ENSG Géomatique

ÉCOLE NATIONALE DES SCIENCES GÉOGRAPHIQUES

Ecole Nationale des Sciences Géographiques

UNIVERSITÉ PARIS-EST

Université Paris-Est

-PROJET INFORMATIQUE -
Rapport développeur

Mastère spécialisé ® Photogrammétrie, Positionnement, Mesure de Déformations

Développement d'une interface graphique pour améliorer des classifications de manière interactive

ENSG Géomatique

ÉCOLE NATIONALE DES SCIENCES GÉOGRAPHIQUES

CHUPIN Clémence

Janvier-Février 2018

☒ Non confidentiel ☐ Confidentiel IGN ☐ Confidentiel Industrie ☐ Jusqu'au ...

Ecole Nationale des Sciences Géographiques
6-8 Avenue Blaise Pascal - Cité Descartes - 77420 Champs-sur-Marne
Téléphone 01 64 15 31 00 Télécopie 01 64 15 31 07

Table des matières

Introduction

Dans le cadre des projets informatiques des étudiants du Mastère Spécialisé PPMD de l'ENSG, le laboratoire MATIS de l'IGN a proposé un sujet se rattachant à la problématique des modèles 3D urbains. Ce sujet s'inscrit dans le cadre de la thèse de M.ENNAFII Oussama, intitulée "Evaluation and selection of 3D city modelling techniques". Pour détecter et caractériser les erreurs des maquettes 3D urbaines, une méthode d'auto-qualification par classification supervisée a été mise en place. Pour permettre la transition vers une classification active,le projet propose de mettre en place une interface graphique pour interagir avec un utilisateur.

Ce rapport développeur a pour objectif de décrire dans le détail le code et les données du programme. Il est complété par le rapport de programmation et le rapport utilisateur.

Chapitre 1 Structure générale

1.1 Configuration requise

Le langage de programmation utilisé pour le code est le Python. La partie interface graphique utilise la bibliothèque Qt, et c'est la version 5 de PyQt qui a été utilisée.

D'autres packages de Python sont nécessaires au bon fonctionnement du programme :

- gdal, permettant la gestion données géospatiales vectorielles ou raster ;
- pyshp, permettant l'extraction et la manipulation de Shapefile ;
- qimage2ndarray, une extension permettant la conversion entre les QImages et numpy.ndarray.

Le programme est multi-plateforme : il peut donc être ouvert sous Linux, Windows ou MAc OS.

1.2 Données et formalisme

Le programme considère plusieurs données en entrée, sélectionnées par l'utilisateur via une première interface :

Données	Type de donnés	Correspondance dans le code
Classes d'erreurs possibles	Fichier .CSV	Variable de type dictionnaire
Résultats de l'auto-qualification	Fichier .CSV	Liste de tuples
Géométrie des entités	Dossier de .SHP	Attribut *geometry* de la classe Bâtiment
Orthophoto	Fichier .TIFF	Tuple (résolution + référence) + Matrice

Figure 1.1 – Données en entrée

De même, le formalisme des données en sortie est imposé :

Données	Type de donnés	Correspondance dans le code
Résultats de l'interaction	Fichier .CSV	Liste de tuples

Figure 1.2 – Données en sortie

Chapitre 2
Détail du code

Pour faciliter l'écriture et la compréhension du code, on utilise la méthode Modèle/Vue/Contrôleur. Dans cette section, on détaillera donc les différentes fonctions implémentées en suivant ce modèle.

2.1 Partie Vue

La partie Vue contient le code relatif à l'interface graphique. Après la réalisation des trois interfaces graphiques avec QtDesigner, l'utilitaire PyUic5 a permis de transcrire le code C++ en langage Python.

Trois fichiers ont alors été créés et définissent les trois classes :

→ La classe *Ui_InterfacePrincipale* du fichier *classificationActive.py* comporte le code de l'interface principale. Elle définit les caractéristiques des boutons, des labels de texte et de la fenêtre graphique dans une première méthode *setupui* : ce sont les attributs de l'interface. Une méthode *retranslateUi* permet de mettre à jour le texte des attributs.

→ La classe *Ui_ChargementFichiers* du fichier *chargementFichiers.py* comporte le code de l'interface de chargement. Elle dispose elle aussi des deux méthodes *setupui* et *retranslateUi*. Pour permettre l'affichage des différentes stratégies dans le menu déroulant, on utilise la variable globale STRATEGIES définie dans le fichier *strategy.py*. Ainsi, ce menu sera mis à jours automatiquement en cas d'implémentation d'une nouvelle stratégie.

→ La classe *Ui_ChoixClasse* du fichier *choixClasse.py* comporte le code de l'interface de sélection d'une nouvelle classe. Elle dispose des méthodes *setupui* et *retranslateUi*.

En cas de modification de l'interface, ces fichiers sont réécrits, mais cela n'impactera pas le code du programme principal. Une seule modification doit être apportée à ces codes générés automatiquement : il faut modifier le chemin de l'image du point d'interrogation. En effet, ce chemin est contenu dans un fichier ressource en C++, qui n'est pas traduisible simplement en Python avec PyUic5.

2.2 Partie Modèle

La partie Modèle contient l'organisation des données. Ainsi, trois nouveaux fichiers viennent définir trois classes : la classe *Building*, la classe *Background* et la classe *Strategy*.

2.2.1 La classe *Building*

La classe *Building* est codée dans le fichier *building.py*. Un objet de type *Building* est défini par 4 attributs : identity (identifiant de l'objet), geometry, classe et probability.

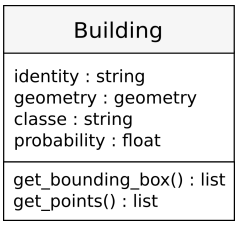

FIGURE 2.1 – Classe *Building*

Deux méthodes lui sont associées :

→ *get_points* retourne la liste de tous les sommets de chaque géométrie de l'entité. Ainsi, si une entité est composée de 3 triangles, la liste de sortie contiendra 9 points. Chaque point est codé par un couple de coordonnées (x,y).

→ *get_bounding_box* parcourt l'ensemble des coordonnées x et y d'une liste et retourne les valeurs maximales et minimales de ces deux paramètres. Cela permet de définir la fenêtre d'emprise d'une entité.

Le fichier *building.py* contient aussi deux fonctions :

→ *read_building* permet de construire des objets de type *Building*. Elle prend comme paramètres d'entrée le dossier contenant les géométries (directory), l'identifiant, la classe et la probabilité d'une entité (building_id, classe, probability). Elle retourne un objet de type *Building*. La géométrie est construite en appelant la fonction *get_geometry*.

→ *get_geometry* retourne les sommets des polygones contenus dans un fichier .SHP. Elle prend comme paramètres d'entrée le chemin du fichier .SHP, et renvoie une liste de tuples. Pour lire les formes contenues dans la géométrie, on utilise le module *shapefile*.

2.2.2 La classe *Background*

La classe *Background* est codée dans le fichier *background.py*. Un objet de type *Background* est défini par 3 attributs : reference_point (coordonnées origines de l'image), pixel_size (taille des pixels de l'image) et image (matrice d'image).

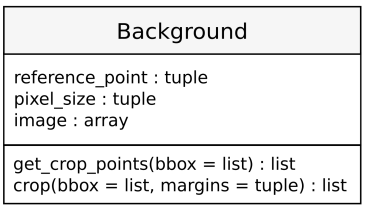

FIGURE 2.2 – Classe *Background*

Deux méthodes lui sont associées :

→ *get_crop_points* renvoie les extrémités de la fenêtre de rognage de l'orthoimage. Avec cette fonction, les extrémités de la fenêtre de rognage, exprimées dans un système métrique (= paramètre bbox), sont transformées en coordonnées pixels.

→ *crop* est la fonction permettant de rogner une image. Elle prend en paramètres les extrémités de la fenêtre de rognage et les valeurs des marges. Pour chaque bande de l'orthoimage, on sélectionne la sous-matrice correspondant aux limites fixées.

Le fichier *building.py* contient aussi une fonction :

→ *read_background* permet de construire des objets de type *Background*. Elle prend comme paramètre d'entrée le chemin de l'orthoimage (filename). La lecture de l'image utilise le package *gdal* et ses méthodes associées. La fonction retourne un objet *Background*, dont l'image associée est composée de 3 bandes.

2.2.3 La classe *Strategy*

Le fichier *strategy.py* définit les classes correspondant aux stratégies de sélection des entités. A ce jour, seules deux méthodes ont été implémentées.

Le fichier définit donc trois classes :

→ La classe *Strategy* est la classe mère. Elle n'a pas d'attributs, mais elle définit une méthode *filtre* qui prend une liste en paramètre mais ne retourne rien.

→ La classe *Naive* est une classe fille, qui hérite des propriétés de la classe *Strategy*. Elle n'a pas d'attributs, et sa méthode *filtre* retourne la liste donnée en paramètre sans aucune modification.

→ La classe *Random* est une classe fille, qui hérite des propriétés de la classe *Strategy*. Elle prend comme attribut le nombre d'entité à retourner (*selection_number*). Sa méthode *filtre* retourne un nombre fixé d'entités, aléatoirement sélectionnées dans la liste donnée en paramètre.

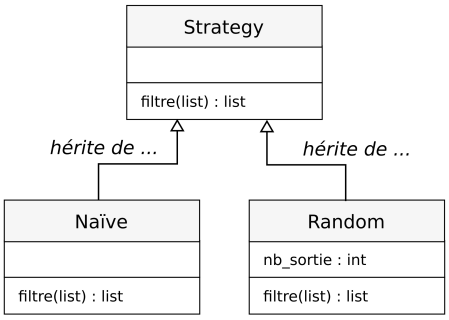

FIGURE 2.3 – Classe *Background*

Ce fichier définit également une variable globale *STRATEGIES*. Sous forme de dictionnaire, cette variable regroupe l'ensemble des classes filles de *Strategy*. Ainsi, les clés du dictionnaire correspondent aux différentes classes stratégies. Les valeurs associées aux clés renseignent les attributs de la classe et les paramètres de la fonction filtre. Cette variable est créée en utilisant la réflexion, c'est à dire la capacité de Python à examiner ses propres structures internes.

2.3 Partie Contrôleur

La partie Contrôleur contient les définitions informatiques des actions effectuées par l'utilisateur dans l'interface. Ces actions viennent modifier les données contenues dans le modèle, ce qui entraine un changement dans la vue.

La partie Contrôleur est codée dans le fichier *interface.py*. Les trois classes définies dans ce document héritent des classes de la partie Vue. Ainsi, il est possible de définir les connexions entre les interfaces et les fonctions.

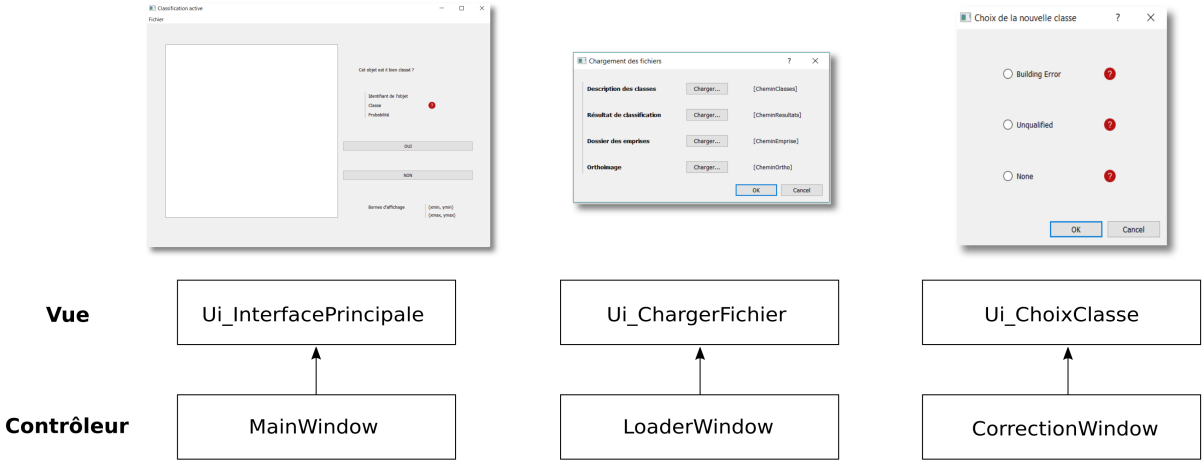

FIGURE 2.4 – Organisation des interfaces et héritages associés

2.3.1 Classe *LoaderWindow*

La classe *LoaderWindow* hérite de la classe *Ui_ChargerFichier*. Elle définit et gère toutes les connexions de l'interface de chargement. Ses principaux attributs enregistrent les chemins d'accès des fichiers et les paramètres de traitement définis par l'utilisateur.

On distingue donc 7 méthodes pour cette classe :

→ *select_classes* est appelée lorsque l'utilisateur clic sur le premier bouton *Charger*. Un objet QFileDialog est appelé et permet à l'utilisateur de sélectionner le fichier des classes. Si le chemin choisi est correct, il s'affiche dans le QLabel associé.

→ *select_results* est appelée lorsque l'utilisateur clic sur le deuxième bouton *Charger*. La fonction demande à l'utilisateur le chemin du fichier des résultats et l'affiche dans un label.

→ *select_background* est appelée lorsque l'utilisateur clic sur le dernier bouton *Charger*. La fonction demande à l'utilisateur le chemin de l'orthoimage et l'affiche dans un label.

→ *select_building_dir* est appelée lorsque l'utilisateur clic sur le troisième bouton *Charger*. La fonction demande à l'utilisateur le chemin du dossier des emprises et l'affiche dans un label.

→ *current_strategy* retourne la méthode de filtrage à utiliser en fonction du choix du menu déroulant (= comboBox). Cette fonction prend en paramètre d'entrée une liste d'entités, et retourne la liste filtrée.

→ *param_strategy* met à jour l'affichage selon la valeur sélectionnée dans le menu déroulant (= comboBox). Ainsi, selon la stratégie sélectionnée, des parties de l'interface seront grisées ou dégrisées.

→ *get_margins* retourne un tuple contenant les valeurs des marges rentrées par l'utilisateur.

2.3.2 Classe *MainWindow*

La classe *MainWindow* hérite de la classe *Ui_InterfacePrincipale*. Elle définit et gère toutes les connexions de l'interface principale. Elle possède de nombreux attributs et des méthodes qui utilisent et modifient ces attributs.

On distingue donc 7 méthodes pour cette classe :

→ *show_loading_window* est une méthode qui se lance lorsque l'utilisateur clic sur l'onglet *Fichier > Charger ....* La fenêtre de chargement des fichiers est appelée et l'utilisateur peut renseigner les chemins d'accès et les paramètres de traitement. Si tous les chemins sont renseignés, la fonction crée les variables nécessaires au traitement : le dictionnaire des classes (*classes*) et la liste des résultats de l'auto-qualification (*results*). Les résultats sont filtrés avec la fonction *current_strategy*, et une liste d'objets *Building* est créée (variable *input_building*). Après avoir chargé l'orthoimage (méthode *read_background* de *Background*), la phase d'affichage et d'interaction est lancée en appelant la méthode *next*.

→ *next* est une méthode qui va tester la longueur de la liste *input_building*. S'il reste des éléments, elle enregistre et enlève le dernier élément de la liste (méthode *pop*) et lance la fonction d'affichage. S'il n'y a plus d'éléments à présenter, la fonction *save* est lancée puis le programme est fermé.

→ *show_building* est une méthode permettant d'afficher un objet *Building*. Elle initie l'affichage graphique en créant une *scene*, puis y ajoute l'orthoimage rognée (méthode *addPixmap*) et la géométrie (méthode *addPolygon*). La mise à l'échelle de la scène est réalisée avec la méthode *fitInView*. Les caractéristiques de l'entité sont affichées dans les différents labels (méthode *setText*), ainsi que les bornes d'affichage. La méthode *setToolTip* permet d'afficher des infobulles.

→ *validate* permet de valider l'entité en cours de visualisation. Sa probabilité est modifiée et l'objet *Building* est ajouté à la liste des résultats de sortie *output_building*.

→ *correct* permet de corriger l'entité en cours de visualisation. La fenêtre de choix d'une nouvelle classe est appelée, et sa nouvelle valeur est enregistrée. La probabilité est modifiée et l'objet *Building* est ajouté à la liste des résultats de sortie *output_building*.

→ *show_correction_window* est une méthode qui se lance lorsque l'utilisateur invalide le choix de la classe. La fenêtre de sélection d'une nouvelle classe est appelée et l'utilisateur peut sélectionner la nouvelle classe, qui est sauvegardée dans une variable *new_label*.

→ *save* permet à l'utilisateur de sélectionner le chemin d'enregistrement de son fichier. Si ce chemin est correct, un fichier .CSV est créé avec les données contenues dans la liste *output_building*. Si le chemin n'est pas valide, un message d'erreur est affiché.

La fonction *show_main_window* est également définie dans le fichier *interface.py*. Elle contient le code pour lancer l'affichage de l'interface principale. C'est cette fonction qui est appelée au lancement du programme.

2.3.3 Classe *CorrectionWindow*

La classe *CorrectionWindow* hérite de la classe *Ui_ChoixClasse*. Elle définit et gère toutes les connexions de l'interface de sélection d'une nouvelle classe.

On distingue deux méthodes pour cette classe :

→ *check* met à jour la valeur des labels selon la classe de l'entité en cours de visualisation. En effet, les nouveaux choix proposés à l'utilisateur sont des classes différentes de celle affichée dans l'interface principale.

→ *new_choice* teste le bouton coché par l'utilisateur, et retourne la valeur du label associé. Cela permet d'enregistrer la nouvelle classe choisie par l'utilisateur.

Chapitre 3
Bilan

L'ensemble des classes et des méthodes définies dans les sections précédentes sont résumées dans le schéma ci-après. Les informations correspondant aux fichiers sources et aux héritages sont également indiqués.

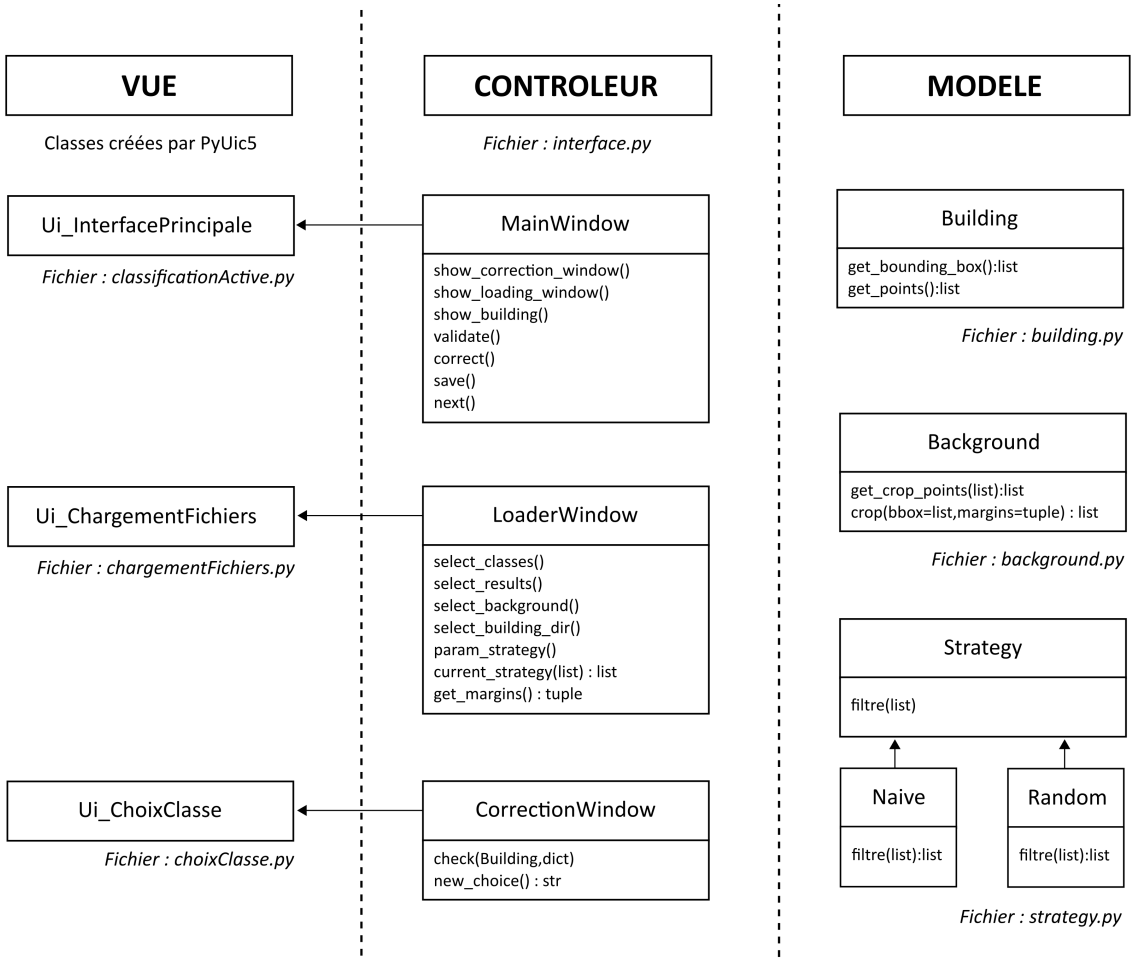

Figure 3.1 – Classes et méthodes associées

Table des figures