideCharIn8Pixels(charToHide, pixels, atRank):
02   byteToHide <- conversion de charToHide en octet
02   atRank <- valeur retournée par l'appel à HideByteIn8Pixels(byteToHide, pixels, atRank)
03 retourner atRank
```

indications Python :

- `ord()` est la fonction Python qui convertit un caractère en octet.

2. **Tests** - Vérifier avec F5 l'absence d'erreur de syntaxe.

Compléter la méthode publique `HideTextInImage()`

C'est la méthode finale (à utiliser après le pré-traitement) qui cache un texte dans une image en se servant des méthodes privées `HideByteIn8Pixels()`, `HideCharIn8Pixels()` et `ConvertToQImage()`. Elle prend deux arguments : le texte à cacher et un objet de type `QImage`.

1. **Programmation** - Séquence des opérations à implémenter :

```
01 HideTextInImage(textToHide, image):
02   exécuter LoadQImage(image)
03   exécuter PreProcessImage()
04   atRank <- 0
05   N <- nombre de caractères de textToHide
06   atRank <- valeur retournée par l'appel à HideByteIn8Pixels(N, Pix, atRank)
07   Pour n allant de 0 à N-1:
08     atRank <- valeur retournée par l'appel à HideCharIn8Pixels(textToHide[n], Pix, atRank)
09   Fin Pour
10 Retourner la valeur retournée par l'appel à ConvertToQImage()
```

indications Python :

- dans le programme, `Pix` s'écrit `self.__Pix` (**attribut privé** `Pix` la classe `SteganoGraphicProcess`)
- la fonction `len()` permet de connaître le nombre de caractères d'une chaîne,

2. **Tests** - Exécuter le fichier avec F5 et vérifier la sortie écran dans le shell Python ; mettre au point si besoin.

5 Utilisation de l'interface graphique

L'interface graphique permet de cacher un texte dans une image ou d'extraire le texte caché dans une image.

Ouvrir le fichier main.py, et l'exécuter avec F5.

1. Dans l'onglet `'Decode'` charger l'image `test.png` et extraire le texte contenu avec le bouton `Decode`,
2. Dans l'onglet `'Encode'`, cacher le texte "votre prénom - votre nom" dans l'image `perroquet.png` et enregistrer l'image modifiée sous le nom `sysin.png`.

6 Bonus

Améliorations de la méthode `FindTextInImage()` de la classe `SteganoGraphicProcess`

Si on cherche à extraire le texte d'une image qui ne contient pas de texte caché, la ligne 4 de l'*algorithme 1* (page 3) retourne quand même une valeur N comprise entre 0 et 255, qui n'a rien à voir avec le nombre de caractères d'un texte caché. Dans certains cas, ce nombre peut entraîner un dépassement du balayage du tableau Pix dans la boucle de la ligne 6 de l'*algorithme 1*.

Implémenter une amélioration de la méthode `FindTextInImage()` qui corrige ce défaut.

Améliorations de la méthode `Decode()` de la classe `DecodeView`

Si l'image ne contient pas de texte caché mais que la valeur de N est compatible avec le nombre de pixels de l'image, la méthode `Decode()` de la classe `DecodeView` provoque l'affichage de "vilains" messages d'erreur dans le shell Python.

Implémenter une amélioration à base de `try:` et `except:` pour gérer correctement cette situation en affichant un message d'erreur clair dans le shell Python.