|  |
| --- |
| CODE REVIEW |

Groupe réviseur : Groupe 3 (Nicolas Solioz + Valentin Bornatici)

Groupe révisé : Groupe 2 (Alexandre Berclaz + Yann Clavien)

**Introduction**

Dans le cadre du module *« 625-1 Organisation du développement logiciel »,* nous sommes en chargé d’étudier l’organisation du code ainsi que la construction des tests unitaires réalisés par nos collègues. Le logiciel développé devrait permettre d’importer une image, y appliquer un filtre et l’exporter. Voici les critères de développement de l’application, donné par le product owner :

* Keep the possibility to do edge detection
* Add the capability to filter the image (using **two** of the filtering of pixel manipulation software) and then to do image edge detection
* The sequence should be: filter first (**one or many**) then detect the edges
* Enhance the interface to be able to do it **simply**

Dans ce “Code Review”, nous allons vérifier si les user stories du product owner ont été appliqués. Nous allons également déterminer si le code est propre et compréhensible (best practices), ainsi qu’analyser le degré de couverture des tests unitaires. Finalement, nous allons tester l’efficacité de ces tests.

**1. Analyse des user stories**

*1.1 Keep the possibility to do edge detection*

User Story réussie

La détection des bords est possible.

Une image contenant capture d’écran

Description générée automatiquement

*1.2 Add the capability to filter the image (using* ***two*** *of the filtering of pixel manipulation software) and then to do image edge detection*

User Story réussie

Il est possible de filtrer l’image de 3 façons différentes : avec le premier filtre, le 2ème ou les 2.

Une image contenant capture d’écran, herbe, arbre

Description générée automatiquementUne image contenant capture d’écran

Description générée automatiquementUne image contenant capture d’écran, arbre

Description générée automatiquement

Une image contenant capture d’écran, herbe

Description générée automatiquement

*1.3 The sequence should be: filter first (****one or many****) then detect the edges*

User Story réussie

Comme vu ci-dessus, il est possible d’appliquer un ou plusieurs filtres. Nous arrivons également à appliquer une détection de bord à une image filtrée.

Une image contenant capture d’écran, arbre

Description générée automatiquement

*1.4 Enhance the interface to be able to do it* ***simply***

User Story réussie

L’interface est claire et compréhensible.

**2. Expérience de l’utilisateur**

Le template original est globalement conservé, la fenêtre est toutefois élargie d’environ 25% pour accueillir trois nouvelles checkbox.

Le démarrage s’effectue correctement et l’application se lance vite.

Il n’y a aucune indication de quoi faire, un texte dans le champ de l’image aurait été un plus.

**Si aucune image chargée**

Le bouton d’ajout d’image fonctionne bien et démarre un pop-up de sélection de fichier ouvert sur le dossier user\images en autorisant les fichiers .png, .jpg et .bmp, l’ajout des fichiers .jpeg aurait été un plus.

La détection des bords est désactivée tant qu’il n’y a pas d’image sélectionnée.

Le bouton de sauvegarde de l’image (Save Image) est actif mais il n’exécute rien tant qu’aucune image est chargée.

Les checkbox sont actives même sans images, en sélectionner une ferme la fenêtre et lève une exception !

**Avec une image chargée**

Les filtres des checkbox sont dynamiques, l’image est instantanément rechargée par une version filtrée. Une fois un des filtres d’image, none filter inclus, cochés, il est possible de choisir le edge detection.

Le bouton de sauvegarde de l’image (Save Image) est actif et démarre un pop-up de sélection de fichier ouvert sur le dossier user\images en autorisant les fichiers .png, .jpg et .bmp, l’ajout des fichiers .jpeg aurait été un plus. De plus, la sauvegarde peut se faire à tout moment, sans considération des filtres appliqués.

**3. Qualité du code**

Les classes « Matrix.cs » et « ExtBitmap.cs » sont des codes déjà développés et fonctionnels. Cependant, lors de leur relecture nous constatons plusieurs lignes difficilement déchiffrables, par exemple :

Une image contenant capture d’écran, texte

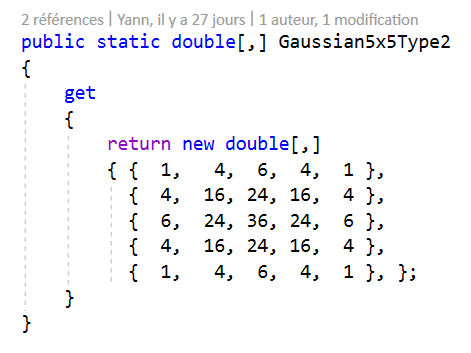
Description générée automatiquement

* Il y’a 4 boucles « for » imbriqué, ce qui rend le déchiffrement du code très difficile. (ConvolutionFilter, ExtBitmap.cs)

Une image contenant capture d’écran

Description générée automatiquement

* Une méthode appel 6 arguments dont un appel à la propre classe (this Bitmap) et 2 matrices. Ce type de variables est complexe. Il serait préférable de séparer les méthodes dans ces cas. (ConvolutionFilter, ExtBitmap.cs)



* Toutes les méthodes de la classe « Matrix.cs » renvoie un tableau de double. L’origine de ces valeurs nous est inconnue, ce sont des valeurs insérées par les premiers développeurs de l’application. (Gaussian5x5Type2, Matrix.cs)

Ces 2 classes (Matrix.cs + ExtBitmap.cs) n’ont aucun commentaire et sont très complexes. Il est difficile de comprendre ce qu’il se passe. Pour ces raisons, nous n’avons pas fait d’analyse de ces classes. Nous conseillons au product owner de renvoyer ce code à l’équipe de développement initial afin qu’il l’éclaircie et le documente.

*3.1 Nomenclature des variables et méthodes*

|  |  |
| --- | --- |
| **Section de code** | **Commentaires** |
| int raz = bmp.Height / 4; | Le nom « raz » n’est pas explicite. On ne sait pas à quoi il fait référence, et il n’est pas utilisé dans le cadre d’une boucle (par exemple : une variable x qui s’incrémente). |
| Une image contenant capture d’écran  Description générée automatiquement | Les noms donnés au formulaire sont les suivants :   * btnOpenOriginal * btnSaveNewImage * cmbEdgeDetection * picPreview * checkBoxNoneFilter * checkBoxRainbowFilter * checkBoxSwapFilter   Tous les noms sont clairs. Dès la première lecture, on comprend la nature et l’utilité de l’élément du formulaire. |
| private void ApplyFilter(bool preview) | Les méthodes ont des noms facilement compréhensibles. |

*3.2 Efficacité des algorithmes / lisibilité*

Une image contenant capture d’écran

Description générée automatiquement

* Plusieurs « else if » imbriqué dans une boucle for, dans ce cas nous privilégierions plutôt un « switch case ». (RainbowFilter, Filters.cs)
* Les paramètres utilisés dans « SetPixel » sont calculés directement dans la parenthèse de la méthode. Dans ce cas, il est préférable d’instancier les variables nécessaires avant l’exécution de la méthode, ça donnerait quelque chose comme   
  temp.SetPixel(i, h, x, y, z)  
  Ce code est plus lisible et il permet de plus facilement modifier la valeur d’un pixel. (RainbowFilter, Filters.cs)

Une image contenant capture d’écran

Description générée automatiquement

* Plusieurs « else if » imbriqué dans un « if ». Dans ce cas, nous privilégierions un « switch case ».



* Le « == true » est de trop. Le code fonctionne si on met simplement « if(preview) » puisque la variable « preview » est un boolean.

*3.3 Commentaires*

Globalement, le code a très peu de commentaires. Voici les commentaires trouvés :

|  |  |  |
| --- | --- | --- |
| **Classe + ligne** | **Commentaire** | **Commentaires** |
| Filters.cs  Ligne 11 |  | Commentaire redondant, puisque le nom de la méthode est censé être suffisamment explicite, comme dans ce cas. |
| Filters.cs  Ligne 31 |  | Commentaire redondant, puisque le nom de la méthode est censé être suffisamment explicite, comme dans ce cas. |
| MainForm.cs  Ligne 1 |  | Le code, nous le savons, a été écrit par Yann Clavien et Alexandre Berclaz. Ce commentaire ne fait aucun sens et on ne sait pas à quelle section de code il fait référence. |
| MainForm.cs  Ligne 198 |  | Ce commentaire fait du sens. |
| MainForm.cs  Ligne 245 |  | Commentaire écrit en anglais / français. |
| Program.cs  Ligne 10 |  | Les rubriques « <summary > » ne sont pas utiles dans ce contexte, puisqu’il existe qu’une seule ligne de code. |

*3.4 Sections de code qui mériteraient des commentaires*

|  |  |
| --- | --- |
| **Section de code** | **Commentaire proposé** |
|  | If there is no image preview and no selected filter, do nothing and exist the method |
| Une image contenant capture d’écran  Description générée automatiquement | If there is no image preview, initialize it |
|  | If there is a preview image, change it to show the new filtered image |
| Une image contenant capture d’écran  Description générée automatiquement | Activate the combo box only if a filter is active |

**4. Couverture et efficacité des tests unitaires**

**Une image contenant capture d’écran

Description générée automatiquement**

Certaines classes ne sont pas testées (Matrix.cs, Program.cs, MainForm.cs).

Nous allons nous concentrer sur les 2 fichiers de tests créés : ExtBitmap et Filters.

*4.1 TestExtBitmap.cs*

Une image contenant capture d’écran

Description générée automatiquement

Nous constatons que la méthode qui teste la détection de bords tente de comparer le hashcode d’une image Bitmap à une valeur fixe. Cela ne peut pas fonctionner, car la valeur du hashcode change à chaque compilation.

De plus, nous constatons que pas toutes les méthodes de ExtBitmap sont testées.

Nous ajouterons également que le lien de l’image Bitmap n’est pas un lien relatif, ainsi il ne peut pas fonctionner sur nos postes respectifs.

*4.2 TestFilters.cs*



Même constat que pour TestExtBitmap.cs, les méthodes testent une valeur fixe de hash qui change à chaque compilation et le lien n’est pas relatif.

*4.3 Couverture du code (sans corriger le lien)*

Une image contenant capture d’écran

Description générée automatiquement

Puisque tous les tests se basent sur un lien qui n’est pas relatif, la couverture est très mauvaise.

*4.4 Couverture du code (en corrigeant le lien)*

Une image contenant capture d’écran

Description générée automatiquement

En corrigeant le lien, la couverture est meilleure, cependant les tests ne passent pas car le hashcode change.

**5. Tests unitaires proposés**

dki