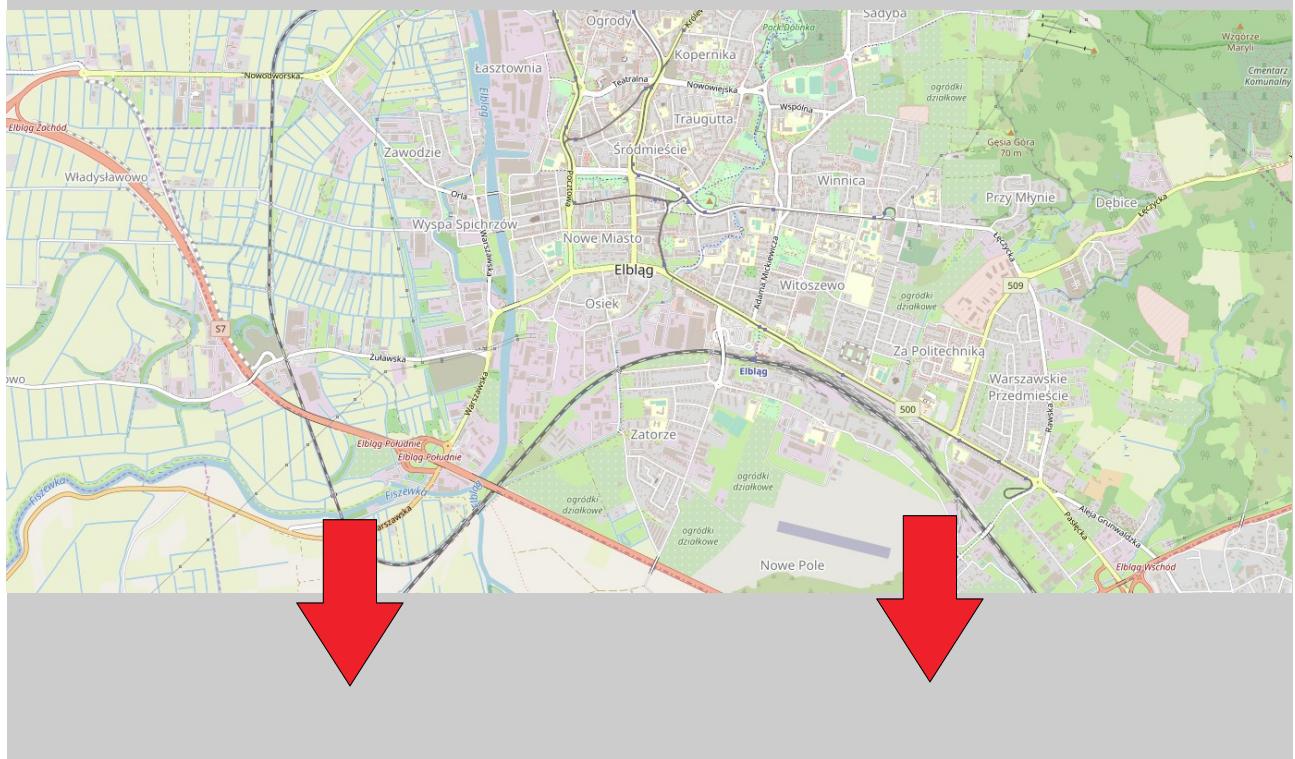


Rapport sur la Reconstruction de modèle 3D à partir d'OpenStreetMap



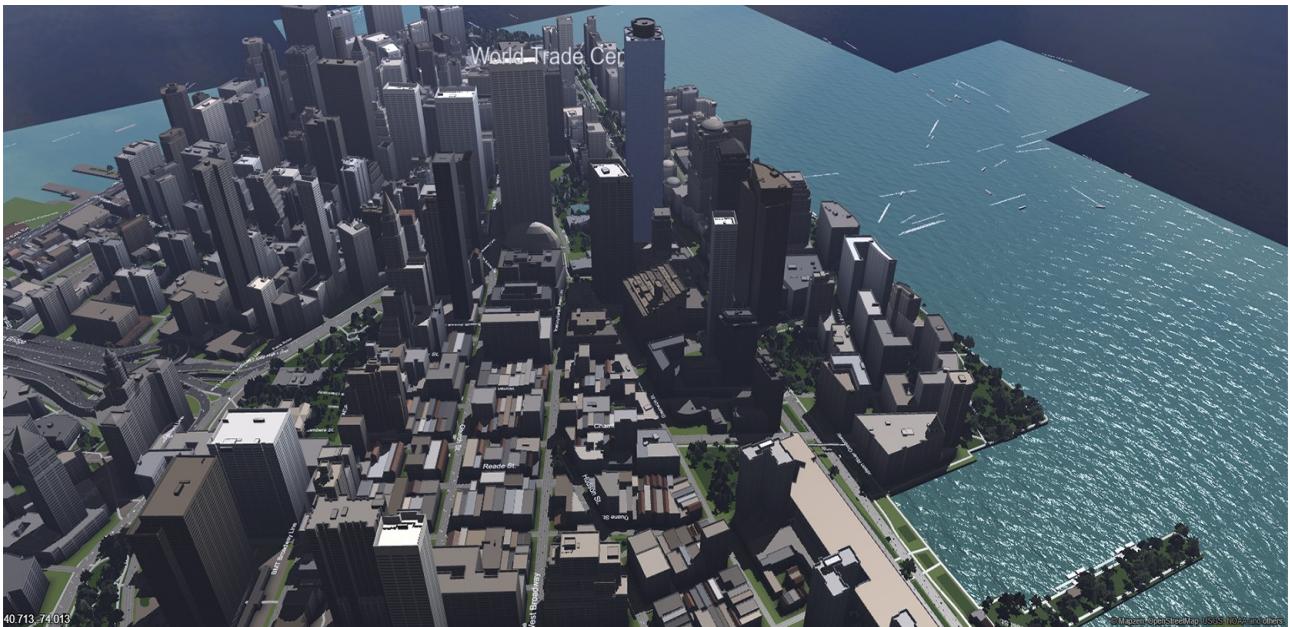


Table des matières

| | |
|---|----|
| I/ Introduction..... | 3 |
| II/ Importation des données OSM sur les bâtiments..... | 4 |
| 1/ Présentation de l'interface « OsmSharp »..... | 4 |
| 2/ Description du modèle de données des éléments OSM simples et complets..... | 6 |
| 3/ Utilisation de la librairie « OsmSharp » pour l'importation..... | 7 |
| a) Création d'un objet de type PBFOsmFile dans la méthode Main..... | 7 |
| b) Filtrage des éléments OSM contenant la clé de tag « building »..... | 7 |
| c) Création d'un nouvel objet de type PBFOsmFile et écriture des données OSM sur les bâtiments dans un fichier XML..... | 8 |
| d) Filtrage des éléments OSM contenant la clé de tag « height » et concaténation des données sur la hauteur à celles contenant un attribut relatif aux bâtiments..... | 8 |
| e) Ajout des éléments OSM correspondant aux attributs et hauteurs des bâtiments à la liste des bâtiments et suppression des éléments en double dans la liste..... | 9 |
| f) Écriture des sous-éléments du tableau d'éléments OSM obtenu dans des fichiers séparés correspondant aux sous-nœuds, sous-chemins et sous-relations..... | 9 |
| g) Importation des données OSM sur les bâtiments dans l'éditeur de Unity..... | 10 |
| 4/ Index des propriétés et méthodes publiques de la classe « PBFOsmFile »..... | 14 |
| III/ Estimation de la hauteur des bâtiments..... | 18 |
| 1/ Méthode du modèle de prédiction de la hauteur par régression..... | 18 |
| a) Présentation de la régression par arbre de décision..... | 18 |
| b) Un exemple d'arbre de décision pour prédire la hauteur des bâtiments..... | 19 |
| c) Construction de notre modèle de régression par arbre de décision..... | 20 |
| c.i) Création d'un objet de type HeightPredictor dans la méthode Main..... | 20 |
| c.ii) Appel à la méthode « Start » et description détaillée de cette méthode..... | 21 |
| c.iii) Description de l'algorithme d'entraînement du modèle de régression par arbre de décision..... | 21 |
| d) Importation du fichier contenant les valeurs de hauteur prédictes dans l'éditeur de Unity.... | 22 |
| 2/ Méthode de prédiction de la hauteur par détection des contours sur des photographies de bâtiments..... | 24 |
| a) Présentation du filtre de Sobel..... | 24 |

| | |
|---|----|
| b) Détection des contours sur une photographie d'un bâtiment du Liechtenstein..... | 25 |
| b.i) Création d'un objet de type ObjectPicture dans la méthode Main..... | 25 |
| b.ii) Application du filtre de Sobel à la photographie et sauvegarde de l'image filtrée..... | 25 |
| b.iii) Visualisation de la photographie originale et des images filtrées..... | 26 |
| c) Formules utilisées pour le calcul de la hauteur d'un bâtiment à partir d'une photographie.. | 29 |
| d) Explication de la méthode de prédiction de la hauteur à partir de la détection des contours sur une photographie d'un bâtiment du Liechtenstein..... | 31 |
| e) Importation des images filtrées dans l'éditeur de Unity..... | 32 |
| f) Index des propriétés et méthodes publiques de la classe ObjectPicture..... | 33 |
| IV/ Implémentation du modèle 3D CityGML sur Unity..... | 36 |
| 1/ Présentation de l'interface utilisateur de Unity..... | 36 |
| 2/ Actions de l'utilisateur nécessaires à la construction de notre modèle..... | 38 |
| a) La fenêtre d'inspecteur de l'objet Loaders..... | 38 |
| b) Construction de notre modèle sans les prédictions sur les hauteurs des bâtiments..... | 41 |
| c) Construction de notre modèle avec les prédictions sur les hauteurs des bâtiments..... | 43 |
| 3/ Actions de l'utilisateur nécessaires à la modification de notre modèle..... | 43 |
| a) La fenêtre d'inspecteur de l'objet associé aux détails d'un bâtiment..... | 43 |
| b) Exemple de modification de la hauteur d'un bâtiment du Liechtenstein..... | 46 |
| 4/ Diagramme de classes de notre modèle..... | 47 |
| 5/ Responsabilités de chaque classe du diagramme..... | 48 |
| V/ Évaluation et validation de notre modèle | 50 |
| 1/ Configuration de la fenêtre d'inspecteur de l'objet Loaders lors de la construction de notre modèle..... | 50 |
| 2/ Comparaison de notre modèle du Liechtenstein dans son ensemble sur Google Earth..... | 52 |
| 3/ Comparaison des modèles de 8 bâtiments du Liechtenstein sur Google Maps/Earth..... | 54 |
| a) La cathédrale de St-Florin..... | 54 |
| b) Le musée des Beaux-Arts du Liechtenstein..... | 56 |
| c) La maison Rouge de Vaduz..... | 58 |
| d) L'université du Liechtenstein et sa salle à manger..... | 60 |
| e) Le gymnase du Liechtenstein..... | 62 |
| f) La garderie de Kosthaus..... | 64 |
| g) Swarovski AG..... | 66 |
| h) Hoval..... | 68 |
| i) Les fenêtres d'inspecteur des objets associés aux détails de ces 8 bâtiments..... | 70 |
| VI/ Conclusion..... | 72 |

I/ Introduction

Les enjeux de reconstruction de modèle 3D CityGML à partir de données « OpenStreetMap » sont importants. En effet, rappelons que le but d'un modèle 3D CityGML est de fournir un service aux utilisateurs ainsi qu'aux autorités des villes pour s'informer et permettre la planification de projets de travaux urbains. Ce service pourrait ainsi bénéficier à un public nombreux et varié.

Dans notre état de l'art, nous avions d'abord décrit à quoi ressemblait un modèle 3D urbain reconstruit à partir d'OSM : nous avions donc commencé par définir le modèle « CityGML » constitué d'un ensemble d'objets géographiques 3D placés sur un terrain donné dans un environnement urbain, ainsi que leurs éléments 2D associés sur OpenStreetMap (Nœuds, chemins et

relations). Ensuite, nous avions présenté les 5 niveaux de détail standards (abrégés « LoDs ») de CityGML puis les méthodes de reconstruction d'un modèle 3D CityGML à partir d'OSM, avec ou sans les données sur la hauteur des bâtiments.

Dans notre modèle 3D CityGML, j'ai donc choisi d'importer les données géographiques à partir d'OSM puis de représenter les bâtiments reconstruits en « LoD1 » (cependant, j'envisage de reconvertir notre modèle en « LoD2 » dans le futur afin de reconstruire les bâtiments avec plus de détails). J'ai également décidé d'omettre les données d'élévation du terrain (mon modèle initial LoD0 correspond à un simple terrain plat) car cela nécessiterait d'importer encore plus de données, et aussi car cela pourrait fausser la hauteur réelle des bâtiments, suivant la géographie du terrain (qui contient d'ailleurs peu de variations dans les villes).

Au niveau des méthodes de reconstruction utilisées sans les données sur la hauteur des bâtiments, j'ai choisi d'implémenter un modèle de prédiction qui utilise une régression à partir d'attributs géométriques en majorité ainsi que cadastres. J'ai également implémenté une méthode qui essaie de prédire la hauteur des bâtiments à partir d'une photographie de l'une de leurs façades, en utilisant la détection des contours grâce au filtre de Sobel.

Au niveau des technologies utilisées, j'ai choisi d'implémenter notre modèle sur Unity qui était le choix idéal pour la créations de mondes virtuels en 3D, puis de l'évaluer avec Google Earth ou Google Maps. De plus, j'ai utilisé le site *Geofabrik.de* pour le téléchargement des données OSM et la librairie *OsmSharp* pour l'extraction d'éléments OSM à partir des fichiers de type PBF (« Protocolbuffer Binary Format ») téléchargés. J'ai ensuite pu importer les données OSM sur Unity en exportant ces éléments dans des fichiers de type XML (« Extensible Markup Language »).

En partie **II/**, on commencera donc par présenter l'interface *OsmSharp* puis citerons les étapes d'importation des données OSM sur les bâtiments plus en détails.

En partie **III/**, nous présenterons les 2 méthodes choisies pour estimer la hauteur des bâtiments : c'est à dire le modèle de prédiction par régression à partir d'attributs géométriques et cadastres ainsi que la détection des contours sur des photographies de bâtiments grâce au filtre de Sobel.

En partie **IV/**, nous présenterons les détails de l'implémentation de notre modèle 3D CityGML dans l'environnement virtuel 3D Unity.

Enfin, nous présenterons en partie **V/** la démarche utilisée pour l'évaluation et la validation des résultats obtenus dans notre modèle : il s'agira notamment de visualiser les bâtiments reconstruits dans notre modèle et de les comparer à une source de données géographiques 3D fiable tel que Google Earth ou Google Maps.

II/ Importation des données OSM sur les bâtiments

1/ Présentation de l'interface « OsmSharp »

« OsmSharp » est le nom de la librairie C# qui permet d'importer des données OSM à partir de fichiers d'extension « osm.pbf » ou « osm.xml ». Elle permet notamment de convertir un ensemble d'éléments OSM natifs en objets OSM complets (voir section suivante), qui englobent la structure complète des sous-éléments d'un élément OSM parent, ainsi que des éléments OSM en géométries. L'interface « OsmSharp » se présente comme suit :

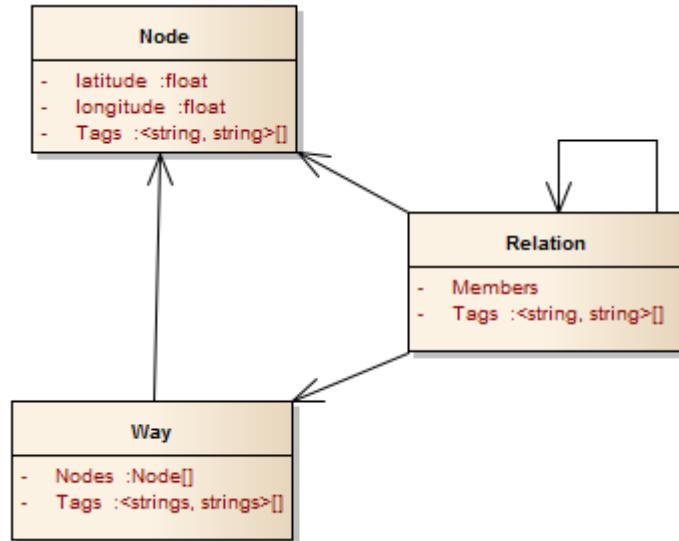


Fig. 1 : Le modèle de données OSM dans l'interface *OsmSharp*, qui consiste en des éléments OSM simples que sont les noeuds, chemins et relations
 (source : <https://docs.itinero.tech/docs/osmsharp/osm.html>)

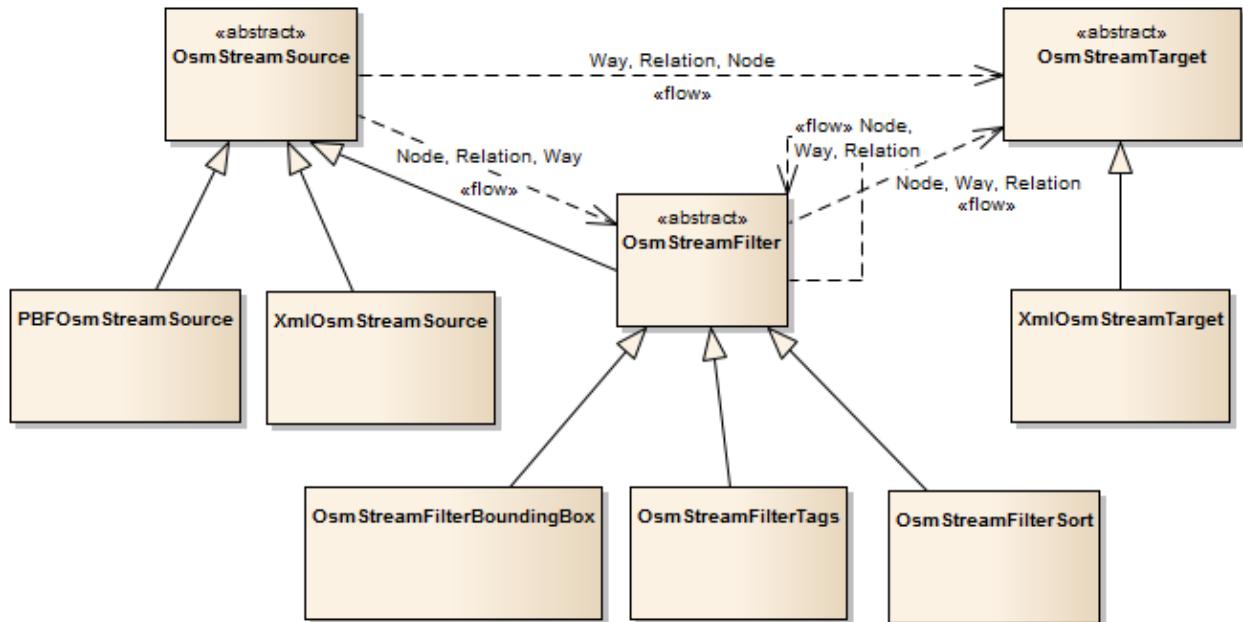


Fig. 2 : Le modèle de lecture (ou « streaming ») des données OSM dans l'interface *OsmSharp*
 (source : <https://docs.itinero.tech/docs/osmsharp/streaming.html>)

Dans le modèle de données OSM, un nœud est un simple objet qui possède une paire de valeurs de latitude et longitude et 0, 1 ou plusieurs « tag(s) » avec une paire « clé-valeur ».

Un chemin est une séquence ordonnée de nœuds qui peut contenir une collection de « tags » en tant que paires « clé-valeur ». Selon l'information géographique des sous-nœuds et les valeurs des « tags », un chemin peut représenter tout type d'objet urbain tel que les routes, espaces verts, lacs, rivières, bâtiments, etc...

Une relation est un objet qui modélise n'importe quelle relation entre des nœuds, chemins et/ou relations. Cet élément OSM est utilisé pour représenter des multi-polygones, c'est à dire des objets urbains complexes tels que les voies de bus, complexes de bâtiments, frontières administratives, etc...

NB : Pour plus d'informations sur les éléments OSM, voir la page de Wiki suivante :
<https://wiki.openstreetmap.org/wiki/Elements>

Les types de base utilisés dans le modèle de lectures des données OSM sont :

- *OsmStreamSource* : Représentation abstraite d'une source de flux de données OSM
- *XmlOsmStreamSource* : Classe dérivée concrète qui permet de lire des fichiers de type OSM-XML
- *PBFStreamSource* : Classe dérivée concrète qui permet de lire des fichiers de type OSM-PBF
- *OsmStreamFilter* : Représentation abstraite d'un filtre de flux de données OSM, utilisée par exemple pour filtrer des « tags » ou sélectionner une certaine région délimitée par un rectangle dans la source
- *OsmStreamTarget* : Représentation abstraite d'une cible pour un flux de données OSM
- *XmlOsmStreamTarget* : Classe dérivée concrète qui permet d'écrire dans des fichiers de type OSM-XML
- *PBFStreamTarget* : Classe dérivée concrète qui permet d'écrire dans des fichiers de type OSM-PBF

2/ Description du modèle de données des éléments OSM simples et complets

Il est nécessaire de bien comprendre le modèle de données des éléments OSM simples et complets que sont les nœuds, chemins et relations avant d'appréhender la suite. Or, l'interface *OsmSharp* ne fournissant pas de diagramme de classes décrivant un tel modèle de données, j'ai moi-même modélisé un diagramme de classes qui en fait la description (voir fig. 3).

Voici une description de chacune des classes modélisées sur notre diagramme en fig.3 :

- *OsmGeoType* : énumération représentant un type d'élément OSM : nœud, chemin ou relation
- *OsmGeo* : représente un élément OSM simple. Chaque élément OSM possède un identifiant unique, un identifiant de groupe, un type (nœud, chemin ou relation), une date de modification, un nom et identifiant d'utilisateur (qui a modifié l'élément en dernier), un numéro de version, un booléen qui indique sa visibilité, et enfin un ensemble d'attributs.
- *Node* : représente un nœud qui contient une paire de valeurs de latitude et longitude. On ne fait pas la distinction entre nœud simple et complet.
- *Way* : représente un chemin simple qui contient un ensemble ordonné d'identifiants de nœuds. Un même identifiant de nœud peut être présent plusieurs fois.
- *Relation* : représente une relation simple qui contient un ensemble de membres.
- *RelationMember* : représente un membre de relation simple qui contient un identifiant unique d'élément OSM, un rôle relatif à la relation et un type d'élément OSM.
- *IIncompleteOsmGeo* : interface implémentée par la classe *CompleteOsmGeo*.
- *CompleteOsmGeo* : représente un élément OSM complet qui a exactement les mêmes attributs que l'élément OSM simple associé, qui peut être obtenu par un appel à la méthode *ToSimple()*.
- *CompleteWay* : représente un chemin complet qui contient un ensemble ordonné de nœuds. Un même nœud peut être présent plusieurs fois.
- *CompleteRelation* : représente une relation complète qui contient un ensemble de membres (sous-éléments OSM complets).
- *CompleteRelationMember* : représente un membre de relation complète qui contient un élément OSM complet et un rôle relatif à la relation.

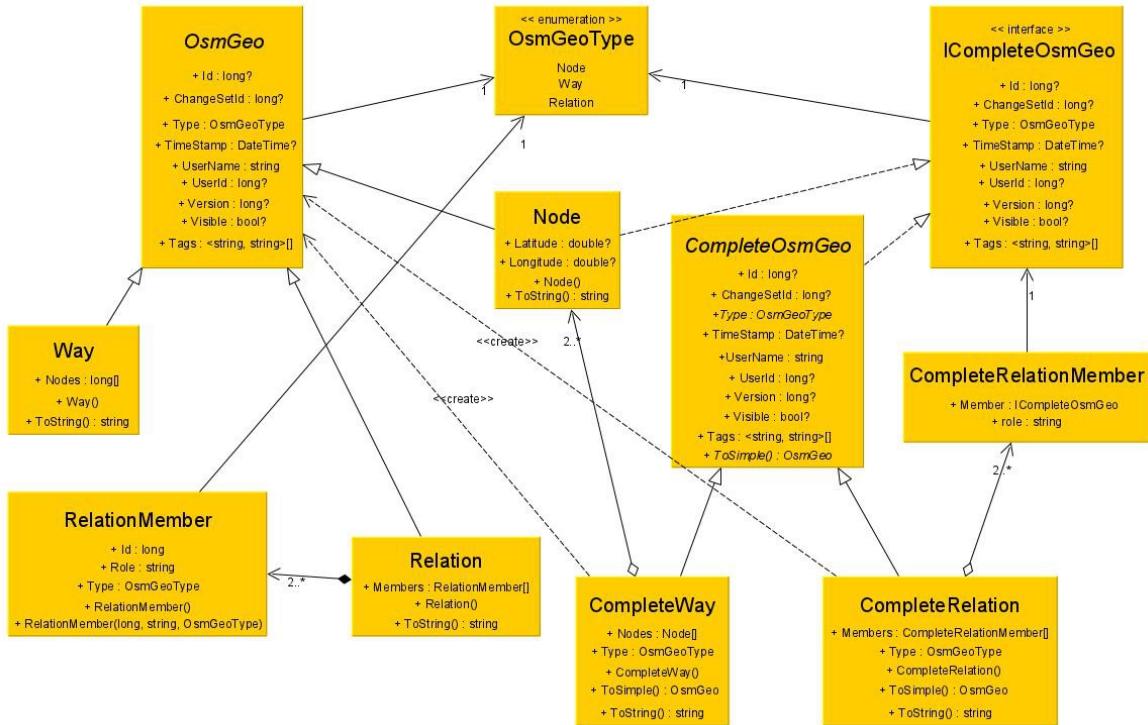


Fig. 3 : Le diagramme de classes du modèle de données de l'interface *OsmSharp* correspondant aux éléments OSM simples et complets

3/ Utilisation de la librairie « OsmSharp » pour l'importation

J'ai d'abord créé un projet séparé de l'implémentation sur Unity (j'en ai été contraint à cause de soucis de compatibilité) nommé « *OsmImport* », qui contient seulement 2 fichiers intitulés *Program.cs* et *PBFOsmFile.cs*. Le premier fichier contient la méthode *Main* qui est le point de départ de l'exécution de notre programme pour la lecture des données OSM. Quant au second fichier, il contient la définition de la classe utilitaire *PBFOsmFile* qui contient un ensemble de propriétés et de méthodes utilisant les fonctionnalités de la librairie *OsmSharp*, utiles pour la manipulation de données OSM de tout type.

Voici ci-dessous toutes les étapes nécessaires pour l'importation de données OSM sur les bâtiments sur Unity :

a) Création d'un objet de type *PBFOsmFile* dans la méthode *Main*

Pour ce faire, on fait un appel à la méthode statique *CreateInstance* de la classe utilitaire *PBFOsmFile* dans la méthode *Main*, en fournissant comme paramètre un chemin de fichier de type OSM-PBF (ayant par convention l'extension « *osm.pbf* ») valide. Par exemple, l'instruction suivante :

```
PBFOsmFile osm = PBFOsmFile.CreateInstance(@"G:\OSMFiles\liechtenstein-latest.osm.pbf");
```

va créer un objet de type *PBFOsmFile* ayant comme nom *osm* à partir du chemin vers le fichier de type OSM-PBF spécifié en paramètre de la méthode statique *CreateInstance*. Dans notre cas, le fichier sur les données du Liechtenstein a été téléchargé à partir du site *Geofabrik.de* sur le lien suivant : <http://download.geofabrik.de/>

b) Filtrage des éléments OSM contenant la clé de tag « building »

Afin de ne garder que les éléments OSM qui correspondent à un bâtiment, il est nécessaire de les filtrer à partir de l'attribut (ou « tag ») ayant comme clé la chaîne de caractères « building ». Pour ce faire, on fait un appel à la méthode *FilterByKeys* à partir de l'instance (ou objet) créée à l'étape précédente, en fournissant les paramètres suivants dans l'ordre :

- un chemin de fichier de type OSM-PBF ou OSM-XML valide, dans lequel écrire les éléments OSM à filtrer
- un tableau de chaînes de caractères qui correspond à l'ensemble des clés de tags à filtrer
- un booléen facultatif (*false*=valeur par défaut) qui indique s'il faut considérer les chaînes de caractères fournies en paramètre comme sous-chaînes de clés de tags à filtrer ou non
- un booléen facultatif qui indique si le fichier de sortie doit être de type OSM-PBF (*false*) ou OSM-XML (*true*=valeur par défaut)

Cette méthode retourne également l'ensemble des éléments filtrés dans un tableau d'objets de type *OsmGeo*.

Par exemple, l'instruction suivante :

```
List<OsmGeo> elemsBuildings = osm.FilterByKeys(@"G:\OSMFiles\liechtenstein-buildings.osm.pbf", new string[] { "building" }, false, false).ToList();
```

va filtrer tous les éléments OSM des données du Liechtenstein correspondant à un bâtiment, puis les écrire dans le fichier de type OSM-PBF dont le chemin est spécifié en premier paramètre. Le tableau retourné par la méthode *FilterByKeys* est ensuite converti en une liste d'objets de type *OsmGeo* nommée *elemsBuildings*.

La prochaine instruction :

```
OsmGeo[] elemsTags = osm.FilterByKeys(@"G:\OSMFiles\liechtenstein-buildings-tags.xml", new string[] { "building:" }, true);
```

va filtrer tous les éléments OSM (sans forcément que ces éléments correspondent eux-mêmes à des bâtiments) des données du Liechtenstein contenant un tag relatif aux bâtiments (notez la présence du caractère ':', qui indique ici que la clé de tag ayant comme préfixe « building » doit contenir des sous-catégories diverses), puis les écrire dans le fichier de type OSM-XML dont le chemin est spécifié en premier paramètre. Le tableau retourné par la méthode *FilterByKeys* a ici pour nom *elemsTags*.

NB :

- Pour plus d'informations sur les attributs d'éléments OSM, voir le lien suivant :

<https://wiki.openstreetmap.org/wiki/FR:Attributs>

- Pour plus d'informations sur la clé de tag « building », voir les liens suivants :

<https://wiki.openstreetmap.org/wiki/FR:Key:building>

<https://taginfo.openstreetmap.org/keys/building#values>

c) Création d'un nouvel objet de type *PBFOsmFile* et écriture des données OSM sur les bâtiments dans un fichier XML

Il est maintenant nécessaire de créer un nouvel objet de type *PBFOsmFile* à partir du fichier sur les données OSM des bâtiments du Liechtenstein créé à l'étape précédente, de façon à exclure toutes les autres données OSM qui ne nous intéressent pas.

On exécute donc l'instruction suivante :

```
PBFOsmFile osm2 = PBFOsmFile.CreateInstance(@"G:\OSMFiles\liechtenstein-buildings.osm.pbf");
```

Ensuite, on fait appel à la méthode *ToXmlFile* à partir de l'instance qui vient tout juste d'être créée. Cette méthode, comme son nom l'indique, va écrire les données OSM du fichier de type OSM-PBF dont le chemin est spécifié plus haut dans un nouveau fichier de type OSM-XML.

L'instruction suivante :

```
osm2.ToXmlFile(@"G:\OSMFiles\liechtenstein-buildings.xml");
```

va donc écrire les données OSM des bâtiments du Liechtenstein dans un fichier XML qui pourra ensuite être directement importé sur Unity.

d) Filtrage des éléments OSM contenant la clé de tag « height » et concaténation des données sur la hauteur à celles contenant un attribut relatif aux bâtiments

Étonnamment, la hauteur ne fait pas partie d'une sous-catégorie de clé de tag ayant comme préfixe « building » : elle définit une clé de tag à part entière. C'est pour cela qu'il faut filtrer les données OSM des bâtiments du Liechtenstein contenant comme clé d'attribut la chaîne de caractères « height ». Ces données OSM contenant l'information sur la hauteur de bâtiments sont cruciales car elles permettront par la suite d'effectuer une prédiction de la hauteur par régression (voir partie III/). En effet, il est très rare de pouvoir extraire l'information sur la hauteur des bâtiments directement depuis OSM.

On exécute ainsi l'instruction suivante :

```
OsmGeo[] elemsHeights = osm2.FilterByKeys(@"G:\OSMFiles\liechtenstein-buildings-heights.xml", new string[] { "height" }, true);
```

qui va filtrer toutes les données OSM des bâtiments du Liechtenstein qui contiennent un attribut sur leur hauteur (et qui sont en nombre de 50), puis écrire ces données dans le fichier de type OSM-XML dont le chemin est spécifié en premier paramètre. Le tableau retourné par la méthode *FilterByKeys* a ici pour nom *elemsHeights*.

La prochaine instruction :

```
PBFOsmFile.Concatenate(@"G:\OSMFiles\liechtenstein-buildings-tags.xml",  
@"G:\OSMFiles\liechtenstein-buildings-heights.xml");
```

va faire un appel à la méthode statique *Concatenate* qui permet de concaténer les données du fichier dont le chemin est spécifié en premier paramètre avec celles du fichier dont le chemin est spécifié en second paramètre. Ici, on va donc concaténer les données OSM sur la hauteur de bâtiments aux autres données contenant un attribut relatif aux bâtiments dans un fichier XML (le type de fichier dans lequel écrire est d'ailleurs spécifié par un paramètre facultatif de type booléen, ici de valeur *true*). On pourra donc facilement retrouver l'information sur la hauteur de 50 bâtiments (ainsi que d'autres informations potentiellement utiles sur les bâtiments) du Liechtenstein à partir de ce fichier XML qui sera importé sur Unity.

e) Ajout des éléments OSM correspondant aux attributs et hauteurs des bâtiments à la liste des bâtiments et suppression des éléments en double dans la liste

Maintenant, on va ajouter les éléments OSM contenant un attribut relatif aux bâtiments ainsi qu'un attribut sur leur hauteur à la liste des éléments correspondant à un bâtiment. Pour ce faire, on

exécute les 2 instructions suivantes :

```
elemsBuildings.AddRange(elemsTags.ToList());  
elemsBuildings.AddRange(elemsHeights.ToList());
```

Comme la liste fusionnée contient très certainement des éléments en double, il est nécessaire d'exécuter l'instruction suivante :

```
OsmGeo[] filtered = PBFosmFile.DropDuplicates(elemsBuildings.ToArray());
```

qui va supprimer les éléments en double dans le tableau passé en paramètre de la méthode *DropDuplicates*. Cette méthode retourne ensuite un nouveau tableau d'objets de type *OsmGeo* ne contenant aucun élément en double, nommé ici *filtered*.

f) Écriture des sous-éléments du tableau d'éléments OSM obtenu dans des fichiers séparés correspondant aux sous-nœuds, sous-chemins et sous-relations

Une difficulté que j'ai rencontrée durant l'importation des données OSM est que les méthodes des classes de la librairie *OsmSharp* ne permettaient pas d'écrire les éléments OSM complets (qui rappelons-le englobent la structure complète des sous-éléments d'un élément OSM parent) dans des fichiers XML ou PBF : la manipulation d'éléments OSM complets était bien sûr possible, mais leur écriture à partir des méthodes prédéfinies de la librairie d'*OsmSharp* était impossible. Il fallait donc que je définisse ma propre méthode pour écrire l'ensemble des sous-éléments des éléments OSM correspondant aux bâtiments du Liechtenstein.

La prochaine instruction :

```
osm.WriteSubElemsTo(@"G:\OSMFiles\liechtenstein-buildings-nodes.xml",  
@"G:\OSMFiles\liechtenstein-buildings-ways.xml", @"G:\OSMFiles\liechtenstein-buildings-  
relations.xml", filtered);
```

va donc écrire tous les sous-éléments du tableau d'éléments OSM passé en dernier paramètre, qui vient d'être obtenu à la sous-section précédente, dans 3 fichiers séparés dont les chemins sont spécifiés dans les 3 premiers paramètres. Plus précisément, cette méthode :

- écrit les sous-nœuds des chemins et relations associé(e)s aux bâtiments du Liechtenstein (tableau *filtered*) dans le fichier dont le chemin est spécifié en premier paramètre
- écrit les sous-chemins des relations associées aux bâtiments du Liechtenstein (tableau *filtered*) dans le fichier dont le chemin est spécifié en second paramètre
- écrit les sous-relations des relations associées aux bâtiments du Liechtenstein (tableau *filtered*) dans le fichier dont le chemin est spécifié en troisième paramètre

Cette méthode accepte également 2 arguments facultatifs de type booléen (de valeur *true* par défaut): le premier indique s'il faut écrire les sous-nœuds des relations contenues dans le tableau d'éléments OSM passé en paramètre ou non, tandis que le 2ème indique s'il faut écrire les sous-éléments dans un fichier de type XML ou PBF. En effet, une relation peut contenir beaucoup de sous-nœuds ce qui pourrait impacter les performances de la méthode, d'où l'importance de ce booléen facultatif.

NB : Si on remplace le dernier paramètre par *null* lors de l'appel à cette méthode, on écrit tous les sous-éléments des éléments OSM du fichier source de type OSM-PBF stocké par l'objet *osm* (nommé *liechtenstein-latest.osm.pbf*) à la place. En fin de compte, on obtiendra exactement le même résultat que précédemment avec ce changement car tous les chemins et toutes les relations de ce fichier source correspondent à des bâtiments : les sous-éléments écrits seront donc les mêmes.

g) Importation des données OSM sur les bâtiments dans l'éditeur de Unity

Une fois les étapes précédentes terminées, on se retrouve donc avec 6 fichiers XML en tout (se trouvant dans le répertoire [G:\OsmFiles](#) dans notre cas) qui ont pour noms :

- *liechtenstein-buildings.xml* : données OSM correspondant aux bâtiments du Liechtenstein
- *liechtenstein-buildings-tags.xml* : données OSM correspondant aux éléments qui contiennent un attribut relatif aux bâtiments du Liechtenstein (ainsi qu'un attribut sur leur hauteur)
- *liechtenstein-buildings-heights.xml* : données OSM correspondant aux bâtiments du Liechtenstein qui contiennent un attribut sur leur hauteur, concaténées aux données précédentes
- *liechtenstein-buildings-nodes.xml* : données OSM correspondant aux sous-nœuds des chemins et relations associé(e)s aux bâtiments du Liechtenstein
- *liechtenstein-buildings-ways.xml* : données OSM correspondant aux sous-chemins des relations associées aux bâtiments du Liechtenstein
- *liechtenstein-buildings-relations.xml* : données OSM correspondant aux sous-relations des relations associées aux bâtiments du Liechtenstein

Tous ces fichiers XML, à l'exception de celui ayant comme nom *liechtenstein-buildings-heights.xml* seront importés sur Unity dans un répertoire bien spécifique appelé *XmlFiles* se trouvant à la racine du projet (nous y reviendrons en partie IV). Pour effectuer l'importation, il suffit de faire un « glisser-déposer » des 5 fichiers XML en question dans ce répertoire (dont le chemin relatif est : *Assets/Resources/XmlFiles*), soit dans l'explorateur de fichiers soit directement dans l'éditeur de Unity comme dans notre exemple.

Cette action de l'utilisateur est illustrée par la figure suivante :

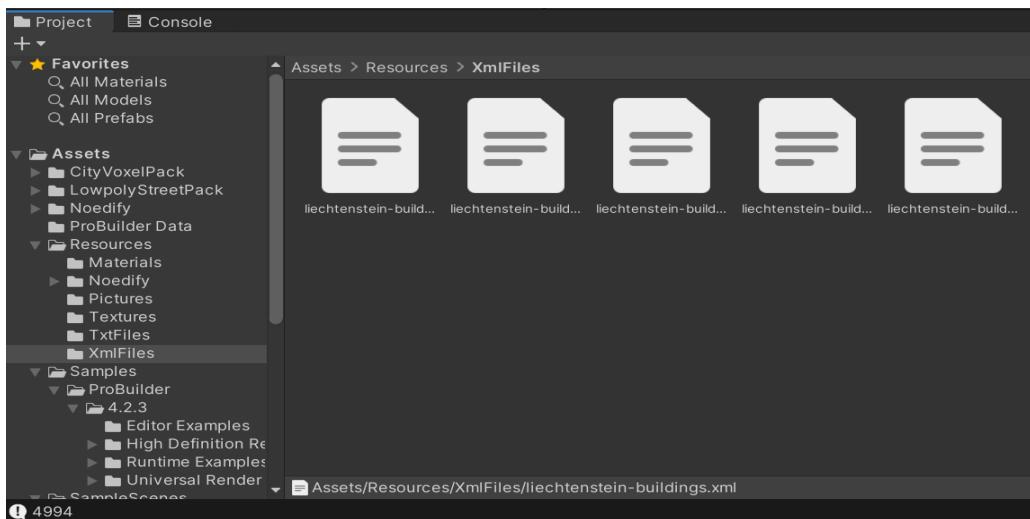


Fig. 4 : Importation des données OSM sur les bâtiments du Liechtenstein à partir de l'éditeur de Unity, par « glisser-déposer » des fichiers XML associés

Fig. 5 : Contenu du fichier de type OSM-XML correspondant aux bâtiments du Liechtenstein

```
<?xml version="1.0" encoding="UTF-8"?><osm version="0.6" generator="OsmSharp"><node id="274855119" lat="47.10901" user="" uid="0" visible="true" version="6018" changeset="0" timestamp="2017-02-17T13:53:20Z" /><node id="274855120" lat="47.1091235" user="" uid="0" visible="true" version="6048" changeset="0" timestamp="2017-02-17T13:53:20Z" /><node id="274855121" lat="47.109355" user="" uid="0" visible="true" version="6074" changeset="0" timestamp="2017-10-01T23:02:54Z" /><node id="277371522" lat="47.1235879" lon="9.5234479" user="" uid="0" visible="true" version="1000" timestamp="2014-08-14T08:20:52Z" /><node id="315636834" lat="47.1395989" lon="9.5225017" user="" uid="0" visible="true" version="3896" changeset="0" timestamp="2017-12-07T15:29:56Z" /><node id="5270199933" lat="47.1362091" lon="9.5219461" user="" uid="0" visible="true" version="19674" changeset="0" timestamp="2017-06-17T22:39:38Z" /><node id="315636854" lat="47.1498066" lon="9.5161459" user="" uid="0" visible="true" version="19959" changeset="0" timestamp="2014-08-14T08:20:52Z" /><node id="7526939" lat="47.1498066" lon="9.5161459" user="" uid="0" visible="true" version="5783" changeset="0" timestamp="2017-08-11T17:26:21Z" /><node id="3015986372" lat="47.1390706" lon="9.5227314" user="" uid="0" visible="true" version="407" changeset="0" timestamp="2013-05-11T15:10:40Z" /><node id="326090068" lat="47.1390398" lon="9.5225938" user="" uid="0" visible="true" version="24331" changeset="0" timestamp="2013-05-11T15:10:40Z" /><node id="341533643" lat="47.2079738" lon="9.5340997" user="" uid="0" visible="true" version="24349" changeset="0" timestamp="2013-05-11T15:10:41Z" /><node id="341533733" lat="47.2072093" lon="9.5322921" user="" uid="0" visible="true" version="4819" changeset="0" timestamp="2013-05-11T15:16:14Z" /><node id="2299755017" lat="47.2066776" lon="9.5329204" user="" uid="0" visible="true" version="4785" changeset="0" timestamp="2013-05-12T11:14:48Z" /><node id="341534760" lat="47.2064899" lon="9.5359144" user="" uid="0" visible="true" version="24442" changeset="0" timestamp="2013-05-12T11:14:48Z" /><node id="341534769" lat="47.2063304" lon="9.5355604" user="" uid="0" visible="true" version="6517" changeset="0" timestamp="2019-06-19T00:33:06Z" /><node id="23008575241" lat="9.5351544" lon="47.205966" user="" uid="0" visible="true" version="24484" changeset="0" timestamp="2009-02-06T02:09:12Z" /><node id="341535233" lat="47.205966" lon="9.5346152" user="" uid="0" visible="true" version="24498" changeset="0" timestamp="2013-05-12T11:14:46Z" /><node id="2299857660" lat="47.2075578" lon="9.525076" user="" uid="0" visible="true" version="24502" />
```

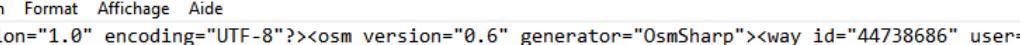
Fig. 6 : Contenu du fichier de type OSM-XML correspondant aux sous-nœuds des chemins et relations associé(e)s aux bâtiments du Liechtenstein



Fichier Edition Format Affichage Aide

```
<?xml version="1.0" encoding="UTF-8"?><osm version="0.6" generator="OsmSharp"><way id="83758674" user="" uid="0" /><nd ref="975488081" /><nd ref="3015986386" /><nd ref="4830790903" /><nd ref="5771534102" /><nd ref="483079675488112" /><nd ref="975488104" /></way><way id="134222984" user="" uid="0" visible="true" version="1" changeset="1" /><nd ref="2775885109" /><nd ref="2775885125" /><nd ref="2775885124" /><nd ref="2775885135" /><nd ref="2775885127" v="1" /><tag k="building" v="roof" /></way><way id="300089872" user="" uid="0" visible="true" version="2" changeset="3" /><nd ref="3382813314" /></way><way id="331268812" user="" uid="0" visible="true" version="1" changeset="4" /><nd ref="3519328160" /><nd ref="1782493241" /><nd ref="1782493239" /><nd ref="1782493240" /><nd ref="3519327982" /><nd ref="30177995" /><nd ref="3530177997" /><nd ref="3530178007" /><nd ref="3530178005" /><nd ref="3530178024" /><nd ref="3530178025" /><nd ref="3937608636" /><nd ref="3937612173" /><nd ref="3937611714" /><nd ref="3937611712" /><nd ref="3937611719" /><nd ref="3952799258" /><nd ref="3952799203" /><nd ref="3952799215" /><nd ref="3952799241" /><nd ref="3952799242" /><nd ref="3955333438" /><nd ref="4355333440" /><nd ref="4355333439" /><nd ref="4355333457" /><nd ref="4355333458" /><nd ref="4355333459" /><nd ref="3024224289" /><nd ref="3024224279" /><nd ref="5029188366" /><nd ref="5029188365" /><nd ref="3024224278" /><nd ref="5679404134" /><nd ref="5679404135" /></way><way id="122542201" user="" uid="0" visible="true" version="1" changeset="1" />
```

Fig. 7 : Contenu du fichier de type OSM-XML correspondant aux sous-chemins des relations associées aux bâtiments du Liechtenstein



The screenshot shows a Microsoft Notepad window with the title "liechtenstein-buildings-heights.xml - Bloc-notes". The content of the file is an XML document describing building features for the city of Triesen in Liechtenstein. The XML includes tags for address, roof shape, height, and building type, along with a detailed description of the Palace and official residence of the Prince of Liechtenstein.

```
<?xml version="1.0" encoding="UTF-8"?><osm version="0.6" generator="OsmSharp"><way id="44738686" user="" uid="0' f="569754862" /><nd ref="3024734891" /><nd ref="569754886" /><nd ref="3024734880" /><nd i="0" visible="true" version="6" changeset="0" timestamp="2018-09-26T09:50:20Z"><nd ref="570096490" /><nd ref="570096491" /><tag k="addr:city" v="Triesen" /><tag k="addr:place" v="Triesen" /><tag k="roof:shape" v="pyramidal" /><tag k="addr:housenumber" v="5" /><tag k="roof:orientation" v="across" /></way><way id="213915462" user="" uid="0" visible="true" version="6" changeset="0" timestamp="2018-09-26T09:50:20Z"><nd ref="570096490" /><nd ref="570096491" /><tag k="building" v="house" /><tag k="building:levels" v="3" /><tag k="height" v="0" /></way><way id="4908772" user="" uid="0" visible="true" version="6" changeset="0" timestamp="2018-09-26T09:50:20Z"><nd ref="280026233" /><nd ref="4830529903" /><nd ref="4830529904" /><nd ref="4830529905" /><nd ref="280026231" /><nd ref="529181599" /><way id="529181599" user="" uid="0" visible="true" version="2" changeset="0" timestamp="2019-02-02T16:52:57Z" /><tag k="description:en" v="Palace and official residence of the Prince of Liechtenstein. Source: Liechtenstein Tourismus" />
```

Fig. 8 : Contenu du fichier de type OSM-XML correspondant aux bâtiments du Liechtenstein contenant un attribut sur leur hauteur

Fig. 9 : Contenu du fichier de type OSM-XML correspondant aux sous-relations des relations associées aux bâtiments du Liechtenstein

```

liechtenstein-buildings-tags.xml - Bloc-notes
Fichier Edition Format Affichage Aide
<?xml version="1.0" encoding="UTF-8"?><osm version="0.6" generator="OsmSharp"><node id="3191525730" lat="47.177" lon="9.617" part="yes" changeset="6" timestamp="2018-09-26T09:50:20Z" user="OsmSharp" uid="1369051614" ref="4830530131" version="1" /><tag k="building:part" v="yes" /><tag k="building:levels" v="3" /></way><way id="25452997" user="" uid="1369051614" changeset="6" timestamp="2018-09-26T09:50:20Z" user="OsmSharp" uid="1369051614" ref="4830530130" version="1" /><nd ref="567527989" /><nd ref="4830530129" /><nd ref="4830530128" /><nd ref="570096490" /><nd ref="570096501" /><nd ref="30241" /><tag k="addr:city" v="Triesen" /><tag k="addr:place" v="Triesen" /><tag k="roof:shape" v="gabled" /><tag k="addr:country" v="LI" /><tag k="building:levels" v="2" /><tag k="addr:housenumber" v="5" /></way><way id="122143096" user="" uid="1369051709" changeset="39" timestamp="2018-10-10T20:01:50Z" /><nd ref="1369051711" /><nd ref="1369051614" /><nd ref="1369051616" /><tag k="addr:street" v="Ober Ställ" /><tag k="building" v="yes" /><tag k="building:levels" v="1" /><tag k="addr:city" v="Triesen" /><tag k="addr:street" v="Oberfeld" /><tag k="building" v="yes" /><tag k="building:levels" v="2" /><tag k="addr:housenumber" v="28" /></way><way id="164674461" user="" uid="0" visible="true" changeset="13" timestamp="2013-07-13" /><nd ref="1768547321" /><nd ref="1768547142" /><nd ref="1768547393" /><nd ref="1792079563" /><nd ref="1792079564" /><tag k="addr:country" v="LI" /><tag k="addr:postcode" v="9495" /><tag k="building:levels" v="2" /><tag k="addr:city" v="Triesen" /><tag k="addr:street" v="Landstrasse" /><tag k="addr:housenumber" v="370" /></way><way id="166816296" user="" uid="0" visible="true" changeset="480817" timestamp="2018-08-17" /><nd ref="1782480818" /><nd ref="1782480832" /><nd ref="1782480831" /><tag k="building" v="house" /><tag k="addr:city" v="Triesen" /><tag k="addr:street" v="Landstrasse" /><tag k="addr:housenumber" v="370" /></way><way id="166816299" user="" uid="0" visible="true" changeset="480817" timestamp="2018-08-17" /><nd ref="1782481232" /><tag k="building" v="apartments" /><tag k="addr:city" v="Triesen" /><tag k="addr:street" v="Oberfeld" /><tag k="building" v="yes" /><tag k="building:levels" v="2" /><tag k="addr:housenumber" v="1" /><tag k="addr:country" v="LI" /><tag k="addr:postcod

```

Fig. 10 : Contenu du fichier de type OSM-XML correspondant aux éléments qui contiennent un attribut relatif aux bâtiments du Liechtenstein (ainsi qu'un attribut sur leur hauteur)

4/ Index des propriétés et méthodes publiques de la classe « PBFOsmFile »

Ici, je présente l'index de toutes les propriétés et méthodes (avec leur signature) de la classe *PBFOsmFile* :

- **string Path** : Propriété de la classe représentant le chemin du fichier source de type OSM-PBF dans lequel lire les données OSM.
- **Boundaries Bounds** : Propriété de la classe représentant les limites géographiques de l'ensemble des nœuds contenus dans le fichier de chemin *Path*. Plus précisément, c'est un objet de type *Boundaries* qui contient les latitudes et longitudes minimale et maximale de la région peuplée par les nœuds du fichier.
- **struct Boundaries** : Structure contenant une latitude et une longitude minimale/maximale.
- **enum GeoAttrs** : Énumération représentant les différents attributs d'un élément OSM : l'identifiant unique, l'identifiant de groupe, le type, le nom et identifiant d'utilisateur, le numéro de version, l'indicateur de visibilité, la date de modification et enfin un attribut ou « tag » (n'importe lequel).
- **static PBFOsmFile CreateInstance(string path)** : Crée une nouvelle instance de la classe *PBFOsmFile* à partir d'un chemin de fichier de type OSM-PBF existant (correspondant à la chaîne de caractères *path*), puis retourne l'instance créée. Un exemple d'utilisation : `PBFOsmFile osm = PBFOsmFile.CreateInstance(@"G:\OSMFiles\antarctica-latest.osm.pbf");`
- **void PrintElems(long nElems, bool verbose = false, bool subVerbose = false)** : Affiche chaque élément OSM du fichier de chemin *Path* dans la console, avec détails ou non selon les valeurs des booléens *verbose* et *subVerbose* (de valeur *false* par défaut). Quelques détails supplémentaires :

1. Dans la console, *nElems* éléments à la fois sont affichés avant chaque pause dont l'utilisateur pourra mettre fin en appuyant sur la touche *Enter*.
 2. Si le premier paramètre *nElems* est négatif ou nul, tous les éléments à la fois seront affichés dans la console.
 3. Si les booléens *verbose* et *subVerbose* ont pour valeurs respectives *true* et *false*, chaque élément du fichier de chemin *Path* sera affiché accompagné des valeurs de chacun de ses attributs (ces attributs correspondant aux constantes de l'énumération *GeoAttrs*).
 4. Si les booléens *verbose* et *subVerbose* ont tous deux *true* comme valeur, chaque élément du fichier de chemin *Path* ainsi que leurs sous-éléments seront affichés accompagnés des valeurs de chacun de leurs attributs respectifs.
- `void PrintNodes(long nElems, bool verbose = false)` : Fonctionne de la même façon que la méthode *PrintElems* mais pour les nœuds contenus dans le fichier seulement. Si le booléen *verbose* a pour valeur *true*, chaque nœud du fichier de chemin *Path* sera affiché accompagné des valeurs de chacun de ses attributs ainsi que de celles de sa latitude et longitude.
 - `void PrintWays(long nElems, bool verbose = false, bool subVerbose = false)` : Fonctionne de la même façon que la méthode *PrintElems* mais pour les chemins contenus dans le fichier de chemin *Path* seulement.
 - `void PrintRelations(long nElems, bool verbose = false, bool subVerbose = false)` : Fonctionne de la même façon que la méthode *PrintElems* mais pour les relations contenues dans le fichier de chemin *Path* seulement.
 - `static void WriteElemsTo(string path, OsmGeo[] elements)` : Écrit les éléments OSM contenus dans le tableau passé en second paramètre dans un fichier dont le chemin est spécifié en premier paramètre. Un exemple d'utilisation :


```
OsmGeo[] elems = new OsmGeo[1];
elems[0] = new Node() {Id = 1, ChangeSetId = 1, Latitude = 0, Longitude = 0, Tags =
new TagsCollection(new Tag("shop", "ski")), TimeStamp = DateTime.Now, UserId = 1424,
UserName = "you", Version = 1, Visible = true};
PBFosmFile.WriteElemsTo(@"G:\OSMFiles\antarctica-extract.osm.pbf", elems);
```
 - `ICompleteOsmGeo[] ExtractRegion(string path, float left, float top, float right,
float bottom, bool completeWays = false, bool isXml = true)` : Extrait une région du fichier de chemin *Path* délimitée par un rectangle dont le coin inférieur gauche a pour longitude *left* et latitude *bottom* et le coin supérieur droit a pour longitude *right* et latitude *top*. Les éléments OSM contenus dans la région extraite sont ensuite écrits dans un fichier OSM-PBF (*isXml=false*) ou OSM-XML (*isXml=true*) dont le chemin est spécifié en premier paramètre. 2 détails importants :
 1. Si le booléen *completeWays* (de valeur *false* par défaut) a pour valeur *false* (resp. *true*), l'ensemble des chemins et relations dont seulement un nœud (resp. tous les nœuds) se trouve(nt) à l'intérieur de la région extraite seront inclus dans le fichier de sortie.
 2. Si la valeur de *left* (resp. *bottom*) est plus grande que celle de *right* (resp. *top*) ou si les valeurs de *left* et *right* (resp. *top* et *bottom*) ne sont pas comprises dans l'intervalle [-180;180] (resp. [-90;90]), alors la méthode devrait retourner une exception.
 3. Exemple d'utilisation :


```
osm.ExtractRegion(@"G:\OSMFiles\antarctica-extract.osm.pbf", -50, -60, 50,
-90, true);
```
 - `ICompleteOsmGeo[] ExtractRegion(string path, IPolygon polygon, bool completeWays =
false, bool isXml = true)` : Fonctionne de la même façon que la méthode précédente mais accepte un objet de type *IPolygon* (correspondant à un polygone simple) à la place pour délimiter les frontières de la région à extraire. Un exemple d'utilisation :

- ```

Coordinate[] coords = new Coordinate[3];
coords[0] = new Coordinate(-50, -90);
coords[1] = new Coordinate(0, -60);
coords[2] = new Coordinate(50, -90);
IPolygon pol = new Polygon(new LinearRing(coords));
osm.ExtractRegion(@"G:\OSMFiles\antarctica-extract.osm.pbf", pol, true);

```
- **OsmGeo[] FilterByAttr(string path, GeoAttrs attr, object value, bool isXml = true) :**  
Filtre tous les éléments du fichier de chemin *Path* vérifiant une certaine condition sur l'un de ses attributs, puis écrit les éléments filtrés dans un fichier de type OSM-PBF (*isXml=false*) ou OSM-XML (*isXml=true*) dont le chemin est spécifié en premier paramètre. Ainsi, cette méthode vérifie pour chaque élément filtré que l'attribut désigné par *attr* (constante de l'énumération *GeoAttrs*) possède la valeur désignée par l'objet *value*. Un exemple d'utilisation :
 

```

OsmGeo[] glaciers = osm.FilterByAttr(@"G:\OSMFiles\antarctica-extract.osm.pbf",
PBFOsmFile.GeoAttrs.Tag, new Tag("natural", "glacier"));

```
  - **OsmGeo[] FilterByTags(string path, Tag[] tags, bool anyTag = false, bool isXml=true) :**  
Filtre tous les éléments du fichier de chemin *Path* contenant au moins l'un des « *tags* » du tableau passé en second paramètre, puis écrit les éléments filtrés dans un fichier de type OSM-PBF (*isXml=false*) ou OSM-XML (*isXml=true*) dont le chemin est spécifié en premier paramètre. Un détail important :
    1. Si le booléen *anyTag* (de valeur *false* par défaut) a pour valeur *true*, tous les éléments contenant au moins un « *tag* » (n'importe lequel) seront écrits dans le fichier de sortie à la place.
    2. Exemple d'utilisation :
 

```

OsmGeo[] glacierAcademy = osm.FilterByTags(@"G:\OSMFiles\antarctica-
extract.osm.pbf", new Tag[] tags = { new Tag("natural", "glacier"), new
Tag("source", "Polish Academy of Science") }, false);

```
  - **OsmGeo[] FilterByKeys(string path, string[] keys, bool containsKey = false, bool isXml = true) :** Filtre tous les éléments du fichier de chemin *Path* contenant au moins l'une des clés de tag du tableau passé en second paramètre, puis écrit les éléments filtrés dans un fichier de type OSM-PBF (*isXml=false*) ou OSM-XML (*isXml=true*) dont le chemin est spécifié en premier paramètre. Nous avions déjà vu plusieurs exemples d'utilisation de cette méthode à la section précédente. Un détail important :
    1. Si le booléen *containsKey* (de valeur *false* par défaut) a pour valeur *true*, les chaînes de caractères des clés fournies en paramètre seront considérées comme sous-chaînes de clés de tags à filtrer à la place.
    2. Exemple d'utilisation :
 

```

OsmGeo[] naturalSource = osm.FilterByKeys(@"G:\OSMFiles\antarctica-
extract.osm.pbf", new string[] keys = { "natural", "source" }, false);

```
  - **void ToXmlFile(string path) :** Convertit le fichier de type OSM-PBF de chemin *Path* en un nouveau fichier de type OSM-XML. Nous avions également déjà vu un exemple d'utilisation de cette méthode à la section précédente. Un exemple d'utilisation :
 

```

osm.ToXmlFile(@"G:\OSMFiles\liechtenstein-buildings.xml");

```
  - **static void Concatenate(string pathSource, string pathOtherSource, bool isXml=true) :** Concatène les données OSM du fichier dont le chemin est spécifié en premier paramètre avec celles du fichier dont le chemin est spécifié en second paramètre. La valeur (*true* ou *false* resp.) du booléen *isXml* spécifie le type (OSM-XML ou OSM-PBF resp.) des fichiers dont il faut concaténer les données. Un exemple d'utilisation :
 

```

PBFOsmFile.Concatenate(@"G:\OSMFiles\liechtenstein-buildings-tags.xml",
@"G:\OSMFiles\liechtenstein-buildings-heights.xml");

```
  - **OsmGeo[] GetAllElems() :** Retourne tous les éléments OSM simples contenus dans le fichier de chemin *Path*.
  - **ICompleteOsmGeo[] GetCompleteElems() :** Retourne tous les éléments OSM complets

contenus dans le fichier de chemin *Path*.

- `Node[] GetNodes()` : Retourne tous les nœuds contenus dans le fichier de chemin *Path*.
- `Way[] GetWays()` : Retourne tous les chemins simples contenus dans le fichier de chemin *Path*.
- `Relation[] GetRelations()` : Retourne toutes les relations simples contenues dans le fichier de chemin *Path*.
- `CompleteWay[] GetCompleteWays()` : Retourne tous les chemins complets contenus dans le fichier de chemin *Path*.
- `CompleteRelation[] GetCompleteRelations()` : Retourne toutes les relations complètes contenues dans le fichier de chemin *Path*.
- `static List<Node> GetSubNodes(CompleteWay way)` : Retourne une liste ordonnée des sous-nœuds du chemin complet passé en paramètre. Un exemple d'utilisation :  
`List<Node> subNodes = PBFOsmFile.GetSubNodes(osm.GetCompleteWays()[0]);`
- `static List<Node> GetSubNodes(CompleteRelation relation)` : Retourne une liste ordonnée des sous-nœuds de la relation complète passée en paramètre. Cette liste contiendra entre autre les sous-ensembles suivants :
  1. tous les sous-nœuds directs de la relation
  2. tous les sous-nœuds des sous-chemins de la relation
  3. tous les sous-nœuds directs des sous-relations de la relation
  4. tous les sous-nœuds des sous-chemins des sous-relations de la relation
  5. Exemple d'utilisation :  
`List<Node> subNodes = PBFOsmFile.GetSubNodes(osm.GetCompleteRelations()[0]);`
- `static List<Way> GetSubWays(CompleteRelation relation)` : Retourne une liste ordonnée des sous-chemins de la relation complète passée en paramètre. Cette liste contiendra entre autre tous les sous-chemins directs de la relation et tous les sous-chemins des sous-relations de la relation.
- `static List<Relation> GetSubRelations(CompleteRelation relation)` : Retourne une liste ordonnée des sous-relations de la relation complète passée en paramètre. Cette liste contiendra donc l'ensemble des sous-relations formé par des appels récursifs à la méthode *GetSubRelations* à l'intérieur de cette propre méthode.
- `static OsmGeo[] DropDuplicates(OsmGeo[] elems)` : Supprime tous les éléments OSM en double dans le tableau passé en paramètre puis retourne un nouveau tableau qui contient les éléments OSM sans leur double. Un exemple d'utilisation :  
`OsmGeo[] filtered = PBFOsmFile.DropDuplicates(elemsBuildings.ToArray());`
- `OsmGeo[] RejectSubDuplicates(string path, OsmGeo[] filteredElems, bool isXml = true)` : Rejette tous les sous-éléments des éléments OSM contenus dans le tableau passé en second paramètre se trouvant parmi les éléments du fichier de chemin *Path* ; puis écrit les éléments restants dans un fichier de type OSM-PBF (*isXml=false*) ou OSM-XML (*isXml=true*) dont le chemin est spécifié en premier paramètre. Si le tableau *filteredElems* est *null*, la méthode rejette tous les sous-éléments OSM des chemins et relations contenu(e)s dans le fichier de chemin *Path* à la place. Un exemple d'utilisation :  
`PBFOsmFile osm2 = PBFOsmFile.CreateInstance(@"G:\OSMFiles\liechtenstein-buildings.osm.pbf");  
osm.RejectSubDuplicates(@"G:\OSMFiles\liechtenstein-extract.xml", osm2.GetAllElems());`
- `void WriteSubElemsTo(string nodesPath, string waysPath, string relationsPath, OsmGeo[] filtered, bool relationSubNodes = false, bool isXml = true)` : Trouve tous les sous-éléments OSM des chemins et relations contenu(e)s dans le tableau passé en 4ème paramètre parmi les éléments du fichier de chemin *Path* ; puis écrit les sous-nœuds (resp. sous-chemins et sous-relations) trouvé(e)s dans un fichier de type OSM-PBF (*isXml=false*) ou OSM-XML (*isXml=true*) dont le chemin correspond au premier paramètre (resp. second

et 3ème paramètre). 2 détails importants :

1. Si le booléen *relationSubNodes* (de valeur *false* par défaut) a pour valeur *true* (resp. *false*), alors les sous-nœuds des relations contenues dans le tableau passé en 4ème paramètre seront inclus (resp. ne seront pas inclus) dans le fichier dont le chemin correspond au premier paramètre.
2. Si le tableau *filtered* est *null*, la méthode trouve tous les sous-éléments OSM des chemins et relations contenu(e)s dans le fichier de chemin *Path* à la place.
3. Exemple d'utilisation :

```
PBFOsmFile osm2 = PBFOsmFile.CreateInstance(@"G:\OSMFiles\liechtenstein-buildings.osm.pbf");
osm.WriteSubElemsTo(@"G:\OSMFiles\liechtenstein-buildings-nodes.xml",
@"G:\OSMFiles\liechtenstein-buildings-ways.xml", @"G:\OSMFiles\liechtenstein-buildings-relations.xml", osm2.GetAllElems());
```

## III/ Estimation de la hauteur des bâtiments

Pour l'estimation de la hauteur des bâtiments, j'ai premièrement choisi d'implémenter un modèle de prédiction par régression à partir d'attributs géométriques et cadastres. Ensuite, j'ai implémenté une méthode de prédiction de la hauteur par détection des contours sur des photographies de bâtiments grâce au filtre de Sobel. Nous expliquerons la première et seconde méthode de prédiction de la hauteur dans la première et seconde section respectivement.

### 1/ Méthode du modèle de prédiction de la hauteur par régression

#### a) Présentation de la régression par arbre de décision

La régression au sens mathématique recouvre plusieurs méthodes d'analyse statistique permettant d'approcher une variable à partir d'autres qui lui sont corrélées (source : Wikipédia). Ces méthodes utilisent l'apprentissage automatique (ou apprentissage « machine », équivalent de l'anglais « machine learning ») et supervisé pour essayer de prédire une variable quantitative à partir de données connues (dont on connaît au moins la valeur de la variable de prédiction pour certaines, l'apprentissage étant supervisé). Si en revanche la variable à prédire était catégorique, il aurait fallu considérer un problème de classification à la place. Dans notre cas, on aimerait prédire la hauteur de bâtiments reconstruits dans notre modèle à partir de leurs attributs géométriques et cadastres. La hauteur est bien une variable quantitative donc il faut appliquer la régression à notre modèle de prédiction.

J'ai d'ailleurs choisi d'utiliser une méthode de régression par arbre de décision. L'algorithme d'apprentissage est simple à comprendre et plutôt léger en calculs (et préparation de données), ce qui en fait l'une des méthodes les plus souvent utilisées en apprentissage automatique. Un arbre de décision représente un ensemble de choix possibles sous la forme graphique de « branches » dans un arbre, où à chacune des extrémités (ou feuilles) est associée la valeur d'une variable à prédire. Il s'agit de plus d'une représentation calculable automatiquement par des algorithmes d'apprentissage supervisé (source : Wikipédia).

Il y a 3 type de nœuds dans un arbre de décision : le nœud racine est le nœud initial au sommet de l'arbre qui représente les instances de données dans leur ensemble. Les nœuds intérieurs sont les nœuds des branches qui représentent chaque attribut des instances de données, une branche représentant elle-même une certaine règle de décision (c'est à dire un ensemble de choix possibles). Finalement, les nœuds « feuille » représentent les valeurs des variables de prédiction associées à chaque instance de donnée (c'est à dire le résultat de la prédiction). Voici ci-dessous un exemple de schéma d'arbre de décision :

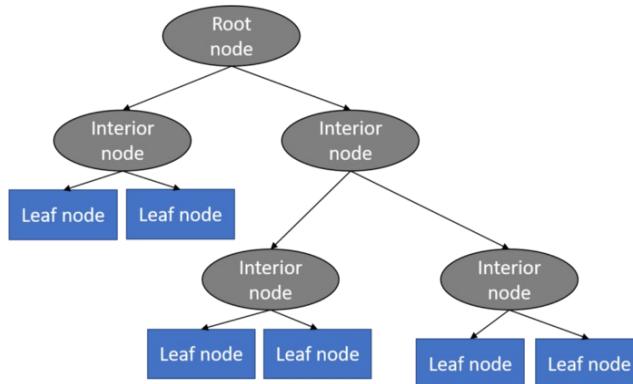


Fig. 11 : Un exemple de schéma d'arbre de décision à 4 niveaux maximum, contenant 1 nœud racine, 4 nœuds intérieurs et 6 nœuds « feuille » (source : <https://towardsdatascience.com/machine-learning-basics-decision-tree-regression-1d73ea003fda>)

### b) Un exemple d'arbre de décision pour prédire la hauteur des bâtiments

Dans notre état de l'art, nous avions déjà cité quels étaient les attributs cadastrés et géométriques des bâtiments qui nous intéressaient. Pour les attributs cadastrés, nous avions le nombre d'étages, le type, l'âge et la surface interne nette. Pour les attributs géométriques, nous avions l'aire au sol (dont la surface réelle correspond au rapport de la surface interne nette par le nombre d'étages), le nombre de bâtiments voisins dans un rayon donné et l'index de périmètre normalisé. J'ai moi-même décidé d'inclure le périmètre, la longueur et la largeur d'un bâtiment comme attributs géométriques supplémentaires. Quant au nombre d'étages et à l'âge des bâtiments, j'ai également décidé de ne pas les inclure pour la prédiction car il est souvent rare que l'élément OSM associé à un bâtiment contienne le nombre d'étages et l'âge comme attributs (ou « tags »). Dans le cas où l'attribut du nombre d'étages et de l'âge sont absents, j'ai d'ailleurs choisi 1 (bâtiment à 1 seul étage) et 0 (âge nul) comme valeurs par défaut respectivement ; ce qui implique qu'il est toujours possible de calculer la surface interne nette d'un bâtiment alors égale à son aire au sol. Or nous avons besoin de cet attribut pour effectuer la prédiction : celui-ci est d'ailleurs crucial pour réduire au maximum l'erreur entre la hauteur prédite et réelle pour chaque bâtiment (que nous avions fixé à 5m dans notre état de l'art).

Nous avons donc 6 attributs géométriques et 2 attributs cadastrés (la surface interne nette et le type de bâtiment) en tout pour notre modèle de prédiction. Connaissant la hauteur de certains bâtiments qui vont alors constituer notre ensemble de données d'entraînement, on peut maintenant construire l'arbre de décision associé puis le réutiliser sur un ensemble de données de test (ce sont les bâtiments dont on cherche à prédire la hauteur à partir des valeurs de leurs propres attributs). Voici ci-dessous un exemple de sous-partie d'arbre de décision pour prédire la hauteur des bâtiments :

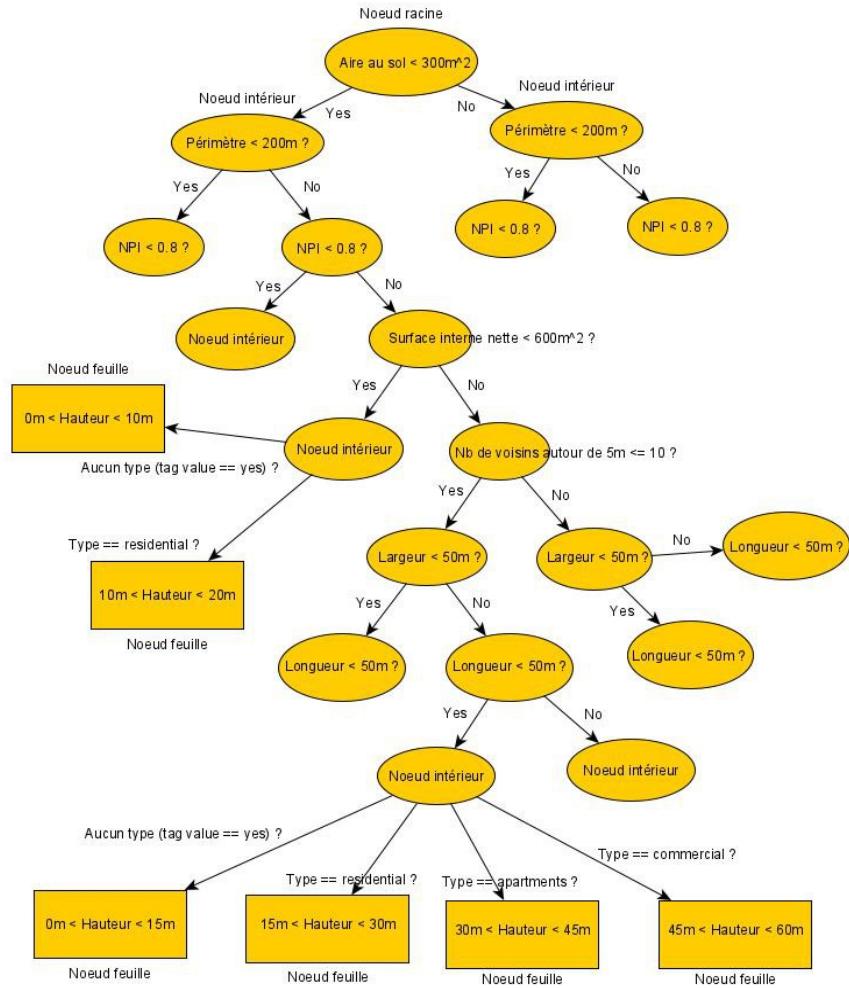


Fig. 12 : Un exemple de sous-partie d'arbre de décision, avec une profondeur maximale de 9 niveaux (conditions et prédictions de hauteur choisi(e)s au hasard ici)

### c) Construction de notre modèle de régression par arbre de décision

J'ai d'abord créé un projet séparé de l'implémentation sur Unity (j'en ai été contraint à cause de soucis de compatibilité) nommé «HeightPrediction» qui contient seulement 3 fichiers intitulés *Program.cs*, *HeightPredictor.cs* et *ObjectPicture.cs* (voir section suivante pour ce dernier fichier). Le premier fichier contient la méthode *Main* qui est le point de départ de l'exécution denotre programme pour construire notre modèle de prédiction (ou régression). Quant au second fichier, il contient la définition des classes *HeightPredictor*, *BuildingFeatures* et *HeightRegression* : la première contient des méthodes (notamment la méthode *Start*) utilisant les fonctionnalités de la librairie *Microsoft.ML* (nom de la librairie C# qui permet d'intégrer du « machine learning » dans notre projet), la seconde des champs pour représenter les attributs géométriques et cadastrales (ainsi que la hauteur réelle), et la dernière un champ unique pour représenter la hauteur à prédire par notre modèle.

Voici ci-dessous toutes les étapes nécessaires à la construction de notre modèle de prédiction de la hauteur des bâtiments du Liechtenstein :

#### c.i) Création d'un objet de type *HeightPredictor* dans la méthode *Main*

Pour ce faire, on exécute l'instruction suivante dans la méthode *Main* :

```
HeightPredictor predictor = new HeightPredictor(@"G:\Buildings\liechtenstein_training.txt",
@"G:\Buildings\liechtenstein_test.txt", @"G:\Buildings\liechtenstein_heights.txt");
```

qui va créer un objet de type *HeightPredictor* ayant comme nom *predictor* à partir de 2 chemins de fichiers texte existants et d'un chemin de fichier texte à créer :

- Le fichier dont le chemin est spécifié en premier paramètre du constructeur correspond à un ensemble d'attributs géométriques et cadastres (associés à une valeur de hauteur réelle) de bâtiments du Liechtenstein se trouvant dans l'ensemble d'entraînement (ce sont les bâtiments du Liechtenstein dont on connaît la hauteur réelle, et il y en a 50 en tout). Ce fichier est généré par notre projet sur Unity lors de l'étape spécifiée en sous-section **b)** de la section 2/ de la partie **IV**.

Chaque ligne du fichier correspond à l'ensemble des attributs géométriques et cadastres (ce sont les 8 que l'on avait spécifié à la sous-section précédente, plus l'attribut du nombre d'étages non utilisé pour la prédiction) d'un bâtiment du Liechtenstein accompagné de sa hauteur réelle.

- Le fichier dont le chemin est spécifié en second paramètre du constructeur correspond à un ensemble d'attributs géométriques et cadastres de bâtiments du Liechtenstein se trouvant dans l'ensemble de test (ce sont les bâtiments du Liechtenstein dont on ne connaît pas la hauteur réelle). Ce fichier est généré par notre projet sur Unity lors de l'étape spécifiée en sous-section **b)** de la section 2/ de la partie **IV**.
- Chaque ligne du fichier correspond à l'ensemble des attributs géométriques et cadastres d'un bâtiment du Liechtenstein dont on cherche à prédire la hauteur.
- Le fichier de sortie dont le chemin est spécifié en dernier paramètre du constructeur est celui dans lequel les valeurs de hauteur prédites pour chaque bâtiment du Liechtenstein de l'ensemble de test seront écrites. Ce fichier pourra ensuite être importé sur Unity.

### c.ii) Appel à la méthode « Start » et description détaillée de cette méthode

On fait un appel à la méthode *Start* de la classe *HeightPredictor* à partir de l'instance créée précédemment en exécutant l'instruction suivante :

```
predictor.Start();
```

Cette méthode va effectivement construire notre modèle de prédiction. Elle contient les instructions suivantes dans l'ordre :

```
MLContext mlContext = new MLContext(seed: 0);
var model = Train(mlContext);
var dataView = Evaluate(mlContext, model);
Predict(mlContext, model, dataView);
```

La première instruction va créer un objet de type *MLContext* à partir d'une valeur de « seed », utilisé pour initialiser un générateur de nombres aléatoires.

La seconde va entraîner notre modèle de régression par arbre de décision en faisant un appel à la méthode *Train*, qui demande un objet de type *MLContext*. Cette méthode retourne le modèle construit à partir d'un « pipeline » utilisant les données du fichier texte correspondant aux bâtiments du Liechtenstein de l'ensemble d'entraînement (voir définition de la méthode *Train* dans la classe *HeightPredictor* pour plus de détails).

La troisième va évaluer notre modèle de prédiction en faisant un appel à la méthode *Evaluate*, qui demande un objet de type *MLContext* ainsi que le modèle en question en premier et second paramètres respectivement. Cette méthode lit les données de l'ensemble de test et les retourne dans un objet de nom *dataView* ici. De plus, elle donne une indication de la précision de notre modèle en

calculant l'erreur quadratique moyenne et le coefficient de détermination à partir des données de l'ensemble de test lues (qui doit alors contenir une valeur de hauteur réelle pour chaque bâtiment du Liechtenstein afin de la comparer avec la hauteur prédictive par notre modèle). Pour plus de détails sur les 2 métriques d'évaluation utilisées, voir les liens suivants:

[https://fr.wikipedia.org/wiki/Erreur\\_quadratique\\_moyenne](https://fr.wikipedia.org/wiki/Erreur_quadratique_moyenne)

[https://fr.wikipedia.org/wiki/Coefficient\\_de\\_d%C3%A9termination](https://fr.wikipedia.org/wiki/Coefficient_de_d%C3%A9termination)

NB : voir définition de la méthode *Evaluate* dans la classe *HeightPredictor* pour plus de détails. Enfin, la dernière instruction va prédire les valeurs de hauteur des bâtiments du Liechtenstein de l'ensemble de test puis les écrire dans le fichier de sortie, en faisant un appel à la méthode *Predict*. Cette méthode demande un objet de type *MLContext*, le modèle de régression construit et les données de l'ensemble de test (se trouvant dans l'objet de nom *dataView*) en premier, second et dernier paramètres respectivement. Pour plus de détails sur la méthode *Predict*, voir sa définition dans la classe *HeightPredictor*.

### c.iii) Description de l'algorithme d'entraînement du modèle de régression par arbre de décision

L'algorithme d'entraînement de notre modèle de régression par arbre de décision est exécuté par la méthode *FastTree* (prédéfinie dans la librairie *Microsoft.ML*) dans la méthode *Train* vue précédemment. L'algorithme d'entraînement utilisé par *FastTree* est un algorithme d'amplification de gradient MART (Multiple Additive Regression Trees). L'amplification de gradient est une technique d'apprentissage automatique pour les problèmes de régression.

L'algorithme construit chaque arbre de régression par étapes, en utilisant une fonction de coût prédéfinie pour mesurer l'erreur à chaque étape, puis la corriger à la suivante. Ce modèle de prédiction est donc en fait un ensemble de modèles de prédiction plus faibles, typiquement des arbres de décision (« Gradient boosting »). Dans les problèmes de régression, la stimulation construit une série de tels arbres par étapes, puis sélectionne l'arbre optimal en utilisant une fonction de coût différentiable arbitraire.

MART apprend un ensemble d'arbres de décision (ou de régression). Dans les nœuds intérieurs, la décision est basée sur le test  $x \leq v$  où  $x$  est la valeur d'un attribut géométrique ou cadastre associé à un bâtiment du Liechtenstein de l'ensemble d'entraînement, et  $v$  est l'une des valeurs possibles de cet attribut. Les fonctions qui peuvent être produites par un arbre de régression sont toutes les fonctions constantes par morceaux, correspondant à des valeurs de hauteur différentes dans notre cas.

L'ensemble des arbres est produit en calculant, à chaque étape, un arbre de régression qui se rapproche du gradient de la fonction de coût, et en l'ajoutant à l'arbre précédent avec des coefficients qui minimisent le coût du nouvel arbre. La sortie de l'ensemble produit par MART sur une instance donnée (correspondant aux attributs géométriques et cadastres d'un bâtiment) est la somme des sorties de l'arbre. Dans le cas de notre problème de régression, la sortie est la valeur prédictive pour la hauteur d'un bâtiment du Liechtenstein de l'ensemble de test.

(sources : <https://docs.microsoft.com/en-us/dotnet/machine-learning/tutorials/predict-prices>  
<https://docs.microsoft.com/en-us/dotnet/api/microsoft.ml.trainers.fasttree.fasttreeregressiontrainer?view=ml-dotnet>  
[https://en.wikipedia.org/wiki/Gradient\\_boosting#Gradient\\_tree\\_boosting](https://en.wikipedia.org/wiki/Gradient_boosting#Gradient_tree_boosting))

### d) Importation du fichier contenant les valeurs de hauteur prédictives dans l'éditeur de Unity

Pour effectuer l'importation, il suffit de faire un « glisser-déposer » du fichier en question (de nom

`liechtenstein_heights.txt` ici) dans un répertoire bien spécifique appelé *TxtFiles* se trouvant à la racine du projet (dont le chemin relatif est : *Assets/Resources/TxtFiles*), soit dans l'explorateur de fichiers soit directement dans l'éditeur de Unity comme dans notre exemple. Cette action de l'utilisateur est illustrée par la figure suivante :

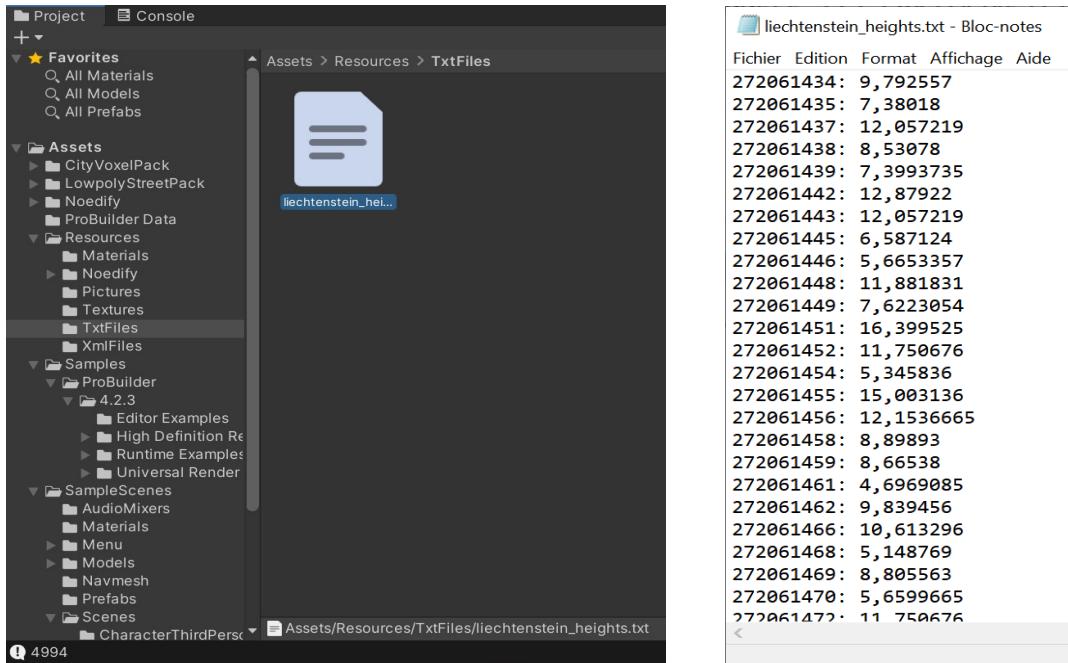


Fig. 13 à gauche : Importation des valeurs de hauteur prédictives pour chaque bâtiment du Liechtenstein de l'ensemble de test à partir de l'éditeur de Unity, par « glisser-déposer » du fichier texte associé

Fig. 14 à droite : Contenu du fichier texte auquel on associe à chaque identifiant de bâtiment du Liechtenstein de l'ensemble de test une valeur de hauteur prédictive

```
liechtenstein_test.txt - Bloc-notes
Fichier Edition Format Affichage Aide
25208578,19554.9995581,884.774114370346,0.560275246311088,1,19554.9995581,8,190.3459,174.8276,industrial,3
25208605,2282.17412681115,232.21998000145,0.729255550740799,1,2282.17412681115,20,78,41.47607,office,3
25452997,9372.2933478004,439.404578208923,0.781022053571848,3,28116.8800434012,2,166.7288,64.58618,yes,9
28712148,2077.59778558189,224.192468643188,0.720716816836304,1,2077.59778558189,26,89.70264,25.28809,yes,3
28712150,1013.18648044369,149.239666700363,0.756075840032068,1,1013.18648044369,13,61.36157,23.06567,cathedral,:,
28712183,523.698680413756,100.197515010834,0.80963398008481,1,523.698680413756,25,40.79858,28.90405,public,3
29598457,3187.39581841215,243.283402442932,0.8226541200016417,1,3187.39581841215,3,80.32666,62.66223,school,3
29599096,902.502773244686,124.907689809799,0.852589607385461,1,902.502773244686,20,26.3999,38.98828,yes,3
30809946,2069.84613505682,202.901287078857,0.79485730061869,1,2069.84613505682,7,68.23279,68.06934,yes,3
30809947,1235.41312747474,160.704805374146,0.775322131056177,1,1235.41312747474,9,68.13306,35.94775,yes,3
30809948,1739.89912879025,185.1056599617,0.798816956429136,1,1739.89912879025,7,67.16675,52.9209,yes,3
30809949,3034.0628731278,245.006057739258,0.796967118974376,1,3034.0628731278,7,77.23022,61.30273,yes,3
30809953,2997.96068067905,272.653950691223,0.711878881605183,1,2997.96068067905,5,103.6177,56.96826,yes,3
30809955,3171.60633641471,254.707631111145,0.783795236495342,1,3171.60633641471,7,83.05811,74.08496,yes,3
30809956,2940.32909038275,255.743819236755,0.751619020452084,1,2940.32909038275,6,99.69727,58.4165,industrial,3
30809957,1930.06317424645,200.331516265869,0.77739442561378,1,1930.06317424645,8,70.31738,62.6626,yes,3
30809958,759.600295800656,112.584314346313,0.867799935260777,1,759.600295800656,8,23.12317,34.0459,yes,3
30809959,2632.55572223623,262.967514038086,0.691658348744316,1,2632.55572223623,7,107.613,26.69287,yes,3
30809960,1034.39578856771,128.749773025513,0.885527061789589,1,1034.39578856771,7,34.51868,31.50244,yes,3
30809963,520.684834336626,91.319242477417,0.885788631384085,1,520.684834336626,7,24.51477,22.77832,yes,3
30809973,2936.97459320792,313.345813751221,0.61309974554658,1,2936.97459320792,6,140.3997,61.2251,yes,3
30809977,16969.2877056569,607.187962532043,0.76052516928091,1,16969.2877056569,9,223.7627,130.7544,yes,3
30809979,1327.48394126953,162.079263687134,0.796878520981523,1,1327.48394126953,8,68.21826,38.40234,yes,3
30809982,6954.72378067451,414.137717247009,0.713838710877988,1,6954.72378067451,8,161.1333,61.30273,yes,3
29800086 4184 70425812213 291 04824756807 9 7879077677879 1 4184 70425812213 8 116 5708 68 29063 yes 3
```

Fig. 15 : Contenu du fichier texte correspondant aux attributs cadastraux et géométriques de bâtiments du Liechtenstein se trouvant dans l'ensemble de test

The screenshot shows a Windows Notepad window with the title "liechtenstein\_training.txt - Bloc-notes". The window contains a large block of text representing a CSV file with 50 data rows. The columns correspond to building attributes such as ID, address, type, and various numerical values. The Notepad interface includes standard window controls (minimize, maximize, close) at the top and status bars at the bottom indicating the line and column count (Ln 1, Col 1), zoom level (100%), and encoding (Windows (CRLF), UTF-8).

Fig. 16 : Contenu du fichier texte correspondant aux attributs cadastraux et géométriques de bâtiments du Liechtenstein se trouvant dans l'ensemble d'entraînement (il y en a 50 en tout)

## 2/ Méthode de prédiction de la hauteur par détection des contours sur des photographies de bâtiments

### a) Présentation du filtre de Sobel

Le filtre de Sobel est un opérateur utilisé en traitement d'image pour la détection de contours. Il s'agit d'un des opérateurs les plus simples qui donne toutefois des résultats corrects. L'opérateur calcule le gradient de l'intensité de chaque pixel. Ceci indique la direction de la plus forte variation du clair au sombre, ainsi que le taux de changement dans cette direction. On connaît alors les points de changement soudain de luminosité, correspondant probablement à des bords, ainsi que l'orientation de ces bords.

En chaque point, le gradient pointe dans la direction du plus fort changement d'intensité, et sa longueur représente le taux de variation dans cette direction. Le gradient dans une zone d'intensité constante est donc nul. Au niveau d'un contour, le gradient traverse le contour, des intensités les plus sombres aux intensités les plus claires.

L'opérateur utilise des matrices de convolution. La matrice de taille  $3 \times 3$  subit une convolution avec l'image pour calculer des approximations des dérivées horizontale et verticale. Soit  $A$  l'image source,  $G_x$  et  $G_y$  deux images qui en chaque point contiennent des approximations respectivement de la dérivée horizontale et verticale de chaque point. Ces images sont calculées comme suit:

$$Gx = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} * A, Gy = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix} * A \quad \text{où } * \text{ est l'opérateur de convolution.}$$

La convolution est le processus consistant à ajouter chaque élément de l'image à ses voisins immédiats, pondéré par les éléments du noyau (ou filtre). C'est une forme de produit de convolution.

Par exemple, si nous avons deux matrices  $3 \times 3$ , la première étant le noyau et la seconde une partie de l'image, la convolution est le processus consistant à retourner les colonnes et les lignes du noyau, multiplier localement les valeurs ayant la même position, puis sommer le tout. L'élément aux coordonnées [2, 2] (l'élément central) de l'image de sortie devrait être pondéré par la combinaison de toutes les entrées de la matrice de l'image. Dans le cas du filtre de Sobel, on a 2 noyaux ce qui donne :  $Gx'[2, 2] = -i + g - 2f + 2d - c + a, Gy'[2, 2] = -i - 2h - g + c + 2b + a$

où  $A' = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}$  et  $Gx', Gy'$  et  $A'$  représentent une matrice  $3 \times 3$  extraite des images respectives  $Gx, Gy$  et  $A$  auxquelles on aura agrandi les dimensions d'une ligne/colonne remplie de valeurs nulles sur chaque côté de l'image (« zero padding »).

En chaque point, les approximations des gradients horizontaux et verticaux peuvent être combinées comme suit pour obtenir une approximation de la norme du gradient :

$$G = \sqrt{Gx^2 + Gy^2}$$

Il suffit maintenant de comparer chaque valeur de l'image résultante  $G$  avec une valeur de seuil choisie afin d'appliquer une segmentation de l'image : ainsi, l'image segmentée contiendra une valeur nulle (resp. égale à 1), c'est à dire un pixel noir (resp. pixel blanc) partout là où les valeurs de  $G$  sont inférieures (resp. supérieures ou égales) à la valeur de seuil.

(sources : [https://fr.wikipedia.org/wiki/Filtre\\_de\\_Sobel](https://fr.wikipedia.org/wiki/Filtre_de_Sobel)  
[https://fr.wikipedia.org/wiki/Noyau\\_\(traitement\\_d%27image\)#Convolution](https://fr.wikipedia.org/wiki/Noyau_(traitement_d%27image)#Convolution))

## b) Détection des contours sur une photographie d'un bâtiment du Liechtenstein

Appliquons maintenant le filtre de Sobel à la détection des contours sur une photographie d'un bâtiment du Liechtenstein dont on ne connaît pas la hauteur : la cathédrale de St-Florin.

Pour cela, il faudra utiliser les fichiers intitulés *Program.cs* et *ObjectPicture.cs* se trouvant dans le projet nommé «HeightPrediction» (vu à la section précédente). Le premier fichier contient la méthode *Main* qui est le point de départ de l'exécution du programme pour la détection des contours. Quant au second fichier, il contient la définition de la classe utilitaire *ObjectPicture* qui contient des méthodes utilisant la classe pré définie *Bitmap* (représentant une image 2D en tant qu'ensemble de pixels) et de l'énumération *ColorComponent* qui contient les valeurs *Red*, *Green* et *Blue*.

Voici ci-dessous toutes les étapes nécessaires à la détection des contours sur la photographie de la cathédrale de St-Florin :

### b.i) Cr ation d'un objet de type *ObjectPicture* dans la m thode *Main*

Pour ce faire, on ex ute l'instruction suivante dans la m thode *Main* :

```
ObjectPicture picture = new ObjectPicture(@"G:\Buildings\cathedral-of-st-florin.jpg",
Color.White, Color.Gray);
```

qui va cr er un objet de type *ObjectPicture* de nom *picture*   partir d'une image de format JPG (les autres formats possibles  tant BMP, GIF, EXIF, PNG et TIFF) dont le chemin est sp cifi  en premier param tre, et des 2 couleurs sp cifi es en second et dernier param tres. Le constructeur va ensuite cr er un objet de type *Bitmap*   partir du chemin d'image envoy  et associer la couleur minimale au gris et la couleur maximale au blanc. Plus g n ralement, le constructeur ex ute les 2 instructions suivantes pour trouver la couleur minimale *MinColor* et maximale *MaxColor*   partir de 2 couleurs *c1* et *c2* envoy es :

```
MinColor = Color.FromArgb(Math.Min(c1.A, c2.A), Math.Min(c1.R, c2.R), Math.Min(c1.G, c2.G),
Math.Min(c1.B, c2.B));
MaxColor = Color.FromArgb(Math.Max(c1.A, c2.A), Math.Max(c1.R, c2.R), Math.Max(c1.G, c2.G),
Math.Max(c1.B, c2.B));
```

Ces couleurs minimale et maximale seront utiles pour d finir la plage de couleurs du b timent   identifier sur l'image, afin d'appliquer une segmentation de l'image pour d tecter les contours.

### b.ii) Application du filtre de Sobel   la photographie et sauvegarde de l'image filtr e

Pour appliquer le filtre de Sobel   la photographie du b timent en question et sauvegarder l'image filtr e sur le disque, on ex ute l'instruction suivante :

```
picture.FindEdges(150, ColorComponent.Red).Save(@"G:\Buildings\cathedral_florin_edges.jpg");
```

qui va d'abord appeler la m thode *FindEdges* en fournissant une valeur de seuil  gale   150 et la valeur litt rale de l'enum ration *ColorComponent* correspondant au rouge en premier et second param tres respectivement. Cette m thode va effectuer les  tapes suivantes dans l'ordre :

- Invoquer la m thode statique *ApplySobelFilter* en fournissant l'image du b timent stock e par l'objet *picture*, une valeur de seuil  gale   150 et la valeur litt rale de l'enum ration *ColorComponent* correspondant au rouge en premier, second et dernier param tres respectivement.
- La m thode *ApplySobelFilter* applique le filtre de Sobel   la composante rouge de l'image envoy e ( quivalente   l'image convertie en niveaux de gris) en invoquant 2 fois la m thode statique *ConvolveMatrix* qui demande un objet de type *Bitmap*, un noyau sous forme de matrice et une valeur litt rale de l'enum ration *ColorComponent* en premier, second et dernier param tres. Dans notre cas, la m thode *ConvolveMatrix* va appliquer une convolution 2 fois sur la composante rouge de l'image originale   partir de chacun des 2 noyaux du filtre de Sobel.
- Ensuite, l'image originale est segment e en s lectionnant seulement les pixels dont la norme du gradient calcul e   partir des r sultats des 2 convolutions est sup rieure ou  gale   150. Finalement, la m thode *ApplySobelFilter* retourne l'image originale segment e (en pixels noir et blanc) qui est un objet de type *Bitmap*.
- Maintenant, la m thode *FindEdges* va segmenter l'image originale une seconde fois en s lectionnant seulement les pixels qui correspondent   un contour blanc sur l'image filtr e (par l'op rateur de Sobel) et dont la couleur se trouve entre le gris (couleur minimale *MinColor*) et le blanc (couleur maximale *MaxColor*). Ensuite, la m thode retourne l'image originale segment e (en pixels noir et blanc) qui est un objet de type *Bitmap*.

Dernièrement, la même instruction va appeler la méthode prédéfinie *Save* de la classe *Bitmap* pour sauvegarder l'objet de type *Bitmap* retourné par *FindEdges* (l'image filtrée) en tant qu'image dont le chemin est spécifié en paramètre.

NB :

- voir définition de la méthode *FindEdges* dans la classe *ObjectPicture* pour plus de détails.
- Dans le cas où il est difficile d'établir une plage de couleurs spécifique au bâtiment sur l'image originale (ou si cette plage correspond à presque toutes les couleurs RGB possibles), le filtre de Sobel peut être appliqué à cette image en appelant la méthode *ApplySobelFilter* à la place de *FindEdges*.

### b.iii) Visualisation de la photographie originale et des images filtrées

Dans les figures ci-dessous, on a visualisé la photographie originale ainsi que les images filtrées par application du filtre de Sobel sur cette photographie :



Fig. 17 : La photographie originale de la cathédrale de St-Florin au Liechtenstein



Fig. 18 : Le résultat de l'application du filtre de Sobel à la photographie originale avec *ApplySobelFilter*. Les contours en blanc peuvent ensuite être analysés sur Unity afin d'estimer les dimensions du bâtiment.



Fig. 19 : Le résultat de l'application du filtre de Sobel à la photographie originale avec *FindEdges*. Les contours en blanc peuvent ici être analysés sur Unity afin d'estimer les dimensions du bâtiment avec encore plus de précision.

Comme la méthode d'estimation des dimensions du bâtiment à partir de l'analyse des contours en blancs est souvent imprécise, il peut être nécessaire de marquer les dimensions du bâtiment sur la photographie directement. Pour ce faire, il faut ouvrir un éditeur d'images tel que *Paint* et tracer un rectangle d'une certaine couleur de marquage (rouge par exemple) sur la photographie, dont les dimensions sont égales à celle du bâtiment. L'avantage de cette méthode de marquage est qu'elle permet de calculer les dimensions du bâtiment sur Unity avec plus de précision, mais son inconvénient elle qu'elle nécessite l'intervention de l'utilisateur.

Voici un exemple de marquage sur une photographie :

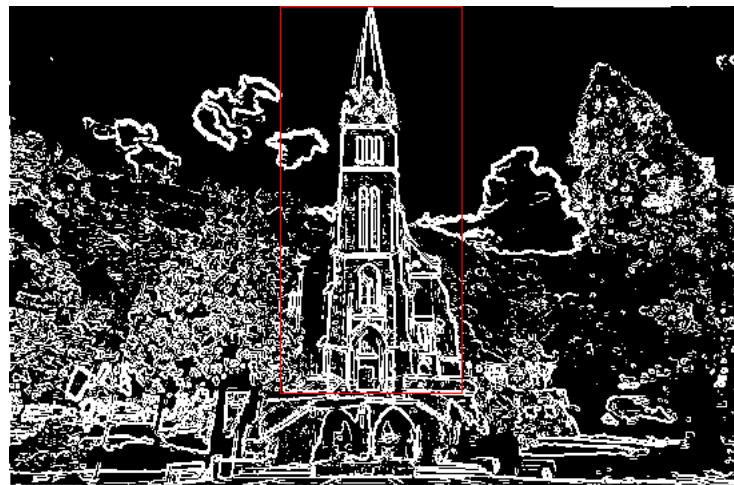


Fig. 20 : Marquage en rouge des dimensions de la cathédrale de St-Florin au Liechtenstein sur la photographie originale à laquelle on a appliqué le filtre de Sobel.

NB : Il est préférable de marquer les dimensions du bâtiment sur une photographie convertie en niveaux de gris (comme ici après application du filtre de Sobel) afin de ne pas confondre la couleur de marquage avec les couleurs de la photographie.

### c) Formules utilisées pour le calcul de la hauteur d'un bâtiment à partir d'une photographie

Pour calculer la hauteur  $H$  d'un bâtiment dont on connaît la largeur  $W$  ou la longueur  $L$  dans notre modèle à partir d'une photographie de l'une de ses faces, on utilise l'une des 2 relations de proportionnalité suivantes :  $H = W * h / w$  ou  $H = L * h / w$   
où  $h$  et  $w$  sont la hauteur et largeur en pixels du bâtiment sur la photographie respectivement.

La largeur  $W$  (resp. longueur  $L$ ) d'un bâtiment dans notre modèle correspond à la différence de la latitude (resp. longitude) maximale par la latitude (resp. longitude) minimale d'un sous-nœud du chemin OSM associé à ce bâtiment. Cette différence de latitudes ou longitudes étant exprimée en degrés, elle doit être convertie en distances en mètres dans notre modèle. Pour cela, on utilise le fait que les coordonnées  $(z, x)$  dans notre modèle d'un point de latitude  $lat$  et longitude  $lon$  (par exemple un sous-nœud de chemin OSM) sont données par :

$$(z, x) = (lat * avgdist(lat) - tcLat * avgdist(tcLat), (lon - tcLon) * C * \cos(lat - tcLat) / 360)$$

où :

- $tcLat$  et  $tcLon$  sont la latitude et longitude du centre du terrain respectivement
- $C=40075017m$  est la circonference de la Terre
- $avgdist(lat)$  est la distance moyenne en mètres que représente un degré à la latitude  $lat$ , calculée comme la moyenne des distances que représente un degré à chaque latitude

inférieure ou égale à  $lat$ . En effet, la distance d'un degré varie en fonction de la latitude à laquelle on se trouve. Par exemple,  $1^\circ$  de latitude à l'équateur (latitude nulle) correspond à 110574m alors qu' $1^\circ$  de latitude aux pôles (latitude minimale/maximale) correspond à 111694m.

De plus, à une latitude donnée  $lat$ ,  $1^\circ$  de longitude correspond à :  $C * \cos(lat) / 360$  m.

Pour plus d'explications sur la formule utilisée pour calculer les coordonnées ( $z$ ,  $x$ ), voir le lien suivant : [https://fr.wikipedia.org/wiki/Coordonn%C3%A9es\\_g%C3%A9ographiques](https://fr.wikipedia.org/wiki/Coordonn%C3%A9es_g%C3%A9ographiques)

Le calcul de la hauteur à partir de l'une des 2 relations de proportionnalité citées plus haut n'est possible seulement si la photographie est prise face au sens de la largeur  $W$  ou la longueur  $L$  du bâtiment. Si la photographie est prise face au sens de la largeur  $W$  (resp. longueur  $L$ ), c'est à dire face à l'axe des latitudes (resp. longitudes) ou face Est/Ouest (resp. face Nord/Sud), alors la relation de proportionnalité à utiliser doit être :  $H = W * h / w$  (resp.  $H = L * h / w$ ).

Or, la photographie d'un bâtiment n'est souvent pas prise face au sens de la largeur  $W$  ou la longueur  $L$ , c'est à dire qu'elle est prise dans une direction autre que Nord-Sud et Est-Ouest. Dans ce cas-là, la largeur  $w$  en pixels du bâtiment sur la photographie ne correspond ni à la largeur  $W$  ni à la longueur  $L$  du bâtiment dans notre modèle (seule la hauteur  $h$  en pixels doit correspondre à la hauteur prédictive  $H$  dans notre modèle).

Il est donc nécessaire de recalculer la largeur et la longueur du bâtiment dans notre modèle à partir du point de vue de la photographie, c'est à dire de la position de notre avatar. Plus précisément, la nouvelle longueur  $L'$  et largeur  $W'$  sont données par :

$$L' = L * \cos(angle) + W * \sin(angle)$$

$$W' = W * \cos(angle) + L * \sin(angle)$$

$$\text{où } angle = \arctan\left(\frac{\min(|x_{pos} - x_{bary}|, |z_{pos} - z_{bary}|)}{\max(|x_{pos} - x_{bary}|, |z_{pos} - z_{bary}|)}\right)$$

avec  $(z_{pos}, x_{pos})$  les coordonnées actuelles de notre avatar, et  $(z_{bary}, x_{bary})$  les coordonnées du barycentre du polygone associé à la forme du bâtiment dans notre modèle.

On remarque donc que la longueur  $L'$  et largeur  $W'$  dépendent de l'angle auquel se trouve notre avatar par rapport à l'axe des latitudes ou longitudes (cet angle étant toujours compris entre 0 et  $45^\circ$  inclus) dans notre modèle. Par ailleurs, cet angle doit correspondre à l'angle de prise du bâtiment sur la photographie par rapport aux directions Nord-Sud et Est-Ouest. Si l'angle de prise n'est pas connu, il faut se positionner dans notre modèle de sorte à avoir le même point de vue que sur la photographie du bâtiment.

Les 2 relations de proportionnalité citées plus haut deviennent alors :

$$H = W' * h / w = (L * \cos(angle) + W * \sin(angle)) * h / w$$

$$H = L' * h / w = (W * \cos(angle) + L * \sin(angle)) * h / w$$

et la hauteur prédictive  $H$  dépend alors de l'angle donné par la formule plus haut, c'est à dire de la position de notre avatar.

La formule utilisée pour le calcul de la nouvelle longueur  $L'$  et largeur  $W'$  est illustrée par la figure suivante :

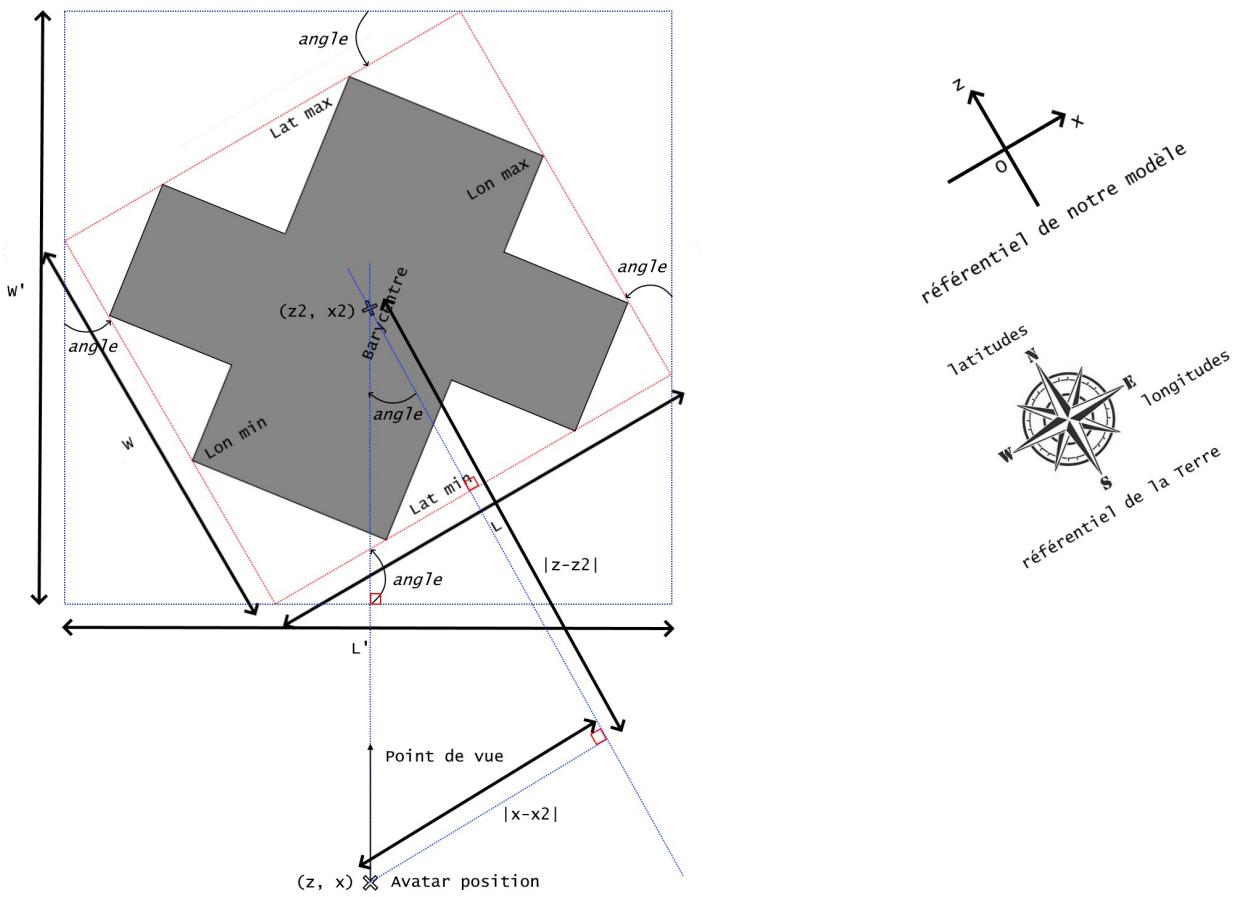


Fig. 21 : Illustration de la méthode de calcul de la nouvelle longueur  $L'$  et largeur  $W'$  d'un bâtiment à partir de sa longueur initiale  $L$ , de sa largeur initiale  $W$ , et de l'angle auquel se trouve notre avatar par rapport à l'axe des latitudes. Ici, le référentiel de notre modèle ainsi que celui de la Terre sont inclinés vers la droite du même angle afin de se trouver alignés par rapport à l'axe des latitudes et longitudes. De plus, le point de vue de notre avatar dans notre modèle doit correspondre avec celui de la photographie du bâtiment.

#### d) Explication de la méthode de prédiction de la hauteur à partir de la détection des contours sur une photographie d'un bâtiment du Liechtenstein

Après application du filtre de Sobel à la photographie d'un bâtiment du Liechtenstein, on peut estimer la hauteur du bâtiment à partir de l'analyse des contours en blancs. L'idée de base est que les contours d'un bâtiment sont souvent verticaux sur une photographie, à condition que ce bâtiment ait été pris droit et de face (le bâtiment ne doit pas se trouver de « travers » sur la photographie). Ainsi, il suffit d'identifier toutes les lignes verticales blanches contenant au minimum  $nPixels$  à la suite dans une même colonne, sur l'image filtrée à laquelle on a appliquée la détection de contours. Les dimensions du bâtiment sont ensuite calculées en fonction des indices de ligne et de colonne minimaux/maximaux correspondant aux positions de ces lignes verticales blanches.

Par exemple, pour le cas de la figure 9 plus haut représentant les contours de la cathédrale de St-Florin au Liechtenstein, on obtient une largeur de 63 pixels et une hauteur de 141 pixels après exécution des instructions suivantes :

```
ObjectPicture picture = new ObjectPicture(@"G:\Buildings\cathedral-of-st-florin.jpg",
Color.White, Color.Gray);
Console.WriteLine(picture.IdentifyObject(20, 150, ColorComponent.Red));
```

où on a :  $nPixels=20$  et  $thr=150$  (voir la méthode *IdentifyObject* de la sous-section e) pour plus de détails).

Or, nous savons grâce à notre modèle 3D sur Unity que cette cathédrale fait exactement 23.06567m de large et 61.36499m de long (voir l'exemple de la cathédrale de St-Florin en partie V/). De plus, on sait que la photographie de cette cathédrale en figure 17, 18, 19 et 20 plus haut a été prise dans le sens de la largeur (il est facile de voir que c'est la face la moins large de la cathédrale). On en déduit donc que la hauteur de la cathédrale de St-Florin serait proche de :  $141 * 23.06567 / 63 = 51.62$ m.

Ce résultat est bien cohérent avec la réalité car la cathédrale fait environ 50m. Finalement, cet exemple montre que la méthode d'estimation de la hauteur du bâtiment à partir de l'analyse des contours en blancs peut être assez précise si la valeur de  $nPixels$  est bien choisie. Cependant, cette méthode a des limites : d'une part, elle nécessite de trouver la valeur optimale de  $nPixels$  et d'autre part, elle n'est pas très efficace lorsque les contours du bâtiment sont discontinus.

De plus, les dimensions du bâtiment (en nombre de pixels) calculées sur la photographie ne représentent parfois pas les dimensions réelles, bien que le rapport de la hauteur sur la largeur soit proportionnel au ratio réel : c'est le cas de notre exemple, donc il s'agit plus d'un coup de chance d'être tombé sur une hauteur cohérente avec la réalité en choisissant :  $nPixels=20$ .

Il est donc ici nécessaire d'utiliser la méthode de marquage qui effectuera une meilleure prédiction de la hauteur. En prenant la photographie de la cathédrale en figure 20 plus haut sur laquelle on a marqué les dimensions en rouge, on obtient ainsi une largeur de 137 pixels et une hauteur de 290 pixels. On en déduit donc que la hauteur de la cathédrale serait proche de :  $290 * 23.06567 / 137 = 48.82515$ m avec cette méthode. Ce résultat est bien plus cohérent avec la réalité ce qui montre que la méthode de marquage est bien plus précise et adaptée à notre cas.

#### e) Importation des images filtrées dans l'éditeur de Unity

Pour effectuer l'importation, il suffit de faire un « glisser-déposer » des images filtrées en question dans un répertoire bien spécifique appelé *Pictures* se trouvant à la racine du projet (dont le chemin relatif est : *Assets/Resources/Pictures*), soit dans l'explorateur de fichiers soit directement dans l'éditeur de Unity comme dans notre exemple.

Cette action de l'utilisateur est illustrée par la figure suivante :

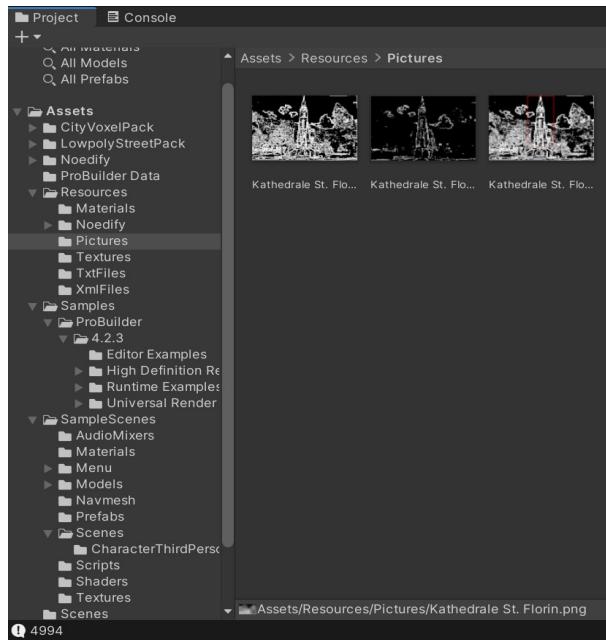


Fig. 22 : Importation des images filtrées par « glisser-déposer » dans l'éditeur de Unity

L'image qui va être utilisée initialement pour le calcul des dimensions du bâtiment sur Unity doit avoir comme nom :

- `<buildingName>.*` où \* est l'extension de l'image (les formats d'image autorisés par Unity étant : JPG, BMP, GIF, PNG, TIFF, EXR, HDR, IFF, PICT, PSD et TGA) et `buildingName` est le nom du bâtiment associé à l'attribut `name` de l'élément OSM correspondant.
- Ou `<buildingId>.*` où \* est l'extension de l'image et `buildingId` est l'identifiant de l'élément OSM correspondant au bâtiment, dans le cas où cet élément ne possède pas d'attribut `name`.

Les autres images importées sur Unity correspondant à ce bâtiment doivent donc avoir un nom différent, comme dans le cas de la figure ci-dessus.

Par ailleurs, il est nécessaire de modifier certaines options des images importées dans la fenêtre de l'inspecteur sur Unity (voir partie IV/) comme suit :

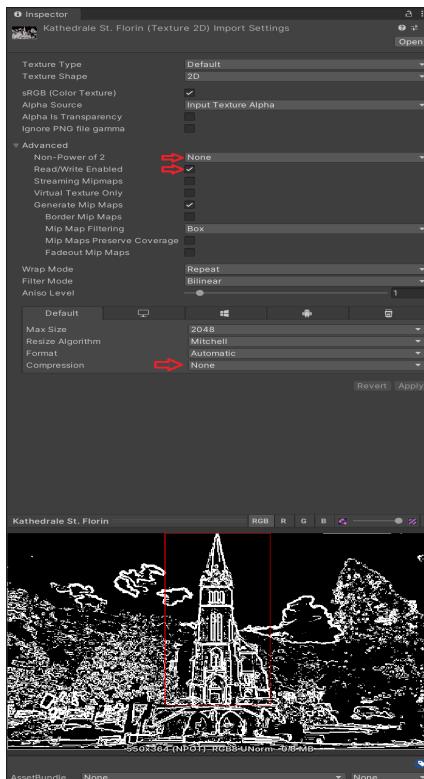


Fig. 23 : Modification des options suivantes dans la fenêtre de l'inspecteur correspondant à l'image du bâtiment sur Unity : « Non-Power of 2 », « Read/Write Enabled » et « Compression »

#### f) Index des propriétés et méthodes publiques de la classe *ObjectPicture*

Ici, je présente l'index de toutes les propriétés et méthodes (avec leur signature) de la classe *ObjectPicture* :

- `delegate (double, double)? RotateObjFunc(double w, double h, double angle)` : Champ de type « délégué » qui définit la signature d'une méthode de la classe *ObjectPicture*
- `enum ColorComponent` : Énumération représentant les 3 composantes de couleur R (rouge), G (vert) et B (bleu). Rappelons qu'une image RGB correspond à la somme de ses 3 composantes R, G et B.
- `Bitmap ObjImage` : Propriété de la classe représentant la photographie d'un bâtiment (qui est un objet de type prédéfini *Bitmap*)
- `Color MinColor` : Propriété de la classe représentant la couleur minimale du bâtiment sur la photographie (la couleur minimale d'un pixel étant le noir et la couleur maximale le blanc)
- `Color MaxColor` : Propriété de la classe représentant la couleur maximale du bâtiment sur la photographie
- `int ObjHeight` : Propriété de la classe représentant la hauteur estimée du bâtiment sur la photographie, calculée par segmentation des couleurs
- `int ObjWidth` : Propriété de la classe représentant la largeur estimée du bâtiment sur la photographie, calculée par segmentation des couleurs
- `Rectangle ObjRect` : Propriété de la classe représentant un rectangle délimitant les contours estimés du bâtiment sur la photographie, calculés par segmentation des couleurs
- `ObjectPicture(string path, Color c1, Color c2)` : Constructeur de la classe associant aux propriétés *ObjImage*, *MinColor* et *MaxColor* une image dont le chemin est *path*, et les couleurs minimale et maximale calculées à partir des couleurs *c1* et *c2* respectivement.  
Un exemple d'utilisation :

- ```

ObjectPicture picture = new ObjectPicture(@"G:\Buildings\cathedral-of-st-florin.jpg",
Color.White, Color.Gray);

```
- `ObjectPicture(Bitmap bitmap, Color c1, Color c2)` : Constructeur de la classe associant aux propriétés *ObjImage*, *MinColor* et *MaxColor* une copie de l'image *bitmap*, et les couleurs minimale et maximale calculées à partir des couleurs *c1* et *c2* respectivement.
Un exemple d'utilisation :

```

ObjectPicture picture2=new ObjectPicture(ObjectPicture.ToGrayscale(picture.ObjImage),
Color.FromArgb(0, 0, 0), Color.FromArgb(25, 25, 25));

```
 - `Bitmap Crop(Rectangle rect)` : Rogne la photographie selon une zone délimitée par un rectangle d'une certaine taille puis retourne l'image rognée. Un exemple d'utilisation :

```

Rectangle rect = picture.ObjRect;
picture.Crop(new Rectangle(rect.X, rect.Y, picture.ObjWidth,
picture.ObjHeight)).Save(@"G:\Buildings\cathedral_florin_cropped.jpg");

```
 - `static void PrintColors(Bitmap bitmap, int nColors)` : Affiche la couleur de chaque pixel de l'image spécifiée en premier paramètre, avec *nColors* couleurs à la fois affichées avant chaque pause (dont l'utilisateur pourra mettre fin en appuyant sur la touche *Enter*). Les pixels sont parcourus de gauche à droite et de haut en bas dans l'ordre des colonnes de l'image.
 - `static Bitmap ToGrayscale(Bitmap bitmap)` : Convertit l'image RGB spécifiée en premier paramètre en image de niveaux de gris, puis retourne cette image. Un exemple d'utilisation :

```

ObjectPicture.ToGrayscale(picture.ObjImage).Save(@"G:\Buildings\cathedral_florin_grayscale.jpg");

```
 - `static Bitmap Convolve(Bitmap bitmap, double[,] kernel, ColorComponent comp)` : Applique une convolution à la composante rouge, verte ou bleue (selon la valeur de *comp*) de l'image RGB spécifiée en premier paramètre à partir du noyau spécifié en second paramètre, puis retourne l'image convolutée.
 - `static double[,] ConvolveMatrix(Bitmap bitmap, double[,] kernel, ColorComponent comp)` : Applique une convolution à la composante rouge, verte ou bleue de l'image RGB (selon la valeur de *comp*) spécifiée en premier paramètre à partir du noyau spécifié en second paramètre, puis retourne l'image convolutée sous forme de matrice 2D.
 - `static Bitmap ApplySobelFilter(Bitmap bitmap, double thr, ColorComponent comp)` : Applique le filtre de Sobel à la composante rouge, verte ou bleue (selon la valeur de *comp*) de l'image RGB spécifiée en premier paramètre, puis retourne l'image filtrée et segmentée. Cette image est segmentée en sélectionnant seulement les pixels dont la norme du gradient calculée à partir des résultats des 2 convolutions est supérieure ou égale à *thr*. Un exemple d'utilisation :

```

ObjectPicture.ApplySobelFilter(picture.ObjImage, 150,
ColorComponent.Red).Save(@"G:\Buildings\cathedral_florin_sobel.jpg");

```
 - `Bitmap FindEdges(double thr, ColorComponent comp)` : Déetecte les contours de la photographie par segmentation, puis retourne l'image segmentée. Cette image est segmentée en sélectionnant seulement les pixels qui correspondent à un contour blanc sur l'image filtrée par l'opérateur de Sobel, et dont la couleur se trouve entre la couleur minimale *MinColor* et maximale *MaxColor*. Un exemple d'utilisation :

```

picture.FindEdges(150,ColorComponent.Red).Save(@"G:\Buildings\cathedral_florin_edges.jpg");

```
 - `Size IdentifyObject(int nPixels, double thr, ColorComponent comp)` : Identifie le bâtiment sur la photographie par détection des contours, puis retourne les dimensions estimées du bâtiment. Pour ce faire, la méthode va identifier toutes les lignes verticales blanches contenant au minimum *nPixels* à la suite dans une même colonne, sur l'image filtrée à laquelle on a appliqué la détection de contours (les contours de bâtiments étant souvent verticaux sur une photographie). Les dimensions sont ensuite calculées en fonction des indices de ligne et de colonne minimaux/maximaux correspondant aux positions de ces lignes verticales blanches.

Un exemple d'utilisation :

```
Console.WriteLine(picture.IdentifyObject(18, 150, ColorComponent.Red));
```

Les fonctions suivantes sont utiles dans le cas où un bâtiment est en rotation sur la photographie et que l'on connaît l'angle de rotation du bâtiment selon chaque axe x , y et z :

- `static (double, double)? RotateObjX(double w, double h, double angle)` : Calcule la taille réelle d'un objet de largeur w et de hauteur h en rotation d'un angle $angle$ selon l'axe x sur la photographie (supposée prise face au plan (Oxy) dans un système de coordonnées dans l'espace d'origine O).
- `static (double, double)? RotateObjY(double w, double h, double angle)` : Calcule la taille réelle d'un objet de largeur w et de hauteur h en rotation d'un angle $angle$ selon l'axe y sur la photographie (supposée prise face au plan (Oxy) dans un système de coordonnées dans l'espace d'origine O).
- `static (double, double)? RotateObjZ(double width, double height, double angle)` : Calcule la taille réelle d'un objet de largeur w et de hauteur h en rotation d'un angle $angle$ selon l'axe z sur la photographie (supposée prise face au plan (Oxy) dans un système de coordonnées dans l'espace d'origine O).
- Un exemple d'utilisation des 3 fonctions précédentes :

```
double angleRad = 30 * Math.PI / 180;
int w = picture.ObjWidth, h = picture.ObjHeight;
(double w, double h)? sizeObjX = ObjectPicture.RotateObjX(w, h, angleRad),
sizeObjY = ObjectPicture.RotateObjY(w, h, angleRad), sizeObjZ =
ObjectPicture.RotateObjZ(w, h, angleRad);
```
- `static (double, double)? RotateObj(double w, double h, double[] angles, RotateObjFunc[] funcs)` : Calcule la taille réelle d'un objet de largeur w et de hauteur h en rotation d'un certain angle selon l'axe x , y et z sur la photographie (supposée prise face au plan (Oxy) dans un système de coordonnées dans l'espace d'origine O). L'argument $funcs$ donne l'ordre de rotation de l'objet, donc l'ordre des angles dans le tableau $angles$.
- `(double, double)? RotateObjXYZ(double w, double h, Quaternion quat)` : Calcule la taille réelle d'un objet de largeur w et de hauteur h en rotation d'un certain angle d'abord selon l'axe x puis l'axe y puis l'axe z sur la photographie (supposée prise face au plan (Oxy) dans un système de coordonnées dans l'espace d'origine O).
- Idem pour `RotateObjXZY`, `RotateObjYXZ`, `RotateObjYZX`, `RotateObjZYX` et `RotateObjZXY...`
- `static bool CheckObjRot(RotateObjFunc func, ObjectPicture xyPic, ObjectPicture xzPic, ObjectPicture yzPic, Quaternion quat, float tol)` : Vérifie la taille (longueur, largeur et hauteur) d'un objet en rotation d'un certain angle selon l'axe x , y ou z à partir de 3 photographies de l'objet, chacune prise face à l'un des 3 plans (Oxy), (Oxz) et (Oyz) (l'origine O étant située au barycentre de l'objet). L'argument $func$ donne l'axe de rotation utilisé et l'argument tol la tolérance utilisée pour la vérification de l'égalité des tailles réelles de l'objet, calculées pour chacune des 3 photographies.

IV/ Implémentation du modèle 3D CityGML sur Unity

Pour l'implémentation de notre modèle 3D CityGML sur Unity, j'ai donc créé un projet sur Unity séparé des 2 autres projets vus en partie II/ et III/. Pour commencer, je présenterai l'interface utilisateur de Unity. Ensuite, je citerai les actions de l'utilisateur nécessaires à la construction (par lancement de notre application sur Unity) ainsi qu'à la modification de notre modèle (pour les attributs des bâtiments). Enfin, je présenterai le diagramme de classes de notre modèle ainsi que les responsabilités de chaque classe.

1/ Présentation de l'interface utilisateur de Unity

L'interface utilisateur de notre projet se présente comme ceci :

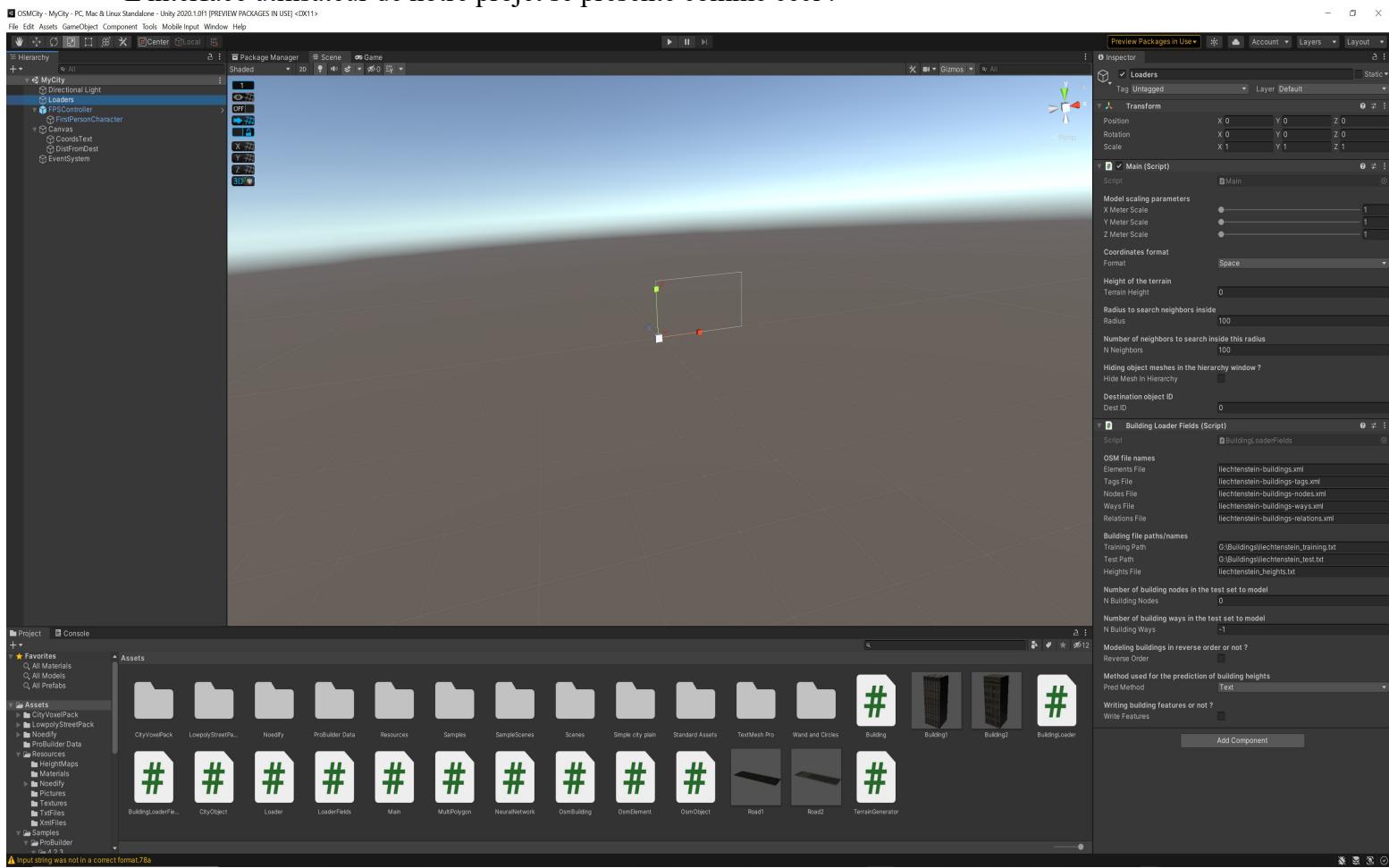


Fig. 24 : L'interface utilisateur de Unity, composée de 4 fenêtres visibles : la scène, la fenêtre de hiérarchie, la fenêtre de projet et la fenêtre d'inspecteur

Nous observons que l'interface utilisateur est ici composée de 4 fenêtres visibles :

- La scène : C'est la fenêtre principale dans laquelle nous pouvons visualiser les objets 3D (ou entités de nom « GameObject » sur Unity) créés par notre application sur Unity. Elle permet à l'utilisateur de naviguer dans l'espace en 3D et d'y créer, modifier et supprimer des objets

3D divers. Dans notre cas, il n'y a rien dans la scène et ce sera la responsabilité de l'application d'y créer automatiquement les objets urbains de notre modèle : c'est à dire le terrain et les bâtiments à y placer (j'envisage cependant de créer d'autres types d'objet urbain par la suite : routes, lacs, rivières, espaces verts, etc...).

- La fenêtre de hiérarchie : Elle référence les objets 3D de la scène pour pouvoir les retrouver facilement. Au cours de l'exécution de notre application sur Unity, le contenu de la scène et de cette fenêtre changent en même temps puisqu'ils référencent tous deux un même objet 3D : tout ajout ou suppression d'objet 3D dans la scène sera reflété(e) dans la fenêtre de hiérarchie et vice-versa. Initialement, cette fenêtre contient 5 entités « *GameObject* » fondamentales invisibles à l'utilisateur dans la scène de nom *MyCity* :
 1. *Directional Light* : contrôle les effets de lumière dans la scène, nécessaire pour voir les objets 3D dans la scène.
 2. *Loaders* : objet associé au « script » nommé *Main* qui va charger les données OSM sur les bâtiments puis effectivement construire notre modèle 3D.
 3. *FPSController* : permet à l'utilisateur d'incarner un avatar (personnage humain) qui peut se déplacer et sauter dans la scène. La position initiale de cet avatar devrait avoir $(x, y, z)=(0, 1, 0)$ comme coordonnées. La présence de cet objet nécessite de supprimer l'objet *MainCamera* correspondant à la caméra principale de la scène.
 4. *CoordsText* : champ de texte permettant d'afficher les coordonnées actuelles de notre avatar sur Terre, par exemple les valeurs de sa latitude et longitude.
 5. *DistFromDest* : champ de texte permettant d'afficher la distance restante à parcourir par notre avatar en mètres pour rejoindre le bâtiment destinataire.
- La fenêtre de projet : Elle permet de naviguer parmi les fichiers du dossier de notre projet. Ce dossier peut notamment contenir :
 1. Des fichiers de métadonnées relatives à Unity et à notre projet.
 2. Des « packages », qui représentent des librairies de Unity permettant de réutiliser du code pour les scripts.
 3. Des entités « assets », qui représentent n'importe quel objet pouvant être réutilisé dans notre projet. Une entité « asset » peut provenir d'un fichier créé en dehors de Unity, tel qu'un modèle 3D, un fichier audio, une image ou tout autre type de fichier pris en charge par Unity. Sur la figure ci-dessus, nous pouvons voir quelques entités « assets » dans la fenêtre de projet, qui doivent alors se trouver dans le dossier spécifique de nom *Assets*.
 4. Des entités « prefabs », qui sont un type d'entité « asset » qui permet de stocker une entité « *GameObject* » avec toutes ses propriétés. Une entité « prefab » agit comme un modèle à partir duquel on peut créer de nouvelles instances d'objet dans la scène. Toutes les modifications apportées à une entité « prefab » sont immédiatement reflétées dans toutes les instances produites à partir de celle-ci. Sur la figure ci-dessus, nous pouvons voir 4 entités « prefab » dans la fenêtre de projet : 2 modèles 3D de bâtiment et 2 modèles 3D de route.
 5. Des scripts, qui sont des fichiers écrits en C# à associer à des objets 3D dans la scène permettant de leur ajouter un comportement. Sur la figure ci-dessus, nous pouvons voir 11 scripts dans la fenêtre de projet, qui doivent alors se trouver dans le dossier spécifique de nom *Assets* afin de compiler.
- La fenêtre d'inspecteur : Elle permet de modifier directement les propriétés d'un objet 3D dans la scène. Les modifications de propriétés d'objet 3D dans cette fenêtre sont alors immédiatement reflétées dans la scène. Dans notre cas, la fenêtre d'inspecteur correspond aux propriétés de l'objet nommé *Loaders* : d'une part sa transformée (coordonnées de position, rotation et taille) inutile ici et d'autre part les valeurs des champs publics du script associé nommé *Main*. Ces valeurs peuvent être directement modifiées par l'utilisateur afin

de changer le résultat de l'exécution de notre application, c'est à dire de la construction de notre modèle.

L'interface utilisateur est également composée de 3 fenêtres cachées sur la figure ci-dessus :

- La fenêtre de gestion des « packages » : Elle permet de gérer l'installation, la mise et à jour et la suppression des « packages » de notre projet. Elle n'est absolument pas importante pour l'utilisateur.
- La fenêtre de jeu : Elle correspond au résultat de l'exécution de notre application. Dans notre cas, cette fenêtre permettra d'incarner un avatar qui peut se déplacer et sauter dans la scène correspondant à notre modèle 3D CityGML.
- La console : Elle affiche une valeur, chaîne de caractères, un résultat de calcul, etc... dès qu'une instruction de type *Debug.Log(...)* est rencontrée lors de l'exécution d'un script. Elle est utile pour identifier et corriger des erreurs lors de la construction de notre modèle mais peu importante pour l'utilisateur.

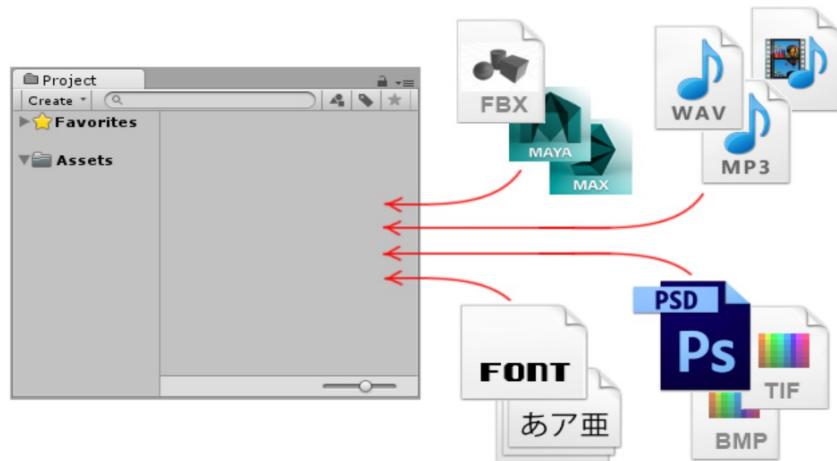


Fig. 25 : Some of the asset types that can be imported into Unity

2/ Actions de l'utilisateur nécessaires à la construction de notre modèle

Pour commencer, l'utilisateur doit s'assurer d'avoir importé les données OSM correspondant aux bâtiments du Liechtenstein dans l'éditeur de Unity (pour ce faire, voir section 3/ de la partie II/). Plus précisément, les 5 fichiers de type OSM-XML en question doivent se trouver dans le répertoire dont le chemin relatif est : *Assets/Resources/XmlFiles*, comme spécifié en sous-section f) de la section 3/ de la partie II/.

a) La fenêtre d'inspecteur de l'objet *Loaders*

L'utilisateur peut modifier les valeurs des champs publics du script *Main* dans la fenêtre d'inspecteur associée à l'objet *Loaders* comme sur la figure suivante :

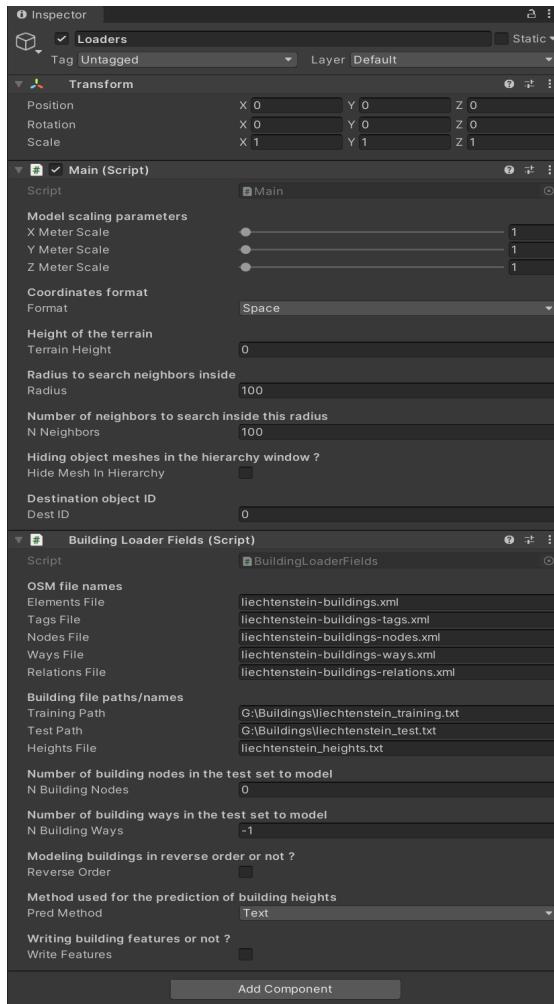


Fig. 26 : Affichage des valeurs des champs publics du script *Main* dans la fenêtre d'inspecteur associée à l'objet *Loaders*, avec possibilité de modification par l'utilisateur

Dans la fenêtre d'inspecteur de l'objet *Loaders*, l'utilisateur peut modifier les valeurs des champs publics suivants :

- *X/Y/Z Meter Scale* : Ces champs correspondent, comme le nom de l'en-tête dans la fenêtre d'inspecteur l'indique, aux paramètres de mise en échelle de notre modèle 3D dans la scène selon l'axe *x*, *y* ou *z*. Par exemple, si la valeur de ces 3 champs est égale à 0.5 (resp. 2), alors notre modèle 3D sera 2 fois moins (resp. plus) grand que dans la réalité : les bâtiments, le terrain ainsi que tout autre objet urbain verront leur taille réduite de moitié (resp. doublée). Sur la figure ci-dessus, la valeur de ces 3 champs est égale à 1 donc notre modèle sera exactement cohérent avec la réalité : toutes les dimensions seront exprimées en mètres.
- *Format* : Ce champ correspond au format d'affichage des coordonnées de notre avatar en haut à gauche dans la fenêtre de jeu. La valeur de ce champ correspond à l'une des 3 valeurs littérales de l'énumération *Coordinates Format* :
 1. *Space* : les coordonnées affichées correspondent aux coordonnées *x*, *y* et *z* de notre avatar par rapport à celles du centre du terrain.
 2. *Lat Lon Deg* : les coordonnées affichées correspondent à la latitude et la longitude de notre avatar exprimées en degrés avec une précision de type *double*.
 3. *Lat Lon DMS* : les coordonnées affichées correspondent à la latitude et la longitude de notre avatar exprimées en degrés, minutes et secondes.
- *Terrain Height* : Ce champ correspond à la hauteur du terrain par rapport au système de

coordonnées 3D de la scène. La coordonnée *y* de la position de chaque bâtiment est ainsi toujours égale à la valeur de ce champ. Notez que la coordonnée *y* de la position de l'objet *FPSController* qui va faire apparaître notre avatar devrait toujours être au moins supérieure à la valeur de ce champ.

- *Radius* : Ce champ correspond grossièrement au rayon en mètres du périmètre autour d'un bâtiment donné à l'intérieur duquel rechercher ses bâtiments voisins. Plus précisément, la valeur de ce champ correspond au rayon du cercle dont le centre coïncide avec le barycentre du polygone (en 2D) associé à la forme d'un bâtiment donné.
- *N Neighbors* : Ce champ entier correspond au nombre de bâtiments voisins à rechercher dans un périmètre autour d'un bâtiment donné, de rayon égal à la valeur du champ *radius*.
- *Hide Mesh In Hierarchy* : Ce champ booléen détermine s'il faut cacher les entités « *GameObject* » correspondant à l'objet de type *ProBuilderMesh* des bâtiments dans la fenêtre de hiérarchie ou non. Si la case correspondant à ce champ est cochée comme dans le cas de la figure ci-dessus, il y aura 2 fois moins d'entités « *GameObject* » dans la fenêtre de hiérarchie ce qui améliore la lisibilité et potentiellement les performances.
- *Dest ID* : Ce champ correspond à l'identifiant d'un bâtiment destinataire existant que l'utilisateur peut entrer afin d'afficher (en bas à gauche dans la fenêtre de jeu) la distance restante à parcourir par son avatar pour rejoindre ce bâtiment.
- *Elements/Nodes/Ways/Relations/Tags File* : Ces champs correspondent aux noms des fichiers OSM à importer se trouvant dans le répertoire de chemin relatif *Assets/Resources/XmlFiles*. Le premier champ correspond à tous les éléments OSM à importer. Les second, 3ème et 4ème champs correspondent à tous les sous-nœuds, sous-chemins et sous-relations de ces éléments respectivement. Enfin, le dernier champ correspond à tous les éléments OSM contenant un attribut sur leur hauteur ou un autre attribut relatif aux bâtiments. Sur la figure ci-dessus, les noms de fichier entrés correspondent aux bâtiments du Liechtenstein.
- *Training/Test Path et Heights File* : Les 2 premiers champs correspondent aux chemins des fichiers texte dans lesquels écrire les attributs géométriques et cadastres des bâtiments se trouvant dans l'ensemble d'entraînement (ou fichier de nom *TagsFile*) et de test (ou fichier de nom *AllPath*) respectivement. Le dernier champ correspond au nom du fichier depuis lequel importer les hauteurs des bâtiments de l'ensemble de test. Ce fichier doit se trouver dans le répertoire de chemin relatif : *Assets/Resources/TxtFiles*. Sur la figure ci-dessus, les noms de chemin/fichier entrés correspondent aux bâtiments du Liechtenstein.
- *N Building Nodes* : Ce champ entier correspond au nombre de bâtiments associés à un nœud OSM à modéliser. Si la valeur de ce champ est négative, tous les bâtiments associés à un nœud OSM se trouvant dans le fichier XML de nom *liechtenstein-buildings.xml* seront modélisés. Sur la figure ci-dessus, la valeur de ce champ est nulle donc aucun bâtiment du Liechtenstein associé à un nœud OSM ne sera modélisé.
- *N Building Ways* : Ce champ entier correspond au nombre de bâtiments associés à un chemin OSM à modéliser. Si la valeur de ce champ est négative, tous les bâtiments associés à un chemin OSM se trouvant dans le fichier XML de nom *liechtenstein-buildings.xml* seront modélisés. Sur la figure ci-dessus, la valeur de ce champ est égale à 1000 donc les 1000 premiers bâtiments du Liechtenstein associés à un chemin OSM seront modélisés.
- *Reverse Order* : Ce champ booléen détermine s'il faut modéliser les bâtiments dans l'ordre inverse ou non (dans l'ordre d'apparition dans le fichier source des éléments OSM associés). Sur la figure ci-dessus, les bâtiments du Liechtenstein associés à un chemin OSM seront modélisés dans l'ordre.
- *Pred Method* : Ce champ correspond à la méthode de prédiction à utiliser pour prédire la hauteur initiale des bâtiments. La valeur de ce champ correspond à l'une des 3 valeurs

littérales de l'énumération *PredictionMethod* :

1. *None* : Aucune méthode de prédiction ne sera utilisée, c'est à dire que les bâtiments ne possédant pas d'attribut sur leur hauteur réelle auront tous une même hauteur arbitraire de $3*nFloors$ mètres (La valeur du facteur de 3m, qui peut-être changée à tout moment par l'utilisateur, correspond à la hauteur approximative d'un étage) dans notre modèle.
 2. *Text* : Les bâtiments ne possédant pas d'attribut sur leur hauteur réelle se verront attribuer une hauteur importée depuis le fichier texte de nom *Heights File*, qui doit se trouver dans le répertoire de chemin relatif : *Assets/Resources/TxtFiles*. Dans notre cas, les hauteurs seront importées depuis le fichier de nom *liechtenstein_heights.txt*. Par exemple, les valeurs des hauteurs importées pourraient être prédites par un modèle de régression comme en section 1/ de la partie III/.
 3. *Picture* : Les bâtiments ne possédant pas d'attribut sur leur hauteur réelle se verront attribuer une hauteur prédite à partir d'une photographie de l'une de leurs façades, filtrée par l'opérateur de Sobel de préférence. Les photographies de chaque bâtiment doivent se trouver dans le répertoire de chemin relatif : *Assets/Resources/Pictures* et le nom des fichiers associés doit correspondre, comme spécifié en section 2/ de la partie III/. Si la photographie d'un bâtiment n'est pas trouvée dans ce répertoire, sa hauteur aura une valeur arbitraire de $3*nFloors$ mètres.
- *Write Features* : Ce champ booléen détermine s'il faut écrire les attributs géométriques et cadastres de chaque bâtiment (valeur *true*) ou non (valeur *false*). Ces attributs seront écrits dans un fichier texte dont le chemin absolu est *TrainingPath* et *TestPath* pour les bâtiments de l'ensemble d'entraînement et de test respectivement. Dans notre cas, on a :
- TrainingPath=G:\Buildings\liechtenstein_training.txt* et
TestPath=G:\Buildings\liechtenstein_test.txt.

b) Construction de notre modèle sans les prédictions sur les hauteurs des bâtiments

Pour cette étape, l'utilisateur doit s'assurer que les chemins et noms de fichiers entrés dans les champs correspondant aux 2 en-têtes *OSM file names* et *Building file names* soient bien valides (seule la valeur du champ de nom *Heights File* n'est pas obligatoire pour cette étape). Sur la figure ci-dessous, c'est d'ailleurs le cas pour les bâtiments du Liechtenstein.

Ensuite, l'utilisateur doit sélectionner la valeur littérale *None* pour le champ de nom *Pred Method* et cocher la case correspondant au champ de nom *Write Features*, puis sauver l'objet *Loaders* dans la scène en appuyant sur CTRL-S (chaque fois que les propriétés d'un objet sont modifiées dans la fenêtre d'inspecteur, il est nécessaire de sauvegarder la scène). Les actions de l'utilisateur pour cette étape sont illustrées dans la figure suivante :

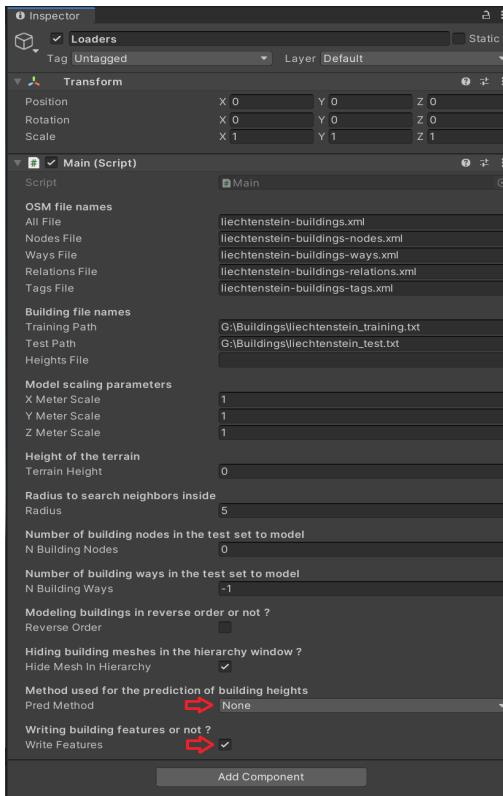


Fig. 27 : Affichage des valeurs des champs de la fenêtre d'inspecteur de l'objet *Loaders* lors de l'étape de construction de notre modèle sans les prédictions sur les hauteurs des bâtiments

Pour démarrer la construction de notre modèle, il suffit de cliquer sur le bouton « Play » dans l'interface de Unity comme illustré sur la figure suivante :

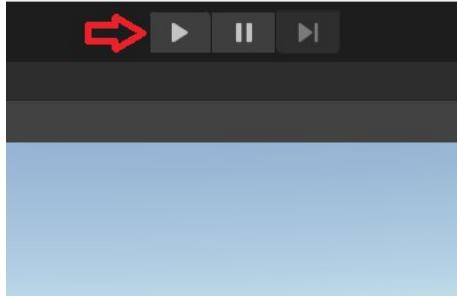


Fig. 28 : Bouton « Play » se trouvant juste au-dessus de la scène dans l'interface de Unity

Après quelques minutes, notre modèle devrait apparaître dans la scène s'il n'y a pas eu d'erreurs de compilation. Le modèle obtenu devrait être similaire à celui de la figure suivante :

Cependant, nous constatons que la très grande majorité des bâtiments du Liechtenstein dans notre modèle (ceux dont on ne connaissait ni la hauteur réelle ni le nombre d'étages) ont tous une même hauteur arbitraire de 3m. On en vient ainsi à l'étape suivante où il est nécessaire de prédire la hauteur des bâtiments à partir des attributs géométriques et cadastres écrits à cette étape, en utilisant notre modèle de régression (pour ce faire, voir sous-section c) de la section 1/ de la partie III/). Une fois que le fichier contenant les hauteurs prédites pour chaque bâtiment du Liechtenstein de l'ensemble de test a été généré, il faut l'importer sur Unity dans le répertoire de chemin relatif : *Assets/Resources/TxtFiles* (cette étape d'importation est spécifiée en sous-section d) de la section 1/ de la partie III/). Notez que l'importation n'est pas nécessaire si le chemin de sortie du fichier contenant les valeurs des hauteurs prédites pointe directement vers ce répertoire.

c) Construction de notre modèle avec les prédictions sur les hauteurs des bâtiments

Pour cette étape, l'utilisateur doit sélectionner la valeur littérale *Text* pour le champ de nom *Pred Method*, décocher la case correspondant au champ de nom *Write Features* (afin de réduire le temps de construction de notre modèle) et ajouter le nom du fichier contenant les valeurs des hauteurs prédites (dans le champ de nom *Heights File*). Ensuite, il faut sauver l'objet *Loaders* dans la scène en appuyant sur CTRL-S. Les actions de l'utilisateur pour cette étape sont illustrées dans la figure suivante :

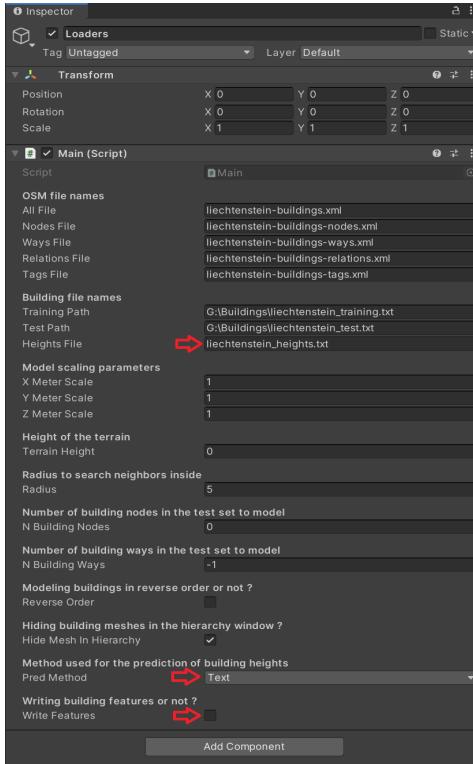


Fig. 29 : Affichage des valeurs des champs de la fenêtre d'inspecteur de l'objet *Loaders* lors de l'étape de construction de notre modèle avec les prédictions sur les hauteurs des bâtiments

Maintenant, il faut reconstruire notre modèle en cliquant de nouveau sur le bouton « Play ». Après quelques minutes, notre nouveau modèle devrait apparaître dans la scène s'il n'y a pas eu d'erreurs de compilation. Le modèle obtenu devrait être similaire à celui de la figure suivante :

Cette fois-ci, les bâtiments du Liechtenstein correspondant à l'ensemble de test auront approximativement toutes une hauteur différente et cohérente avec la réalité (voir partie V/ pour l'évaluation de la cohérence de notre nouveau modèle avec la réalité).

3/ Actions de l'utilisateur nécessaires à la modification de notre modèle

a) La fenêtre d'inspecteur de l'objet associé aux détails d'un bâtiment

Une fois notre modèle construit, l'utilisateur peut directement modifier les attributs d'un bâtiment (en particulier sa hauteur) dans la fenêtre d'inspecteur de l'objet associé aux détails de ce bâtiment. Un tel objet a comme nom « *Building Details ID=...* » et a pour parent l'objet conteneur de nom *Building Details* dans la fenêtre de hiérarchie. La modification des attributs d'un bâtiment par l'utilisateur est illustrée par la figure suivante :

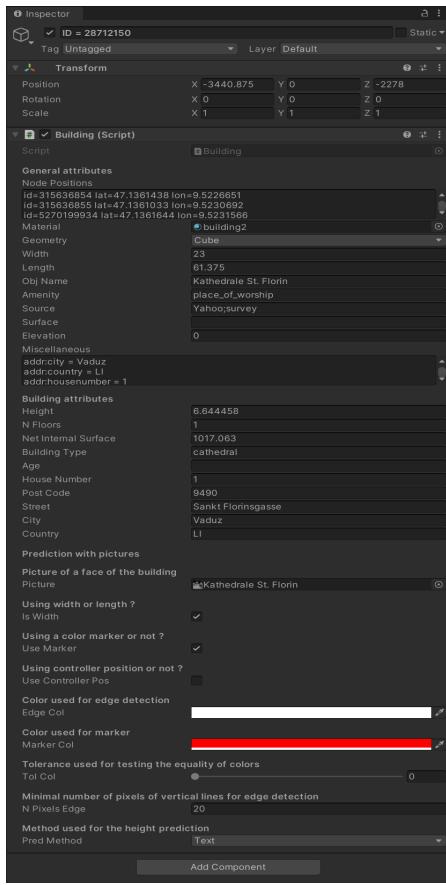


Fig. 30 : Affichage des attributs du bâtiment d'identifiant 28712150, c'est à dire de la cathédrale de St-Florin, dans la fenêtre d'inspecteur de l'objet associé

En fait, l'utilisateur modifie les attributs d'un bâtiment en même temps que les valeurs des champs publics de la classe *CityObject* et *Building*. Dans la classe *CityObject*, on a les attributs généraux suivants :

- *Node Positions* : Correspond aux valeurs de latitude et longitude de chacun des sous-nœuds d'un chemin ou d'une relation OSM associé(e) à un bâtiment (ou à la latitude et longitude d'un nœud OSM associé à un bâtiment). L'identifiant permet d'identifier à quel nœud correspond chaque paire de latitude et longitude. Notez que les modifications de ce champ par l'utilisateur ne seront jamais validées par notre modèle.
- *Material* : La texture utilisée pour le modèle des bâtiments.
- *Geometry* : Correspond à la forme géométrique primitive qui est utilisée pour modéliser les nœuds associés à des bâtiments.
- *Width* : La largeur en mètres du bâtiment. Sa modification par l'utilisateur ne sera validée que pour le cas des nœuds OSM associés à un bâtiment (et si la valeur entrée est strictement positive).
- *Length* : La longueur en mètres du bâtiment. Sa modification par l'utilisateur ne sera validée que pour le cas des nœuds OSM associés à un bâtiment (et si la valeur entrée est strictement positive).
- *Obj Name* : Le nom du bâtiment, qui peut être modifié à tout moment par l'utilisateur.
- *Amenity* : La fonction du bâtiment ou son usage dans la vie courante.
- *Source* : La source de toute information ou marque associée à un bâtiment.
- *Surface* : Le revêtement ou matériau de construction utilisé pour un bâtiment.
- *Elevation* : L'altitude à laquelle se trouve un bâtiment, exprimée en mètres.
- *Miscellaneous* : Tout attribut divers non cité précédemment qui est relatif à un bâtiment.

Dans la classe *Building*, on a les attributs relatifs aux bâtiment suivants:

- *Height* : La hauteur réelle, prédite ou arbitraire en mètres du bâtiment (la hauteur arbitraire correspond en fait plutôt à la hauteur d'un étage du bâtiment). Sa modification par l'utilisateur ne sera validée que dans le cas où la valeur littérale actuelle du champ *Pred Method* est *None* (et si la valeur entrée est strictement positive).
- *NFloors* : Le nombre d'étages du bâtiment. Sa modification par l'utilisateur ne change la hauteur que dans le cas où la valeur littérale actuelle du champ *Pred Method* est *None* (et si le nombre d'étages entré n'est pas négatif ou nul). En effet, la hauteur arbitraire d'un bâtiment est de *Height***NFloors* mètres dans ce cas.
- *Net Internal Surface* : La surface interne nette du bâtiment, exprimée en mètres au carré.
- *Building Type* : Le type du bâtiment, qui peut être modifié à tout moment par l'utilisateur.
- *Age* : L'âge du bâtiment, compté depuis le jour où il a été construit.
- *Post Code/Street/City/Country* : L'adresse du bâtiment (code postal, rue, ville et pays)
- *Picture* : Une photographie de la façade d'un bâtiment qui sera utilisée pour la prédiction de sa hauteur dans le cas où la valeur littérale actuelle du champ *Pred Method* est *Picture*. La photographie devrait avoir été filtrée par l'opérateur de Sobel au préalable. La photographie utilisée peut être modifiée à tout moment par l'utilisateur.
- *Is Width* : Un champ booléen qui indique s'il faut utiliser la longueur ou la largeur d'un bâtiment pour la prédiction de sa hauteur avec la photographie actuelle. Sa modification par l'utilisateur ne sera effective que dans le cas où la valeur littérale actuelle du champ *Pred Method* est *Picture*.
- *Use Marker* : Un champ booléen qui indique s'il faut utiliser la méthode de prédiction de la hauteur d'un bâtiment avec la photographie actuelle par marquage ou non. Sa modification par l'utilisateur ne sera effective que dans le cas où la valeur littérale actuelle du champ *Pred Method* est *Picture*.
- *Use Controller Pos* : Un champ booléen qui indique s'il faut utiliser la position actuelle de notre avatar pour estimer la hauteur d'un bâtiment avec la photographie. Typiquement, l'angle de point de vue de notre avatar qui observe le modèle de bâtiment dans la scène doit correspondre avec l'angle de prise du bâtiment sur la photographie.
- *Edge Col* : La couleur qui sera utilisée pour l'identification des lignes verticales sur la photographie actuelle, lors de la prédiction de la hauteur d'un bâtiment par détection des contours. Sa modification par l'utilisateur ne sera effective que dans le cas où la valeur littérale actuelle du champ *Pred Method* est *Picture* et que le booléen *UseMarker* a pour valeur *false*.
- *Marker Col* : La couleur qui est utilisée pour le marquage des dimensions d'un bâtiment sur la photographie actuelle. Sa modification par l'utilisateur ne sera effective que dans le cas où la valeur littérale actuelle du champ *Pred Method* est *Picture* et que le booléen *UseMarker* a pour valeur *true*.
- *Tol Col* : La tolérance utilisée pour la vérification de l'égalité de la couleur d'un pixel de la photographie actuelle avec la couleur de marquage *Marker Col* ou la couleur d'une ligne verticale *Edge Col*. La valeur entrée doit être comprise entre 0 et 1 inclus.
- *N Pixels Edge* : Le nombre minimal de pixels utilisé pour l'identification des lignes verticales sur la photographie actuelle, lors de la prédiction de la hauteur d'un bâtiment par détection des contours. Sa modification par l'utilisateur ne sera effective que dans le cas où la valeur littérale actuelle du champ *Pred Method* est *Picture* et que le booléen *UseMarker* a pour valeur *false*.
- *Pred Method* : Indique la méthode de prédiction de la hauteur d'un bâtiment actuellement utilisée (*None*, *Text* ou *Picture*). Elle peut être modifiée à tout moment par l'utilisateur et a pour effet de changer la hauteur d'un bâtiment.

b) Exemple de modification de la hauteur d'un bâtiment du Liechtenstein

Reprenons l'exemple de la cathédrale de St-Florin en section 2/ de la partie III/. Si nous recherchons l'objet associé de nom : *Building Details*=28712150 dans la fenêtre de hiérarchie une fois notre modèle construit, nous obtenons la fenêtre d'inspecteur suivante :

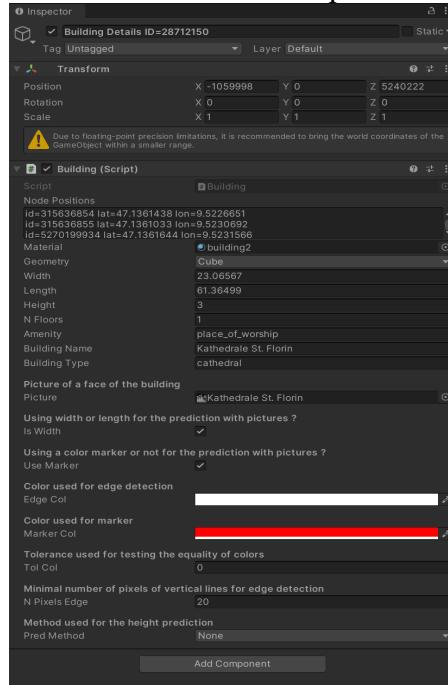


Fig. 31 : Affichage des attributs du bâtiment d'identifiant 28712150 correspondant à la cathédrale de St-Florin au Liechtenstein dans la fenêtre d'inspecteur de l'objet associé

On voit que la hauteur affichée ici est une hauteur arbitraire de 3m. Maintenant, si nous modifions la valeur littérale du champ *Pred Method* de *None* à *Picture*, nous constaterons que la hauteur du bâtiment sera effectivement mise à jour comme le montre la figure suivante :

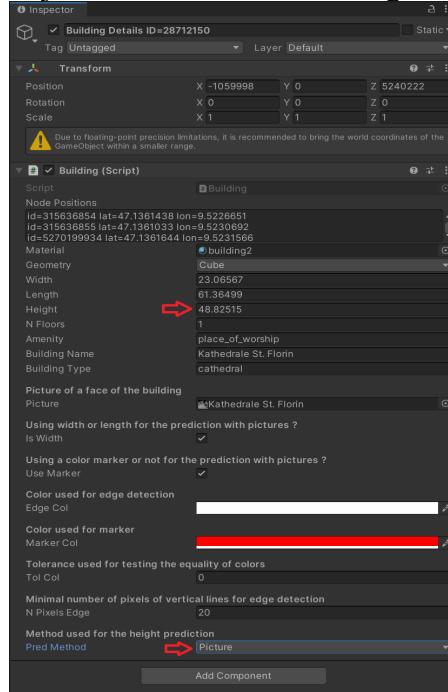


Fig. 32 : Mise à jour de la hauteur de la cathédrale de St-Florin au Liechtenstein à une valeur cohérente avec la réalité dans la fenêtre d'inspecteur de l'objet associé

Cet exemple montre que la hauteur d'un bâtiment peut-être mise à jour lors de la modification de certains de ses attributs (en particulier la valeur du champ *Pred Method*) dans la fenêtre d'inspecteur de l'objet associé.

4/ Diagramme de classes de notre modèle

Voici le diagramme de classes actuel de notre modèle associé aux scripts (à l'exception des scripts de nom *NeuralNetwork.cs* et *MultiPolygon.cs*) se trouvant dans le dossier *Assets* :

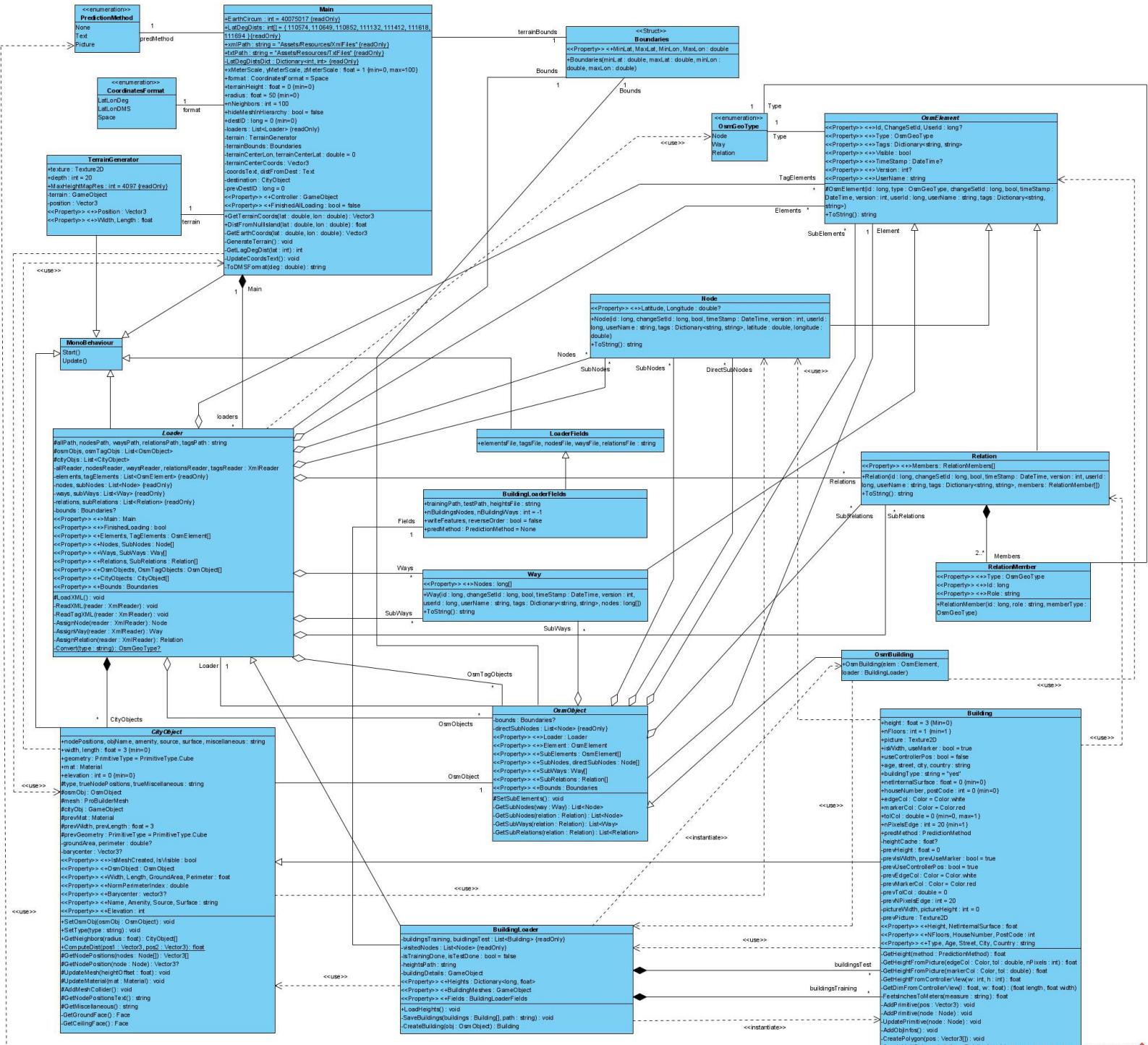


Fig. 33 : Diagramme de classes de notre modèle 3D

5/ Responsabilités de chaque classe du diagramme

Les responsabilités de chaque classe du diagramme sont les suivantes :

- Classe prédéfinie *MonoBehaviour* : Classe parente de toutes les classes définies dans un script de Unity et associées à une entité « *GameObject* ». Chaque classe qui en dérive peut redéfinir les méthodes *Start()* et *Update()* afin d'ajouter un comportement à l'entité « *GameObject* » associée. De plus, ces classes dérivées ne peuvent pas être instanciées : les instances de *MonoBehaviour* doivent être créées par un appel à la méthode prédéfinie *AddComponent()*.
- Énumération *PredictionMethod* : Définit la méthode de prédiction de la hauteur de bâtiments à utiliser lors de la construction du modèle.
- Énumération *CoordinatesFormat* : Définit le format utilisé pour afficher les coordonnées de notre avatar dans la fenêtre de jeu.
- Énumération *OsmGeoType* : Définit le type d'un élément OSM associé à un objet de type *OsmElement*.
- Structure *Boundaries* : Définit les limites en terme de latitude et longitude min/max d'un ensemble d'éléments OSM complets (objets de type *OsmObject*), à partir des latitudes et longitudes des sous-nœuds.
- Classe *LoaderFields* dérivée de *MonoBehaviour* : Contient les champs publics généraux de l'objet *Loaders* dans la scène. Ces champs permettent de gérer les entrées de l'utilisateur avant la construction de notre modèle.
- Classe *BuildingLoaderFields* dérivée de *LoaderFields* : Contient les champs publics de l'objet *Loaders* dans la scène qui sont relatifs aux bâtiments. Ces champs permettent de gérer les entrées de l'utilisateur avant la construction de notre modèle.
- Classe *Main* dérivée de *MonoBehaviour* :
 1. Teste si les entrées de l'utilisateur dans la fenêtre d'inspecteur de l'objet *Loaders* sont valides.
 2. Crée un objet de type *BuildingLoader* pour la construction des bâtiments, et assigne à cet objet les chemins de fichiers de type OSM-XML ou texte entrés par l'utilisateur.
 3. Calcule la longueur et la largeur du terrain à partir d'un objet de type *Boundaries*, ainsi que la position de son coin inférieur gauche.
 4. Crée un objet de type *TerrainGenerator* à partir des données de terrain calculées.
 5. Calcule les coordonnées réelles en mètres d'un point du terrain donné par sa latitude et longitude, en fonction de l'origine se trouvant au centre du terrain.
 6. Calcule les coordonnées réelles en mètres d'un point terrestre donné par sa latitude et longitude, en fonction de l'origine se trouvant au point de latitude et longitude nulles. Ce point particulier s'appelle « *Null Island* », île imaginaire se trouvant à intersection de l'équateur et du méridien de Greenwich (pour plus de détails, voir le lien suivant : https://fr.wikipedia.org/wiki/Null_Island).
 7. Calcule la distance d'un point terrestre à « *Null Island* ».
 8. Met à jour les coordonnées actuelles de notre avatar dans la fenêtre de jeu.
- Classe abstraite *Loader* dérivée de *MonoBehaviour* :
 1. Importe les données OSM depuis 5 fichiers de type OSM-XML se trouvant dans le répertoire dont le chemin relatif est : *Assets/Resources/XmlFiles*.
 2. Initialise les listes d'éléments OSM de noms suivants : *elements*, *tagElements*, *nodes*, *ways*, *relations*, *subNodes*, *subWays*, *subRelations* à partir des données importées.
 3. Calcule les limites (objet de type *Boundaries*) de l'ensemble des nœuds et sous-nœuds du fichier source, à partir de leurs latitudes et longitudes.
- Classe *TerrainGenerator* dérivée de *MonoBehaviour* :

1. Génère le terrain à partir de sa largeur, longueur et position, et y applique une texture à l'initialisation.
 2. Gère les changements de propriétés du terrain dans la fenêtre d'inspecteur associée à un objet de type *TerrainGenerator*.
- Classe abstraite *CityObject* dérivée de *MonoBehaviour* :
 1. Gère les propriétés géométriques d'un objet urbain, notamment : sa géométrie (objet de type prédéfini *ProBuilderMesh*), texture, largeur, longueur, aire au sol, son périmètre et barycentre, etc... ainsi que qualitatives : son nom, sa fonction (clé de tag « amenity »), son altitude, etc... ces propriétés géométriques sont fixes pour un objet urbain donné et ne devraient jamais être modifiées.
 2. Calcule les coordonnées réelles d'un ensemble de nœuds OSM à partir de leur latitude et longitude.
 3. Trouve le nombre d'objets urbains voisins du même type (par exemple des bâtiments) dans un périmètre d'un certain rayon autour d'un objet urbain (par exemple un bâtiment) donné.
 4. Trouve la face (objet de type prédéfini *Face*) correspondant au sol et au plafond d'un bâtiment.
 - Classe abstraite *OsmElement* : Définit un élément OSM simple
 - Classe *Node* dérivée de *OsmElement* : Définit un nœud OSM simple
 - Classe *Way* dérivée de *OsmElement* : Définit un chemin OSM simple
 - Classe *Relation* dérivée de *OsmElement* : Définit une relation OSM simple
 - Classe *RelationMember* : Définit un membre de relation OSM simple
 - Classe abstraite *OsmObject* : Définit un élément OSM complet de la façon suivante :
 1. En trouvant les sous-nœuds (resp. sous-chemins et sous-relations) d'un élément OSM simple (propriété de nom *Element* et de type *OsmElement*) à partir de leur identifiant unique dans la liste *subNodes* (resp. *subWays* et *subRelations*) initialisée dans la classe *Loader*.
 2. En calculant les limites (objet de type *Boundaries*) à partir des latitudes et longitudes des sous-nœuds de cet élément OSM complet.
 - Classe *OsmBuilding* dérivée de *OsmObject* : Définit un élément OSM complet associé à un bâtiment.
 - Classe *BuildingLoader* dérivée de *Loader* :
 1. Initialise 2 listes de bâtiments (objets de type *Building*) de noms *buildingsTraining* et *buildingsTest* correspondant à l'ensemble d'entraînement et de test de notre modèle de régression respectivement.
 2. Crée un bâtiment (objet de type *Building*) à partir d'une instance de type *OsmObject* et l'ajoute à l'une des 2 listes initialisées précédemment. Plus précisément, le bâtiment est ajouté à l'ensemble d'entraînement si l'instance de type *OsmObject* associée se trouve dans la liste *osmTagObjs* et l'élément OSM associé contient un attribut sur sa hauteur. De même, le bâtiment est ajouté à l'ensemble de test si l'instance de type *OsmObject* associée se trouve dans la liste *osmObjs* et l'élément OSM associé ne contient pas d'attribut sur sa hauteur.
 3. Écrit les attributs géométriques et cadastres des bâtiments (objets de type *Building*) de l'ensemble d'entraînement et de test dans 2 fichiers textes différents se trouvant dans le répertoire de chemin relatif : *Assets/Resources/TxtFiles*.
 4. Charge le fichier contenant les valeurs des hauteurs prédictes pour chaque bâtiment de l'ensemble de test.
 - Classe *Building* dérivée de *CityObject* :
 1. Gère les attributs cadastres d'un bâtiment (les attributs géométriques étant gérés par

- la classe parente *CityObject*, à l'exception de la hauteur), notamment : le nombre d'étages, la surface interne nette, l'âge, le type, etc... met à jour ces propriétés s'il y a un changement dans la fenêtre d'inspecteur associée à un objet de type *Building*.
2. Gère la hauteur d'un bâtiment : on lui assigne soit une valeur réelle si l'élément OSM associé contient un attribut sur sa hauteur, soit une valeur prédite à partir de la méthode de prédiction sélectionnée à l'initialisation (voir partie III/ pour plus de détails), soit une valeur arbitraire égale à $3*nFloors$ mètres (si les 2 cas précédents échouent).
 3. Définit des méthodes de prédiction de la hauteur à partir de photographies de bâtiments, si besoin d'estimer la hauteur à partir de telles méthodes.
 4. Crée le polygone en 3D associé à la forme d'un bâtiment (objet de type *ProBuilderMesh*) à partir d'un ensemble de nœuds OSM et d'une hauteur réelle, prédite ou arbitraire : on ajoute le bâtiment à notre modèle dans la scène. Si les dimensions d'un bâtiment sont modifiées par l'utilisateur, on met à jour l'objet de la scène associé à ce bâtiment.

V/ Évaluation et validation de notre modèle

Une fois notre modèle construit sur Unity, il est nécessaire de l'évaluer et de le valider à partir d'une source de données géographiques fiable. J'ai donc d'abord choisi d'évaluer notre modèle dans son ensemble en le comparant à une vue aérienne de la même zone sur Google Earth. Ensuite, j'ai choisi d'évaluer le modèle de 5 bâtiments célèbres du Liechtenstein en comparant leurs dimensions dans notre modèle d'une part et sur Google Maps/Earth d'autre part. Enfin, j'ai validé les résultats obtenus dans notre modèle en partie VI/.

Pour commencer, je présenterai la configuration de la fenêtre d'inspecteur de l'objet *Loaders* permettant une telle évaluation :

1/ Configuration de la fenêtre d'inspecteur de l'objet *Loaders* lors de la construction de notre modèle

Lors de la construction de notre modèle sans les prédictions sur les hauteurs des bâtiments (voir sous-section b) en section 2/ de la partie IV/), la fenêtre d'inspecteur de l'objet *Loaders* a la configuration suivante :

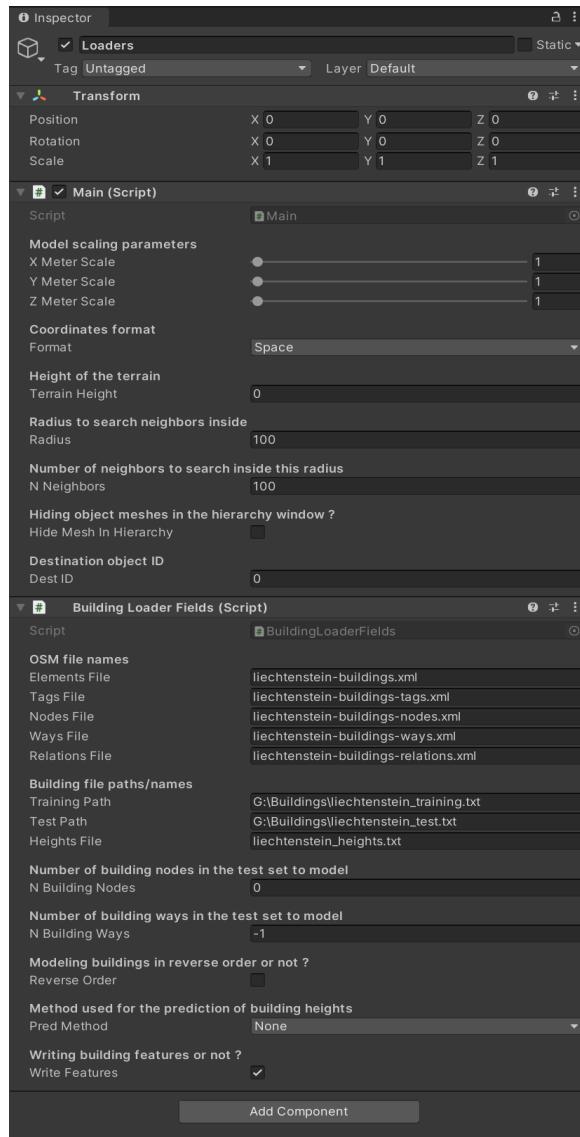


Fig. 34 : Configuration de la fenêtre d'inspecteur de l'objet *Loaders* lors de l'étape de construction de notre modèle sans les prédictions sur les hauteurs des bâtiments

Une fois notre modèle de régression construit (comme spécifié en sous-section c) en section 1/ de la partie III/, on reconstruit notre modèle sur Unity avec les prédictions obtenues sur les hauteurs des bâtiments (voir sous-section c) en section 2/ de la partie IV/). La fenêtre d'inspecteur de l'objet *Loaders* a cette fois-ci la configuration suivante :

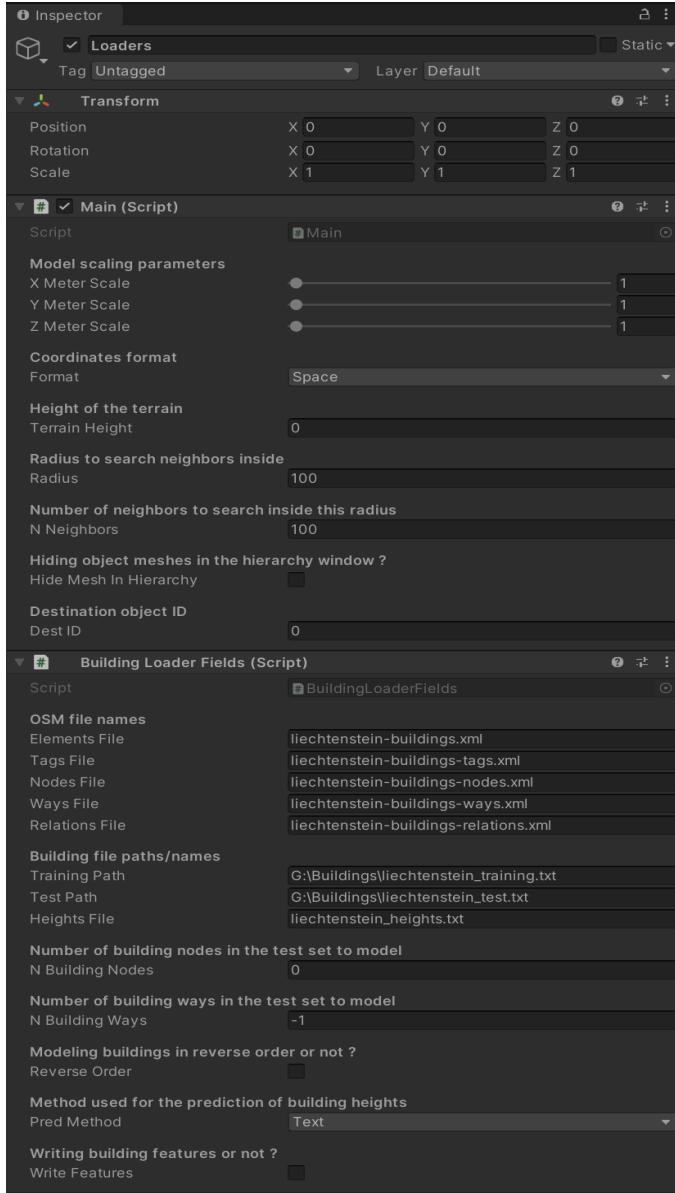


Fig. 35 : Configuration de la fenêtre d'inspecteur de l'objet *Loaders* lors de l'étape de construction de notre modèle avec les prédictions sur les hauteurs des bâtiments

2/ Comparaison de notre modèle du Liechtenstein dans son ensemble sur Google Earth

Pour comparer notre modèle dans son ensemble, il faut se positionner approximativement au même endroit dans notre modèle et sur Google Earth. Pour cela, on peut s'aider des valeurs de latitude et longitude affichées dans notre modèle (lors du déplacement de notre avatar dans la fenêtre de jeu) et les comparer à celles de Google Earth. Finalement, on bascule de nouveau dans la scène de l'interface de Unity puis on se positionne environ à la même altitude (457m) au-dessus du sol à cette latitude et longitude. La figure suivante correspond à une vue aérienne d'une partie de la ville de Vaduz au Liechtenstein sur Google Earth :



Fig. 36 : Vue aérienne d'une partie de la ville de Vaduz au Liechtenstein sur Google Earth

La vue aérienne de notre modèle équivalente à cette partie de la ville de Vaduz au Liechtenstein est la suivante :

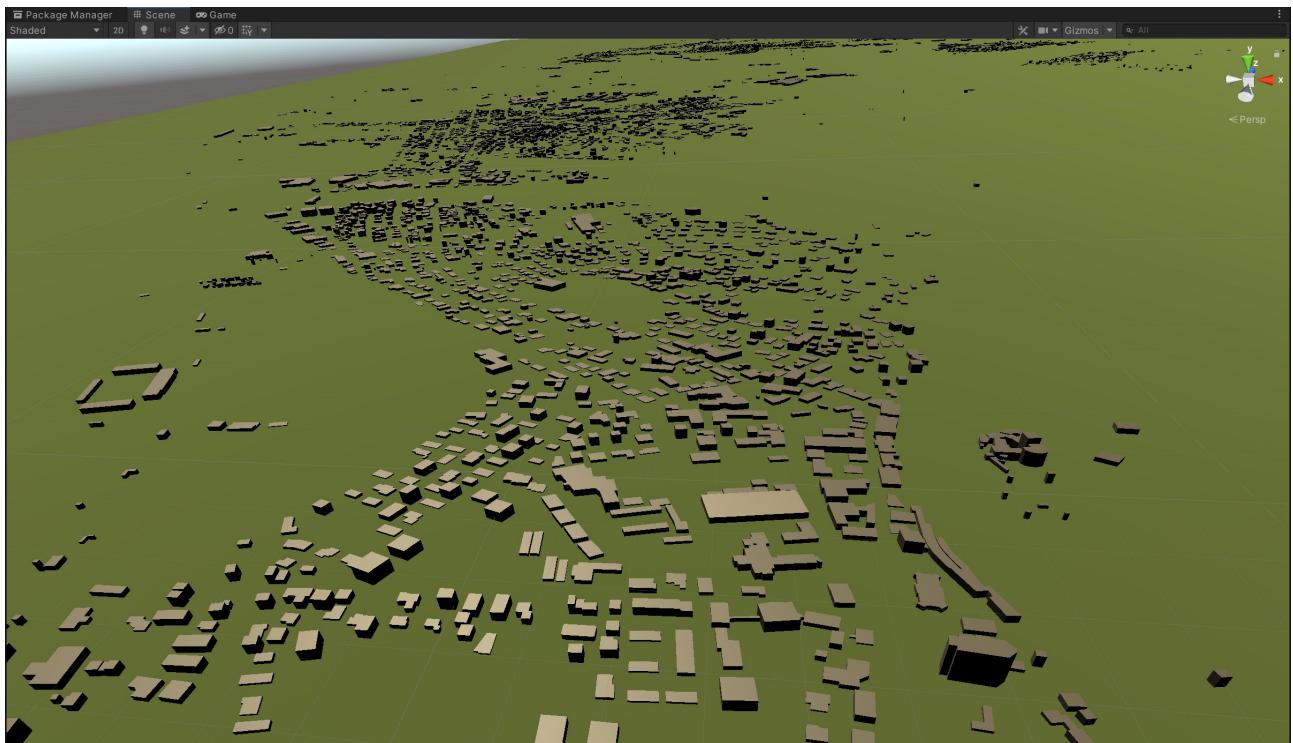


Fig. 37 : Vue aérienne équivalente à cette partie de la ville de Vaduz au Liechtenstein dans notre modèle

3/ Comparaison des modèles de 8 bâtiments du Liechtenstein sur Google Maps/Earth

Pour comparer les modèles de 8 bâtiments « célèbres » du Liechtenstein, j'ai utilisé le point de vue citadin d'une part et le point de vue aérien d'autre part pour chacun de ces bâtiments sur Google Maps et Google Earth respectivement. Pour le point de vue citadin, je me suis positionné au même endroit que sur Google Maps dans mon modèle à partir de la fenêtre de jeu, en utilisant les valeurs de latitude et longitude affichées ainsi qu'un GPS intégré (qui affiche la distance actuelle en mètres entre notre avatar et le barycentre du polygone associé à la forme du bâtiment destinataire). Pour le point de vue aérien, il me suffisait de naviguer dans la scène jusqu'à la position de notre avatar, à une altitude qui correspondait approximativement à celle de Google Earth.

a) La cathédrale de St-Florin

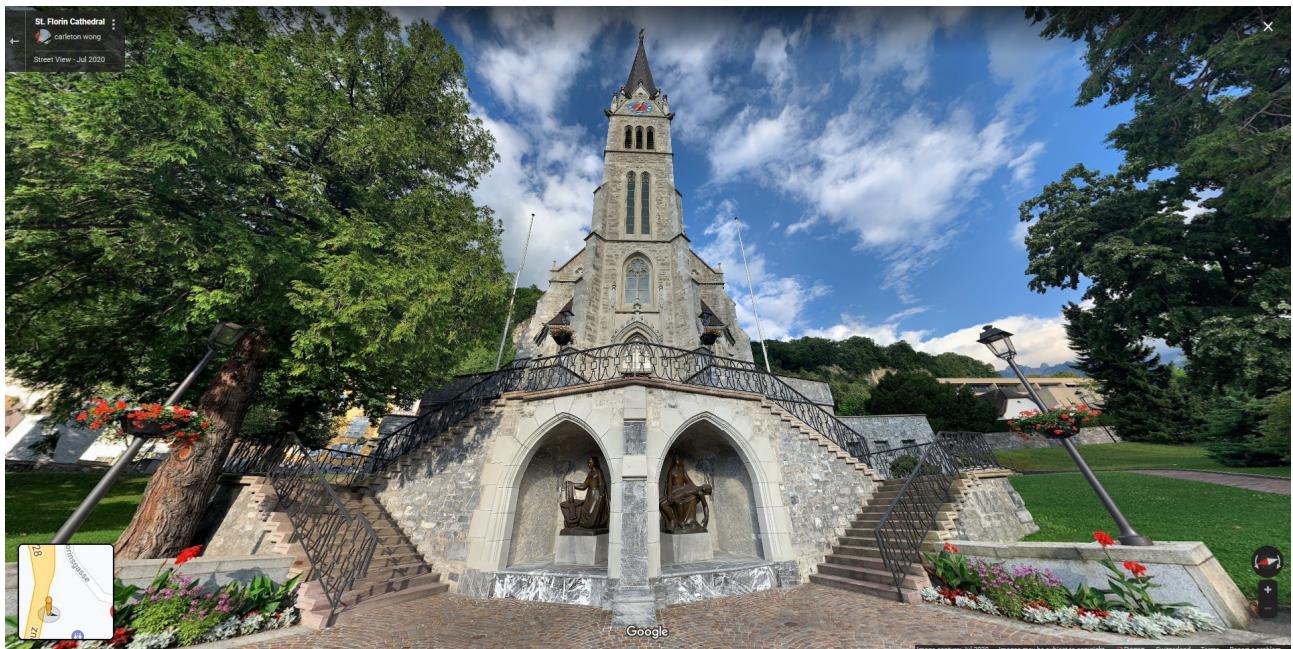


Fig. 38: Photographie de la cathédrale de St-Florin sur Google Maps

OSM source : <https://www.openstreetmap.org/way/28712150#map=19/47.13620/9.52308>

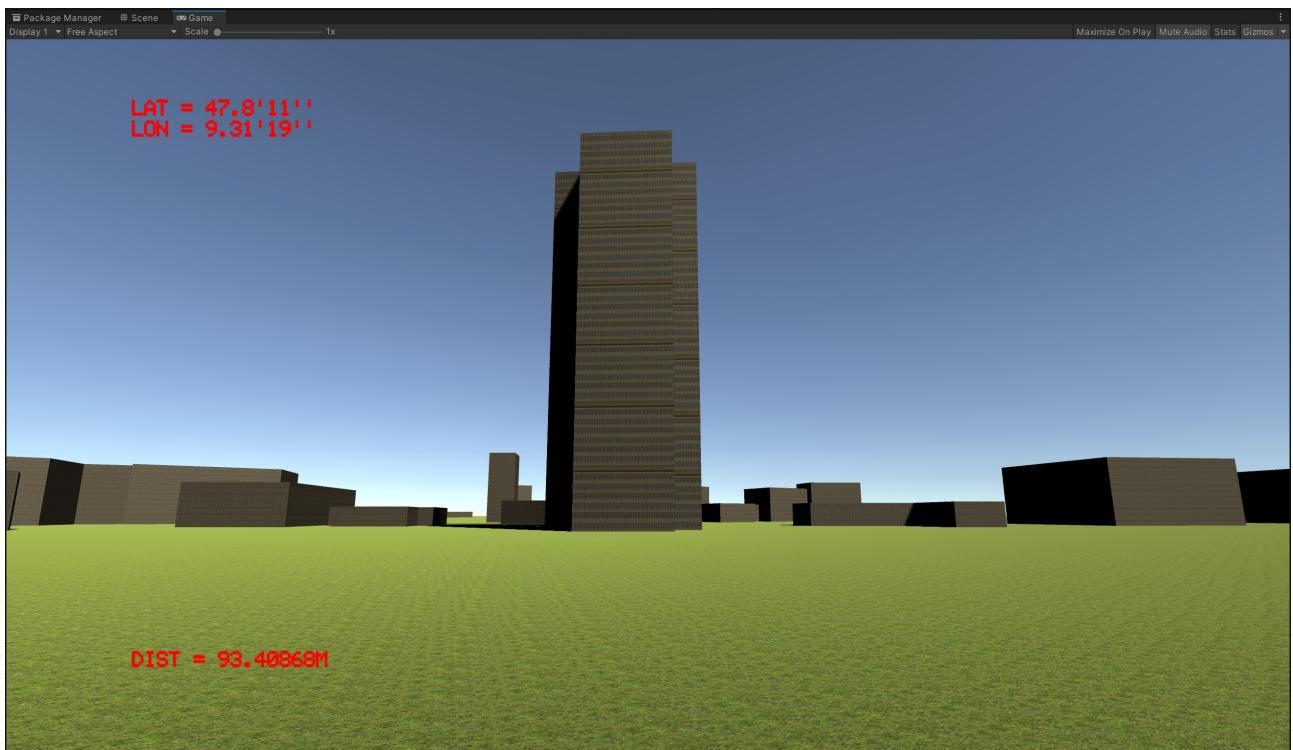


Fig. 39 : Vue citadine équivalente de la cathédrale de St-Florin dans notre modèle



Fig. 40 : Vue aérienne de la cathédrale de St-Florin sur Google Earth

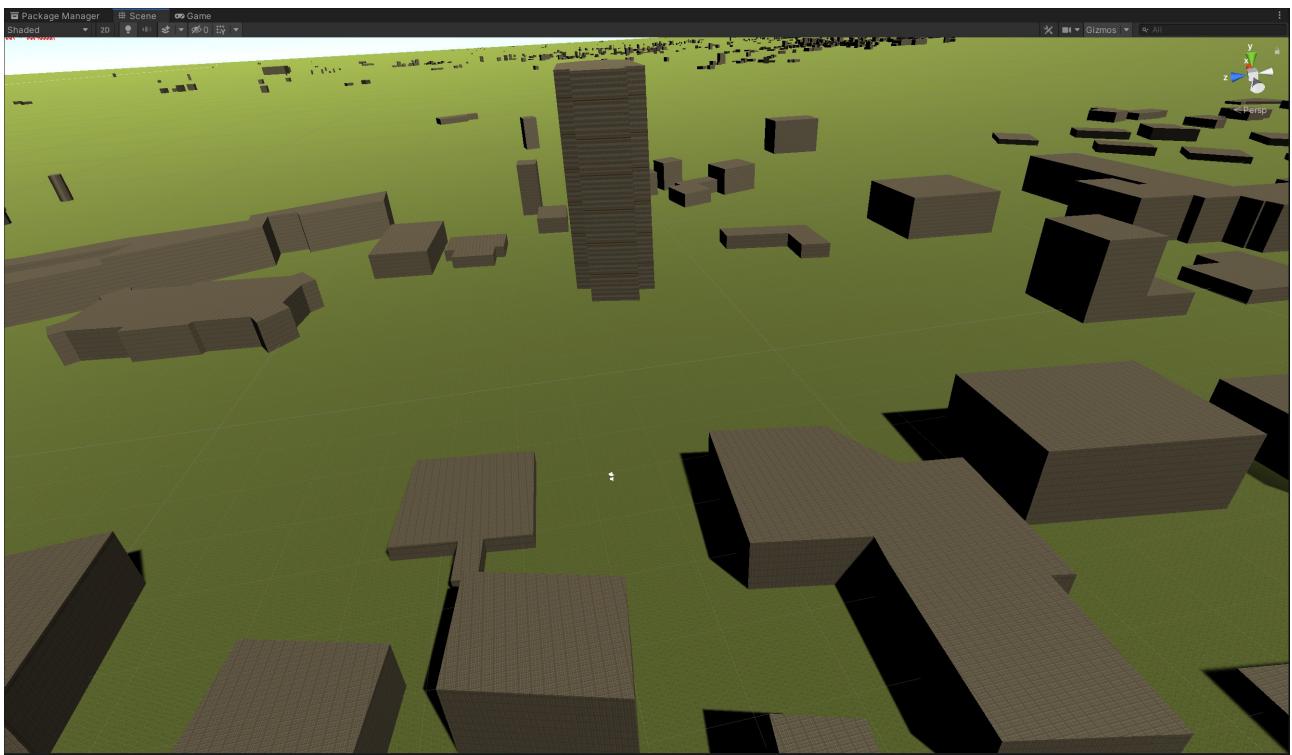


Fig. 41 : Vue aérienne équivalente de la cathédrale de St-Florin dans notre modèle

b) Le musée des Beaux-Arts du Liechtenstein

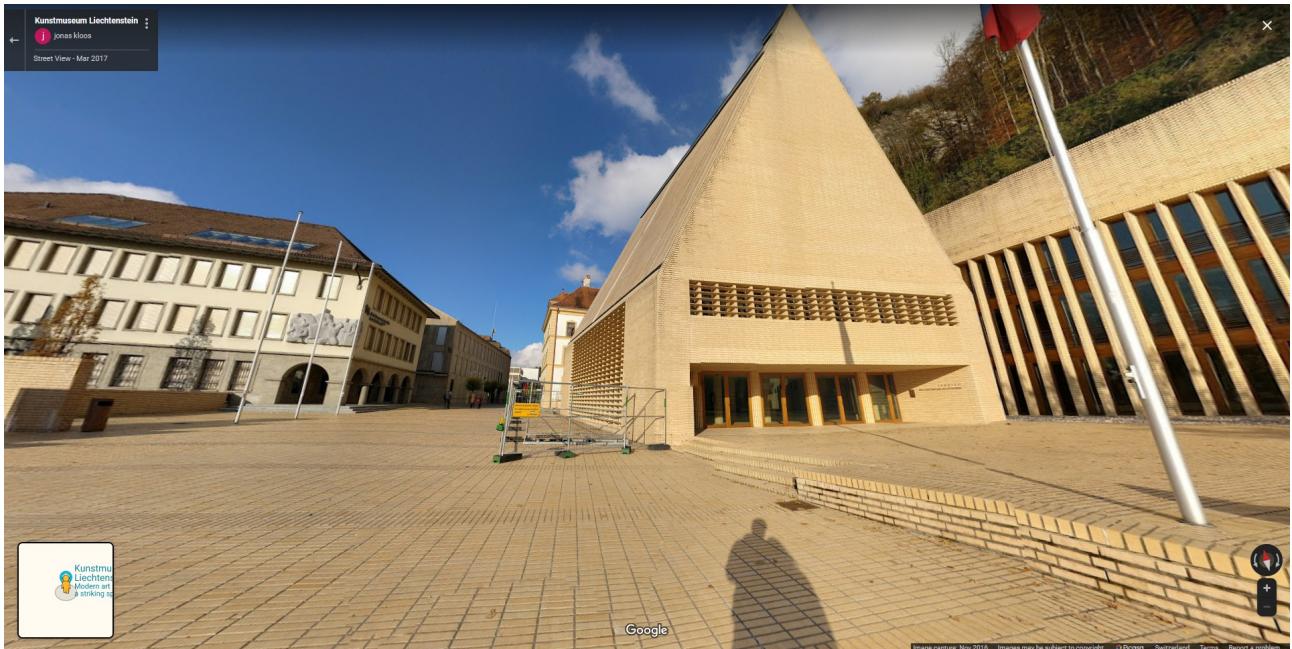


Fig. 42: Photographie du musée des Beaux-Arts du Liechtenstein à droite sur Google Maps

OSM source : <https://www.openstreetmap.org/way/28712148#map=19/47.13939/9.52224>

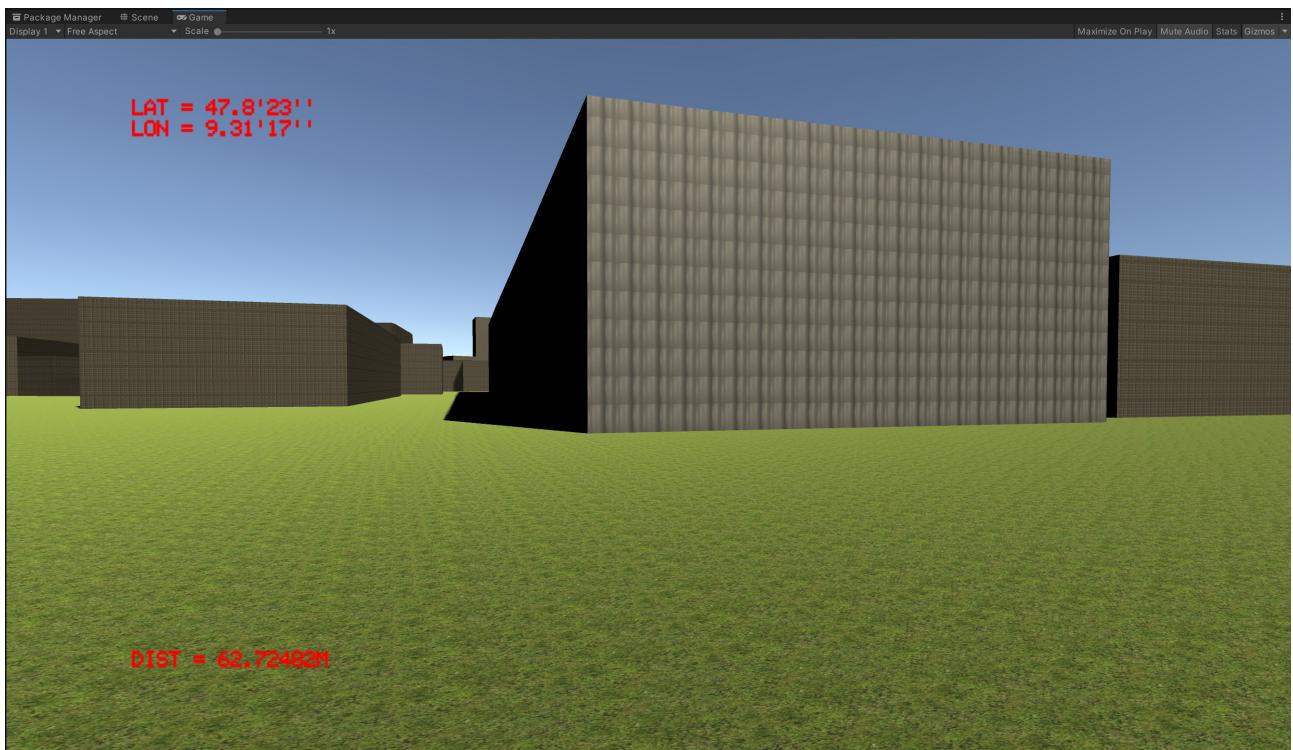


Fig. 43 : Vue citadine du musée des Beaux-Arts du Liechtenstein dans notre modèle

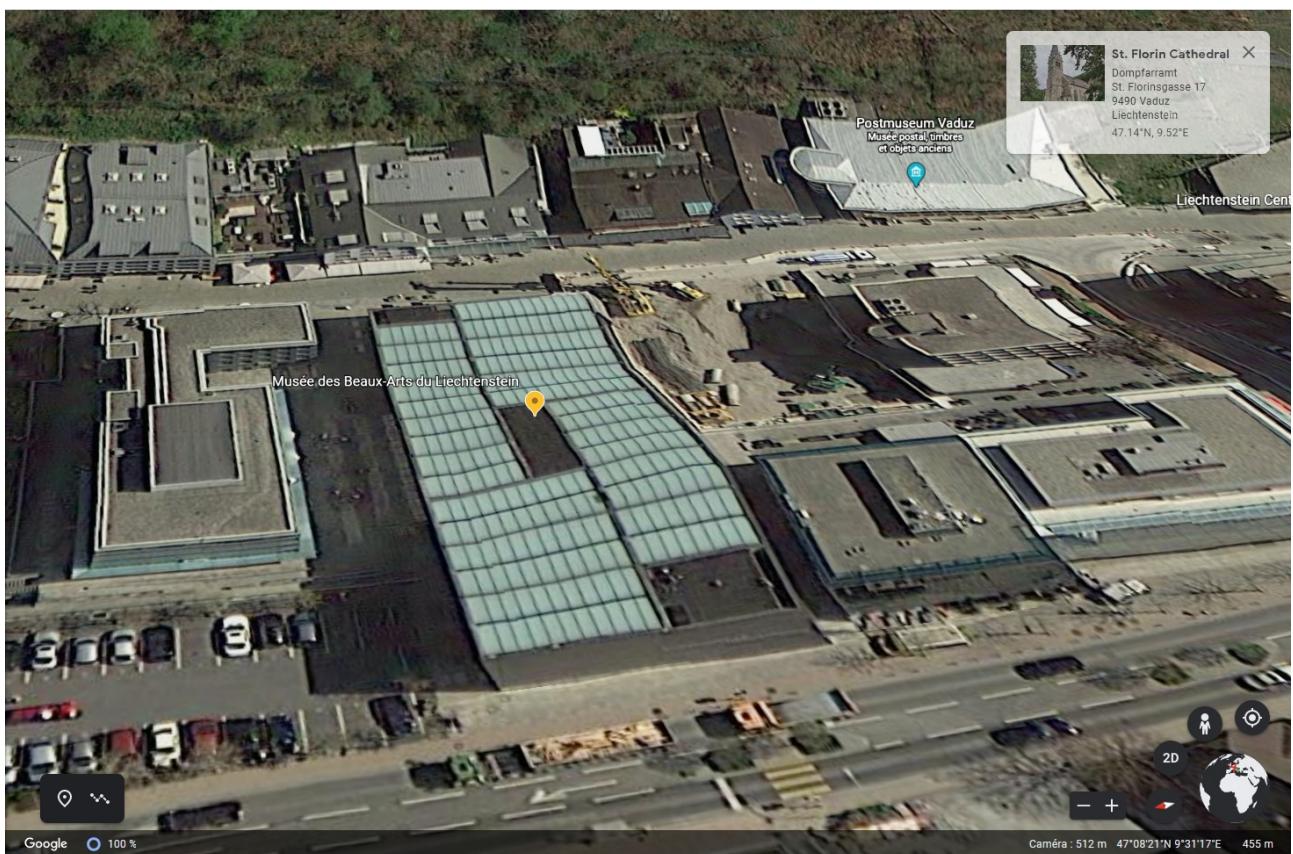


Fig. 44 : Vue aérienne du musée des Beaux-Arts du Liechtenstein sur Google Earth

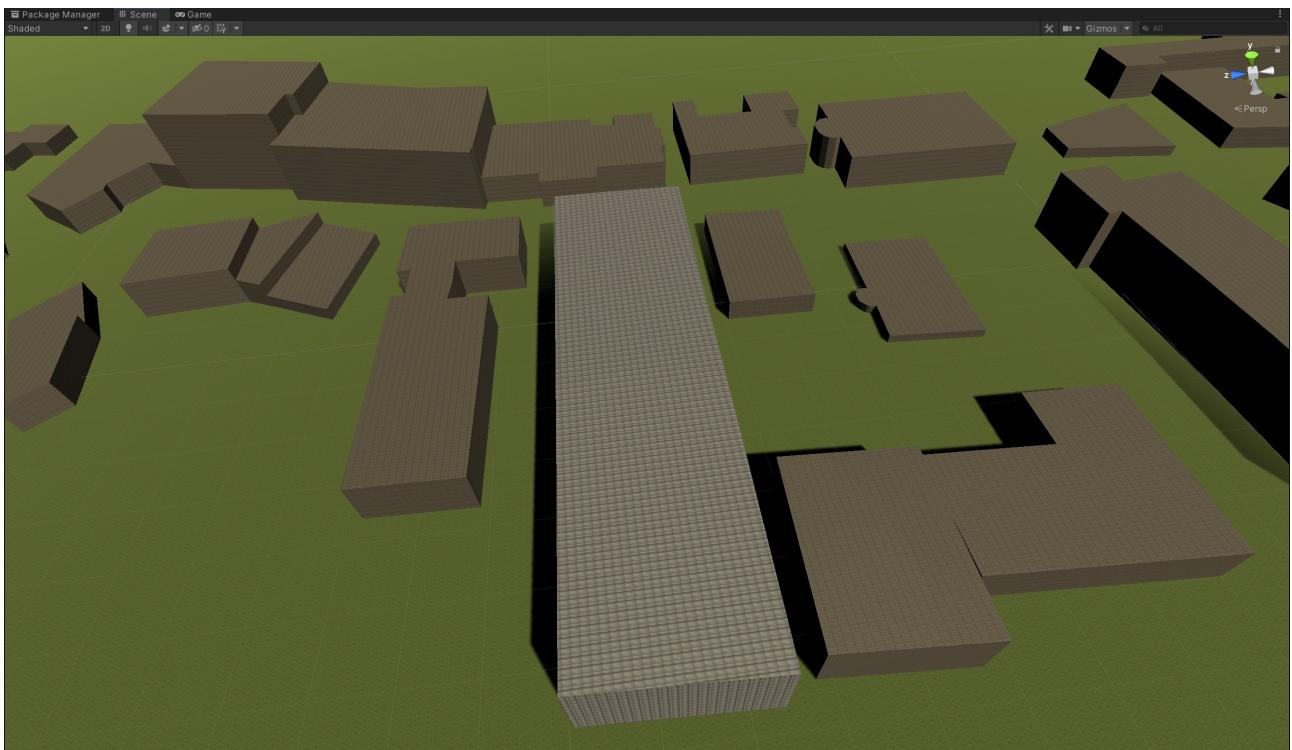


Fig. 45 : Vue aérienne équivalente du musée des Beaux-Arts du Liechtenstein dans notre modèle

c) La maison Rouge de Vaduz



Fig. 46: Photographie de la maison Rouge de Vaduz sur Google Maps

OSM source : <https://www.openstreetmap.org/way/351951309#map=19/47.14476/9.52218>

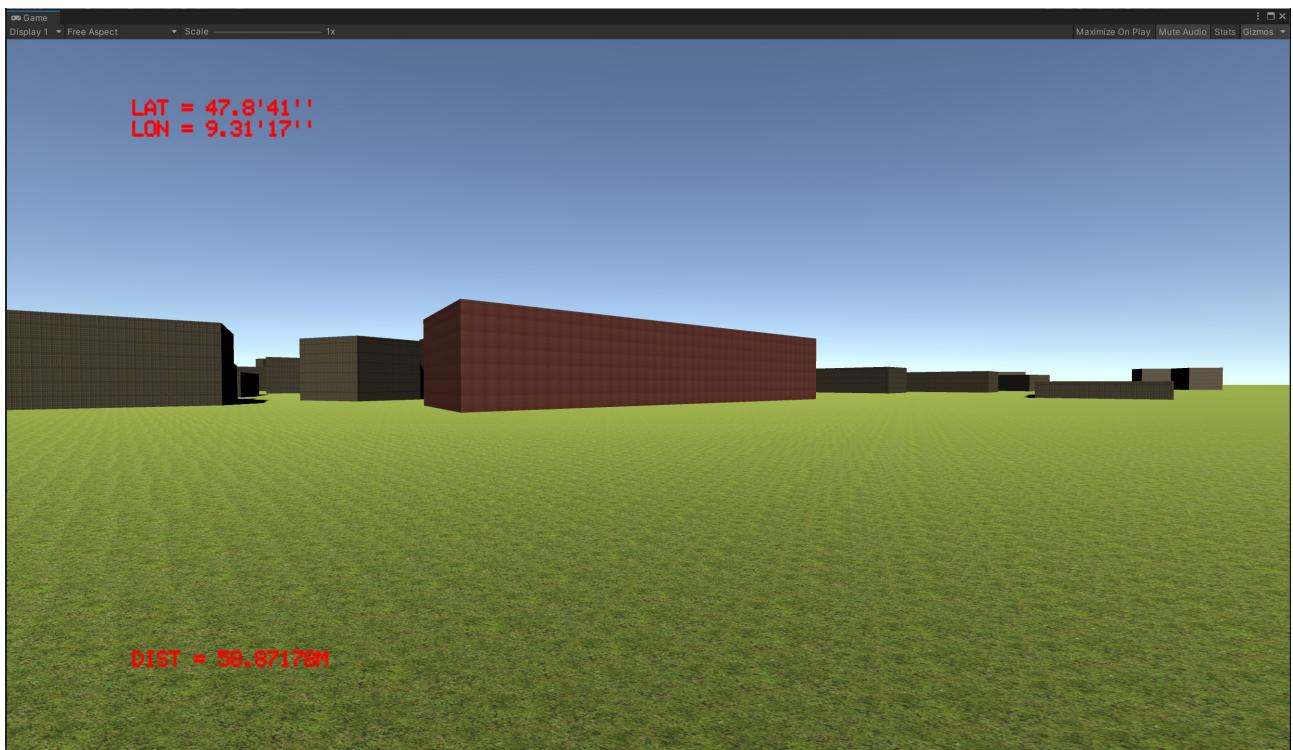


Fig. 47: Vue citadine équivalente de la maison Rouge de Vaduz dans notre modèle



Fig. 48 : Vue aérienne de la maison rouge de Vaduz sur Google Earth

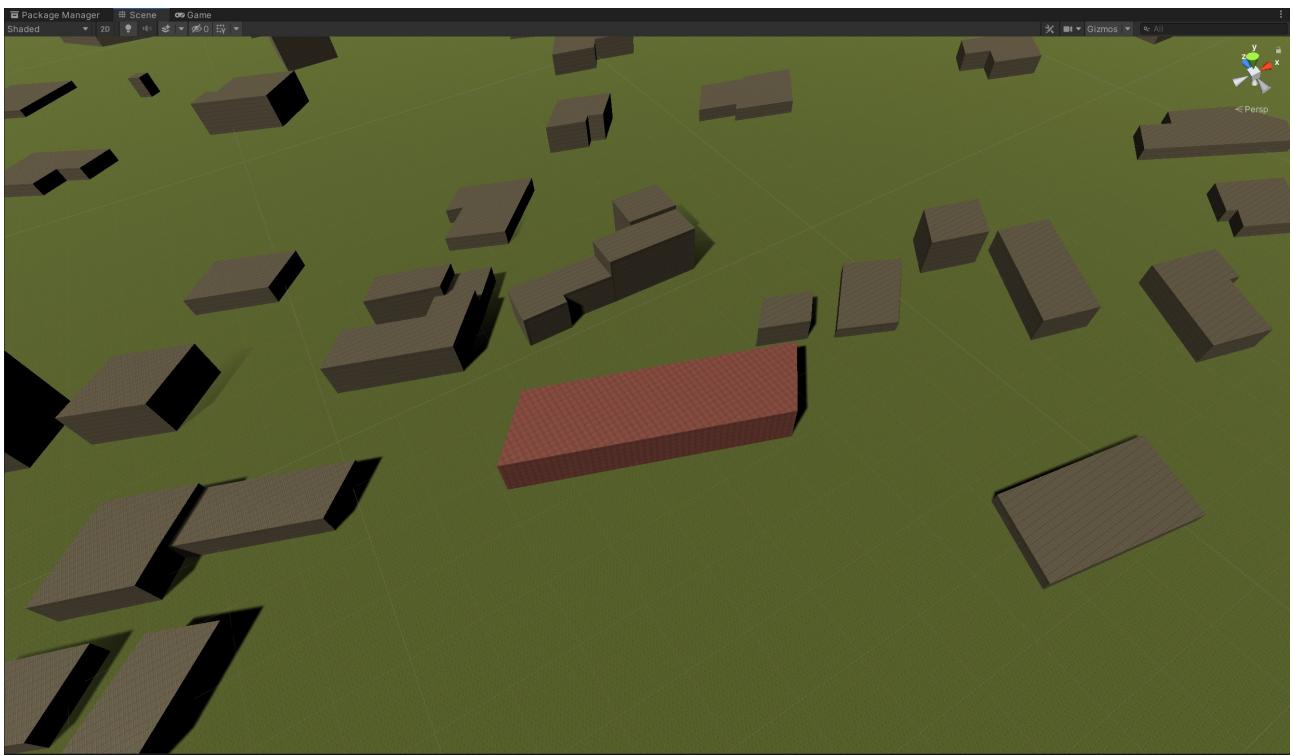


Fig. 49 : Vue aérienne équivalente de la maison rouge de Vaduz dans notre modèle

d) L'université du Liechtenstein et sa salle à manger



Fig. 50: Photographie de l'Université du Liechtenstein et de sa salle à manger en arrière-plan et premier plan respectivement sur Google Maps

OSM Source : <https://www.openstreetmap.org/way/484271148#map=18/47.14963/9.51699>



Fig. 51: Vue citadine équivalente de l'Université du Liechtenstein et de sa salle à manger dans notre modèle



Fig. 52: Vue aérienne du site de l'Université du Liechtenstein sur Google Earth

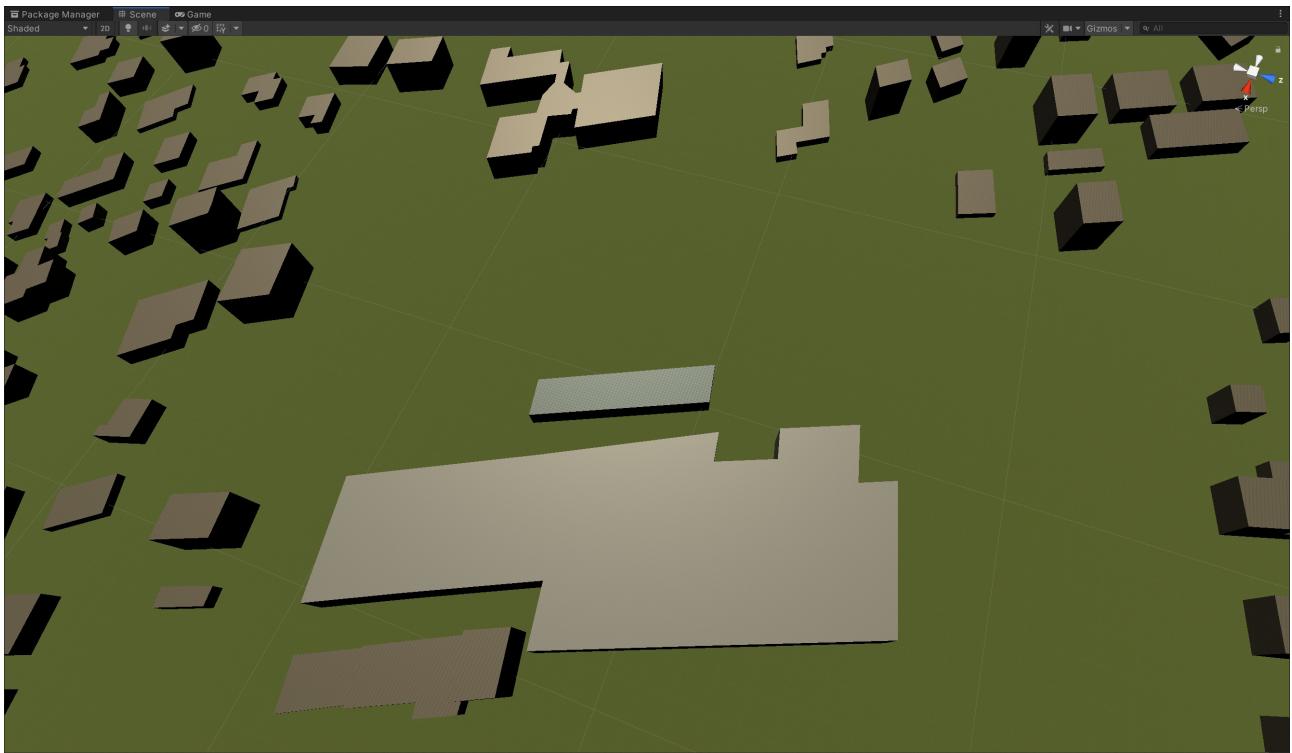


Fig. 53: Vue aérienne équivalente du site de l'Université du Liechtenstein dans notre modèle

e) Le gymnase du Liechtenstein

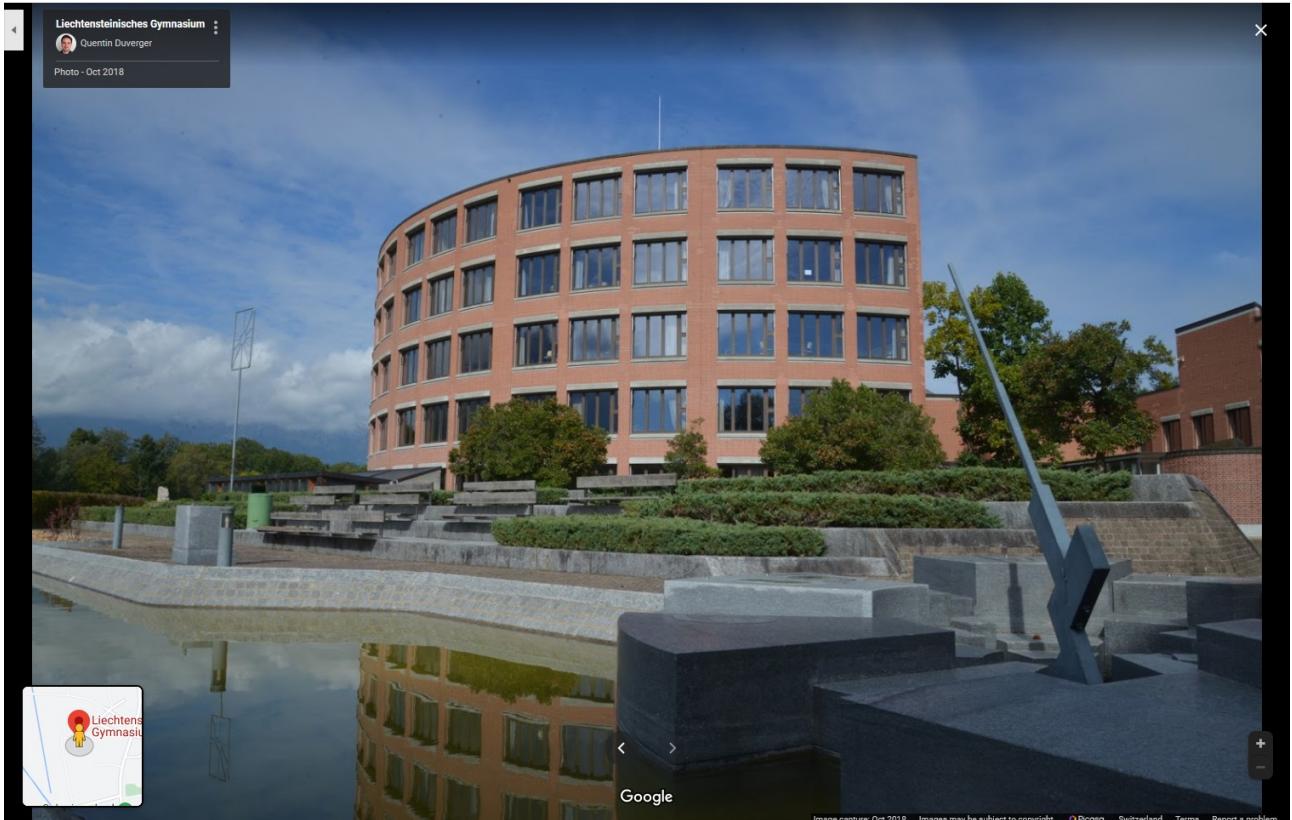


Fig. 54 : Photographie du gymnase du Liechtenstein sur Google Maps

OSM Source : <https://www.openstreetmap.org/way/485352141#map=19/47.15528/9.50451>

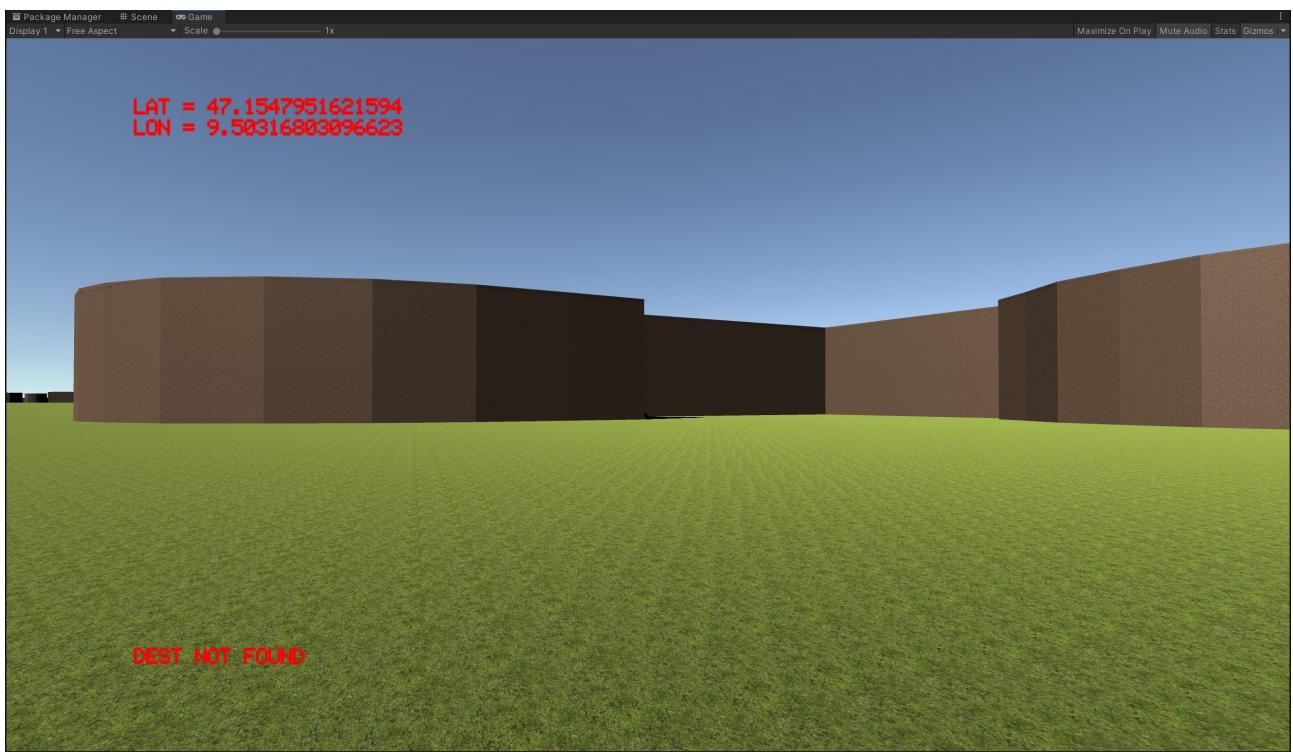


Fig. 55 : Vue citadine équivalente du gymnase du Liechtenstein dans notre modèle



Fig. 56: Vue aérienne du site du gymnase du Liechtenstein sur Google Earth

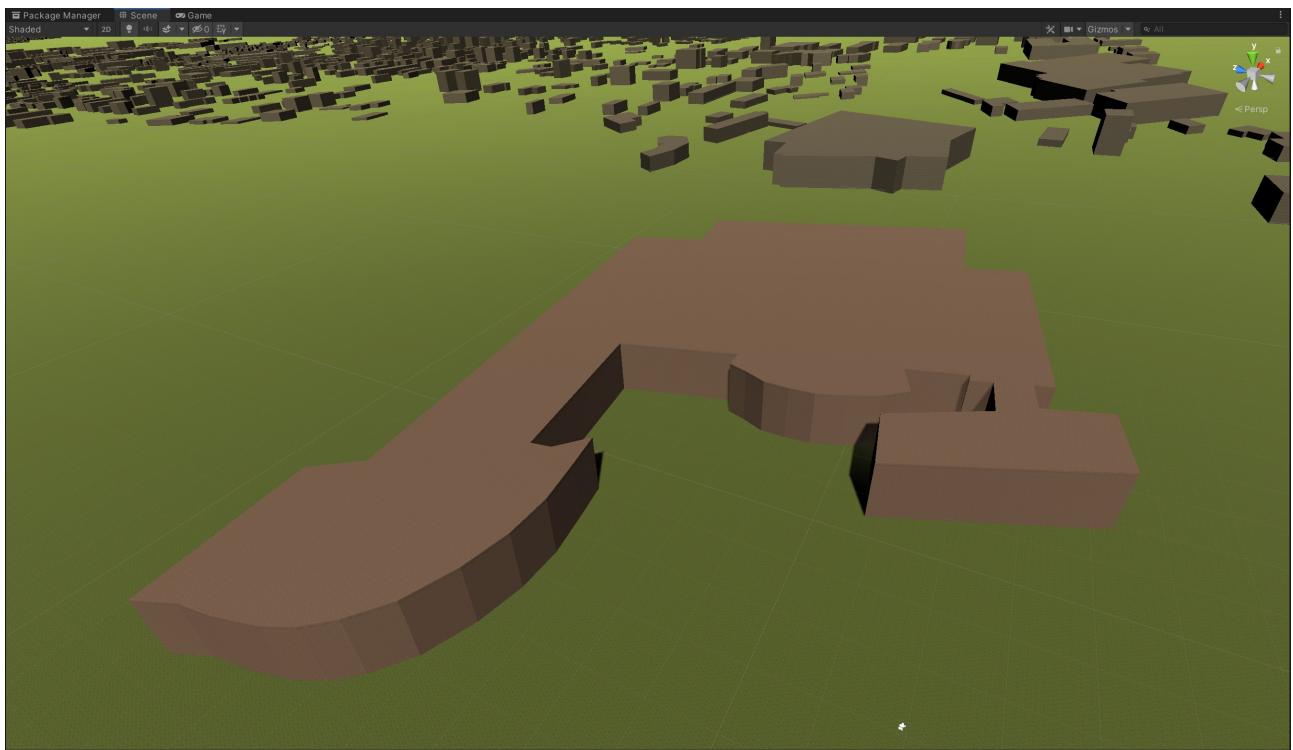


Fig. 57: Vue aérienne équivalente du site du gymnase du Liechtenstein dans notre modèle

f) La garderie de Kosthaus



Fig. 58 : Photographie de la garderie de Kosthaus sur Google Maps

OSM source : <https://www.openstreetmap.org/way/44964336#map=18/47.10545/9.52550>

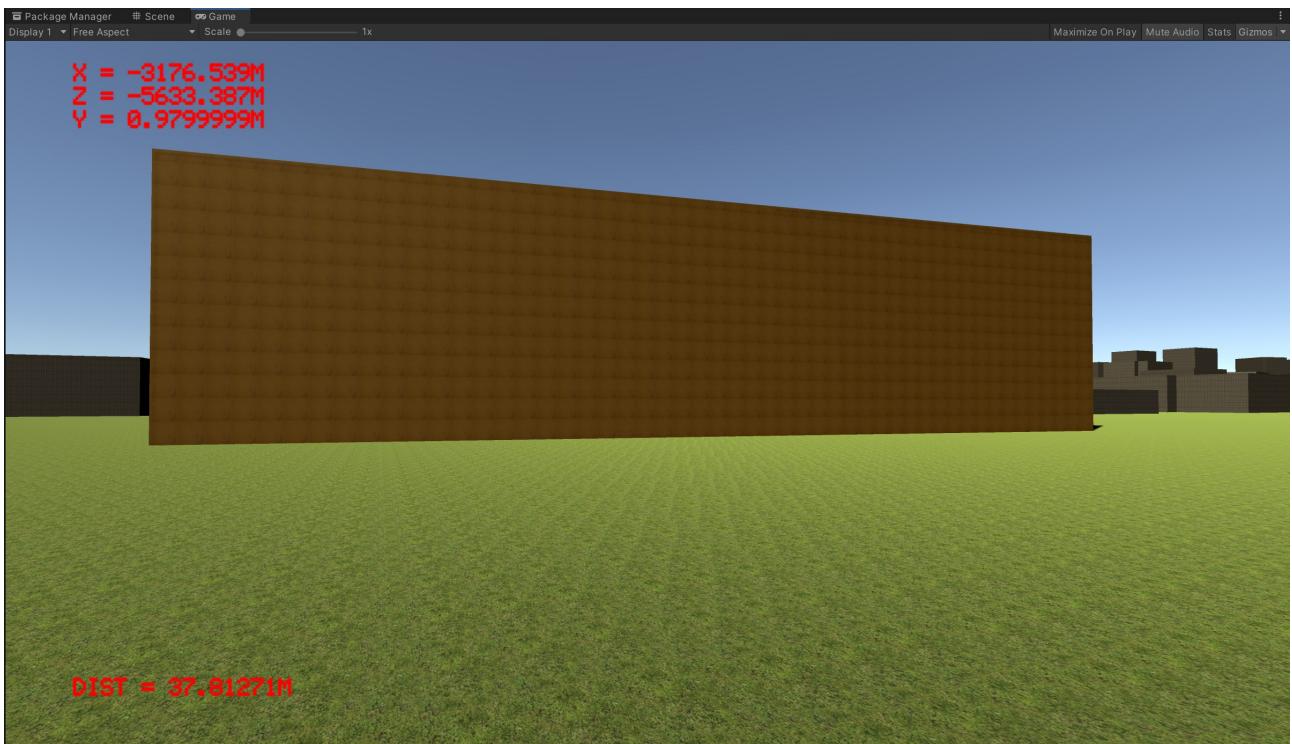


Fig. 59 : Vue citadine équivalente de la garderie de Kosthaus dans notre modèle



Fig. 60: Vue aérienne de la garderie de Kosthaus sur Google Earth

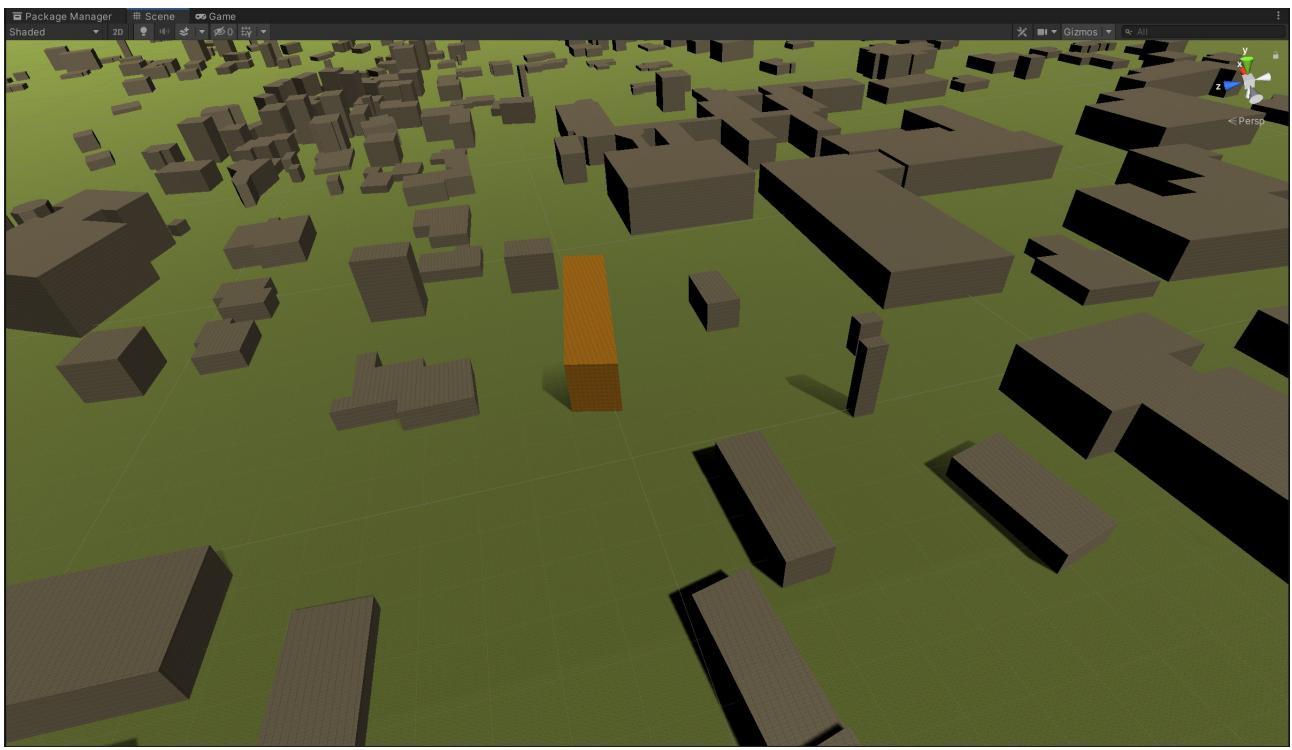


Fig. 61: Vue aérienne équivalente de la garderie de Kosthaus dans notre modèle

g) Swarovski AG



Fig. 62 : Photographie du site de l'entreprise Swarovski AG sur Google Maps

OSM Source : <https://www.openstreetmap.org/way/25208578#map=18/47.10865/9.52156>

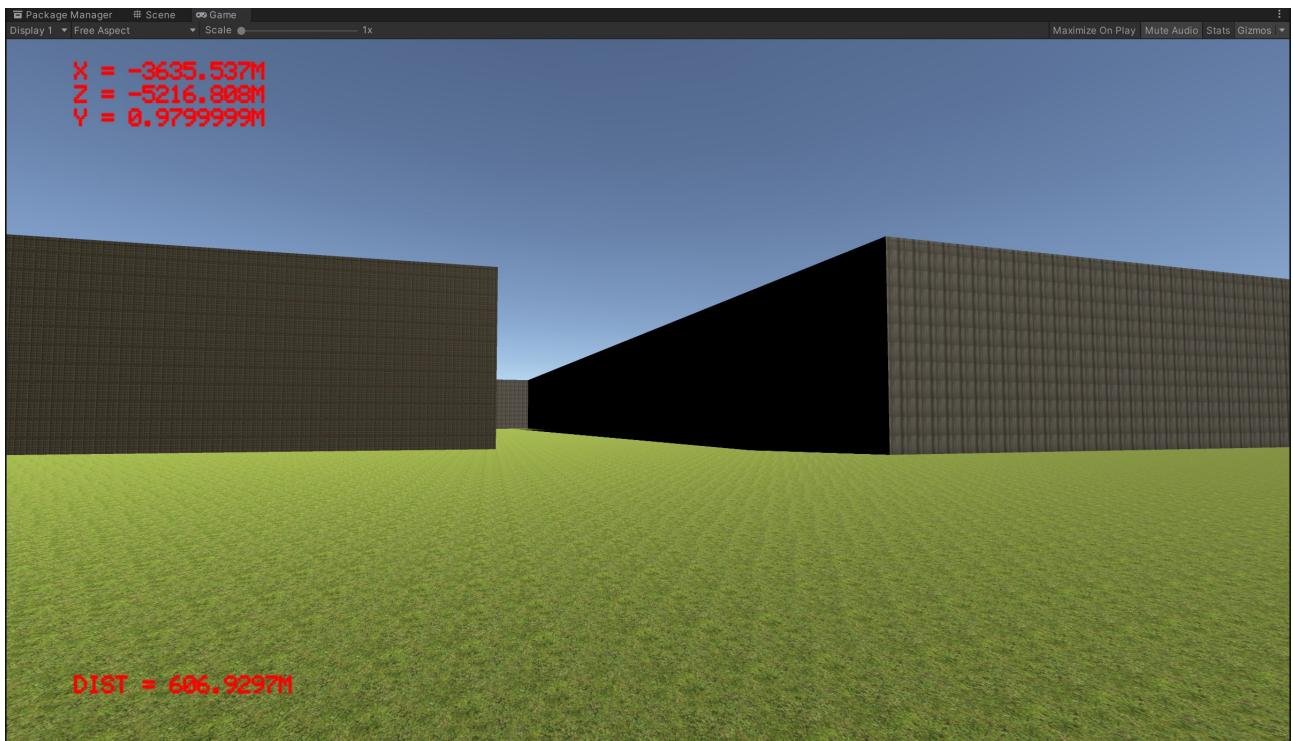


Fig. 63 : Vue citadine équivalente du site de l'entreprise Swarovski AG dans notre modèle



Fig. 64: Vue aérienne du site de l'entreprise Swarovski AG sur Google Earth

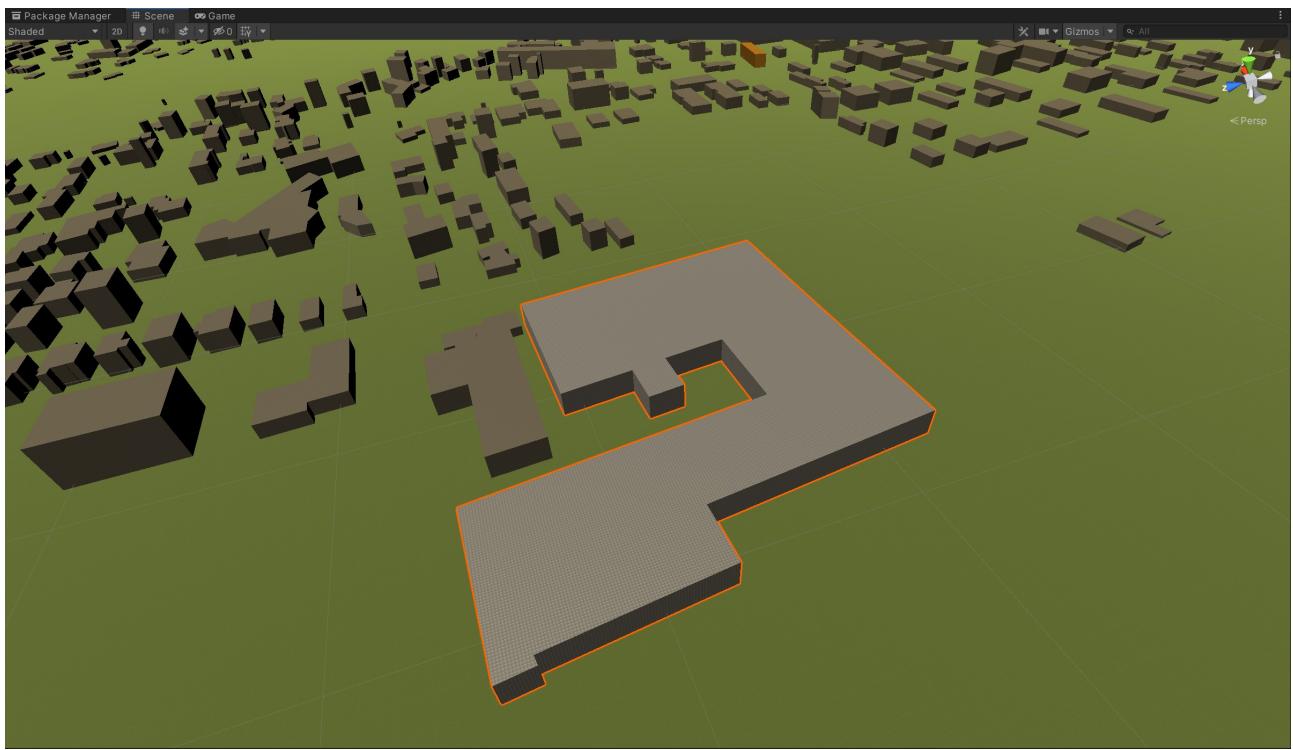


Fig. 65: Vue aérienne équivalente du site de l'entreprise Swarovski AG dans notre modèle

h) Hoval



Fig. 66 : Photographie du site de l'entreprise Hoval sur Google Maps

OSM Source : <https://www.openstreetmap.org/way/529570113#map=18/47.12228/9.52259>

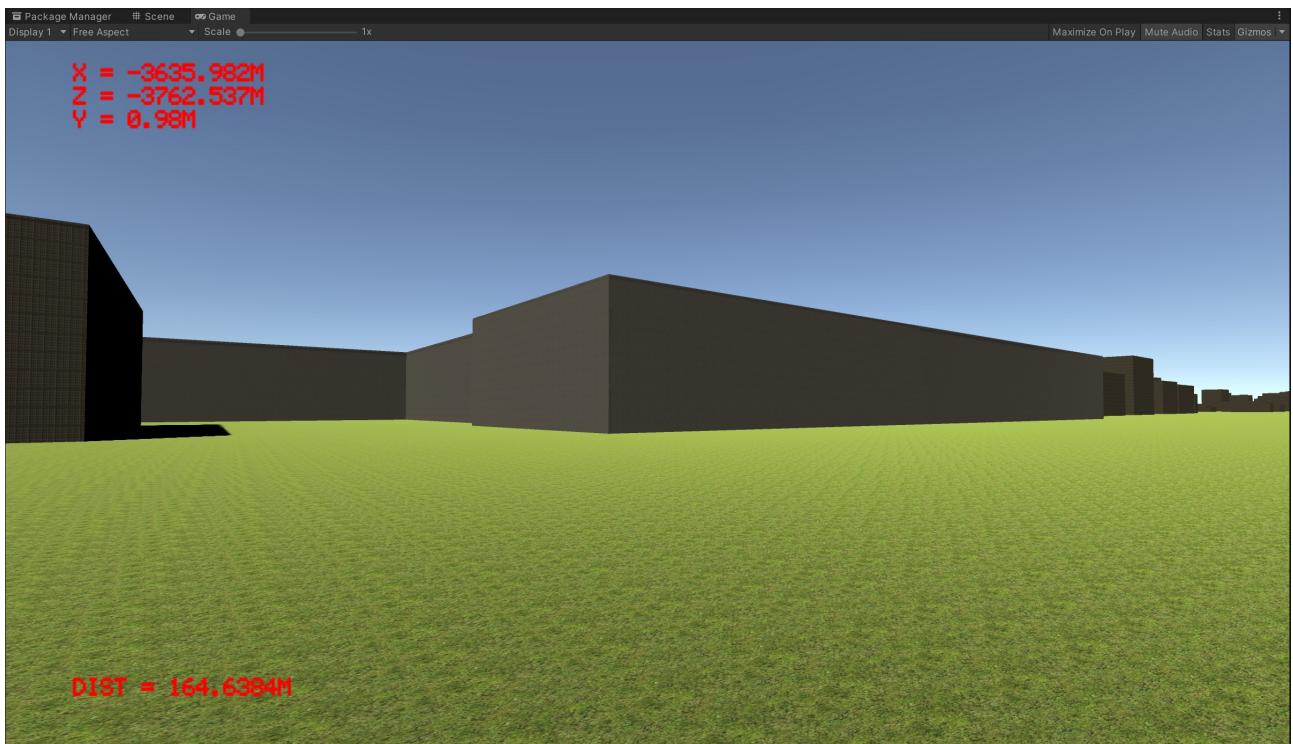


Fig. 67 : Vue citadine équivalente de l'entreprise Hoval dans notre modèle



Fig. 68 : Vue aérienne du site de l'entreprise Hoval sur Google Earth

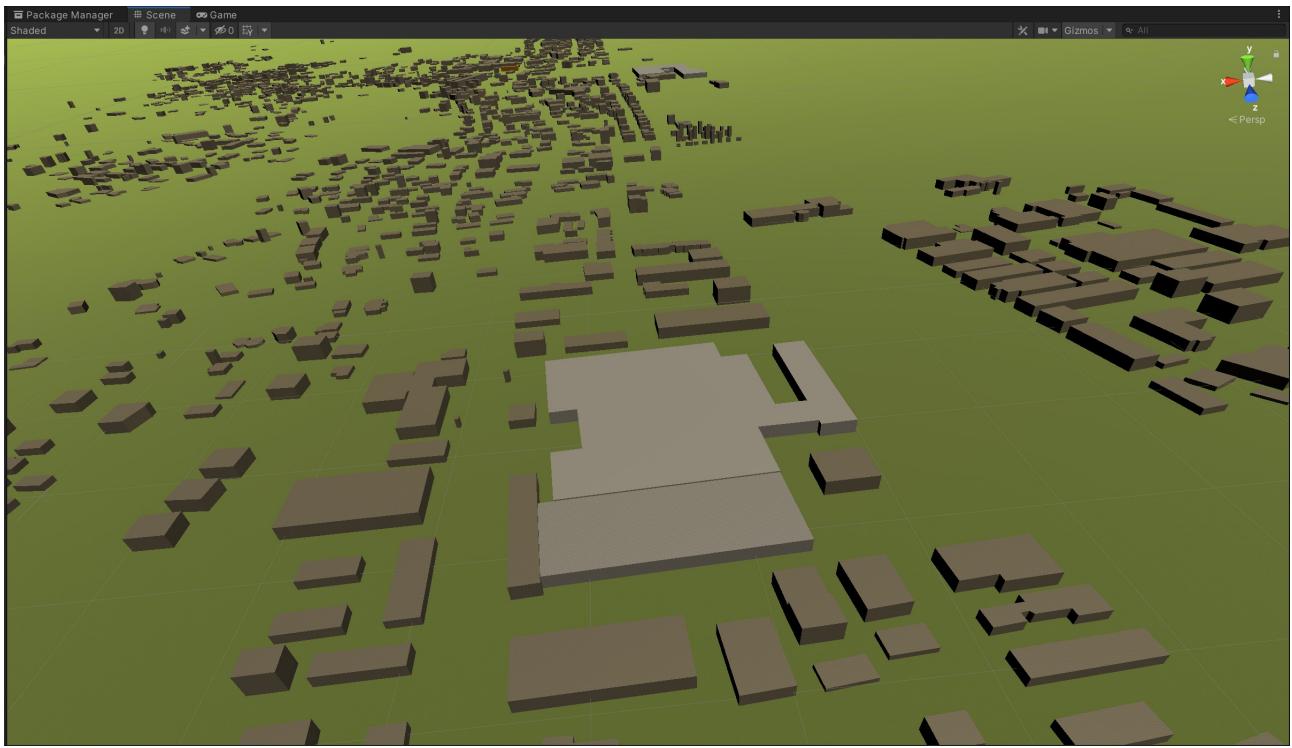


Fig. 69 : Vue aérienne équivalente du site de l'entreprise Hoval dans notre modèle

i) Les fenêtres d'inspecteur des objets associés aux détails de ces 8 bâtiments

Voici ci-dessous les fenêtres d'inspecteur des objets associés aux détails de chacun des 8 bâtiments vus plus haut :

Inspector

| | |
|--------------------------|--|
| ID | 391994214 |
| Tag | Untagged |
| Layer | Default |
| Transform | |
| Position | X -5588.25 Y 0 Z -180.5 |
| Rotation | X 0 Y 0 Z 0 |
| Scale | X 1 Y 1 Z 1 |
| Building (Script) | |
| Script | Building |
| General attributes | |
| Node Positions | id=3952813409 lat=47.1553778 lon=9.5033043 id=3952813405 lat=47.1553389 lon=9.5033094 id=3952813429 lat=47.1552933 lon=9.5033576 |
| Material | building2 |
| Geometry | Cube |
| Width | 124 |
| Length | 188.75 |
| Obj Name | |
| Amenity | |
| Source | |
| Surface | |
| Elevation | 0 |
| Miscellaneous | addr:city = Vaduz addr:street = Marianumstrasse root:levels = 0 |
| Building attributes | |
| Height | 11.7084 |
| N Floors | 5 |
| Net Internal Surface | 59148.13 |
| Building Type | school |
| Age | |
| House Number | 45 |
| Post Code | 9490 |
| Street | Marianumstrasse |
| City | Vaduz |
| Country | L1 |

Inspector

| | |
|--------------------------|--|
| ID | 29598457 |
| Tag | Untagged |
| Layer | Default |
| Transform | |
| Position | X -4172.875 Y 0 Z -755 |
| Rotation | X 0 Y 0 Z 0 |
| Scale | X 1 Y 1 Z 1 |
| Building (Script) | |
| Script | Building |
| General attributes | |
| Node Positions | id=326085903 lat=47.1502093 lon=9.5165041 id=326085904 lat=47.1501341 lon=9.5160999 id=4107526960 lat=47.1500339 lon=9.5161402 |
| Material | building2 |
| Geometry | Cube |
| Width | 63 |
| Length | 80.375 |
| Obj Name | Mehrzweckhalle |
| Amenity | university |
| Source | |
| Surface | |
| Elevation | 0 |
| Miscellaneous | addr:city = Vaduz addr:street = Fürst-Franz-Josef-Strasse addr:country = L1 |
| Building attributes | |
| Height | 11.63834 |
| N Floors | 1 |
| Net Internal Surface | 3197.781 |
| Building Type | school |
| Age | |
| House Number | 9 |
| Post Code | 9490 |
| Street | Fürst-Franz-Josef-Strasse |
| City | Vaduz |
| Country | L1 |

Fig. 70 à gauche : Fenêtre d'inspecteur de l'objet associé au bâtiment principal du gymnase du Liechtenstein à Vaduz, d'identifiant 391994214

Fig. 71 à droite : Fenêtre d'inspecteur de l'objet associé au bâtiment principal de l'université du Liechtenstein à Vaduz, d'identifiant 29598457

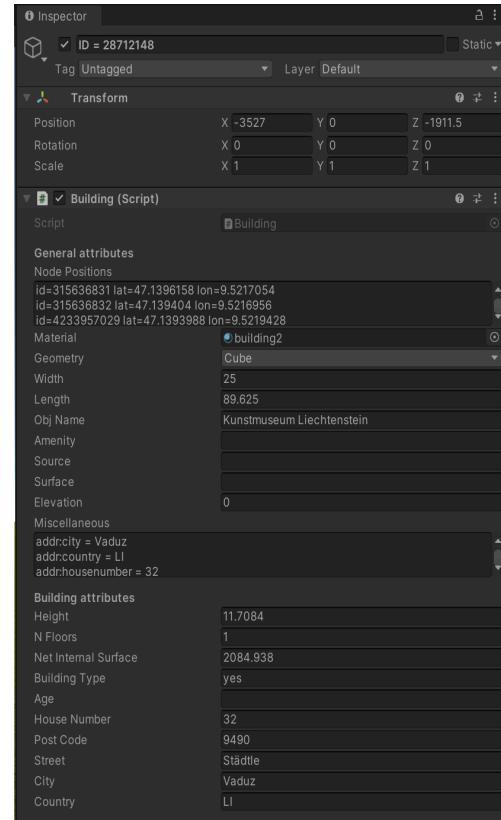
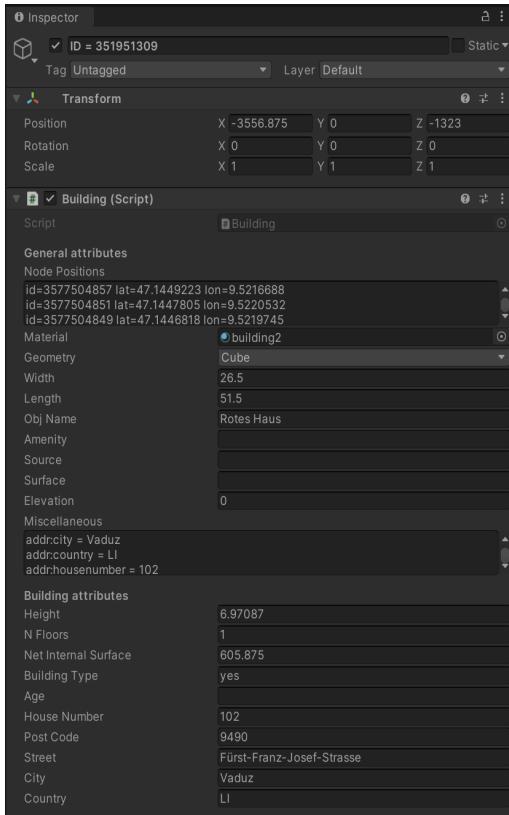


Fig. 72 à gauche : Fenêtre d'inspecteur de l'objet associé à la maison Rouge de Vaduz, d'identifiant 351951309

Fig. 73 à droite : Fenêtre d'inspecteur de l'objet associé au musée des Beaux-Arts du Liechtenstein à Vaduz, d'identifiant 28712148

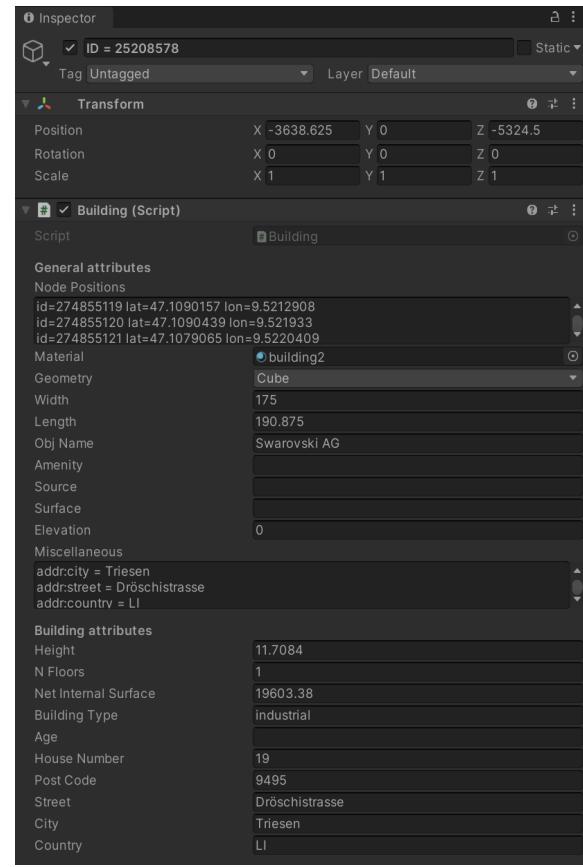
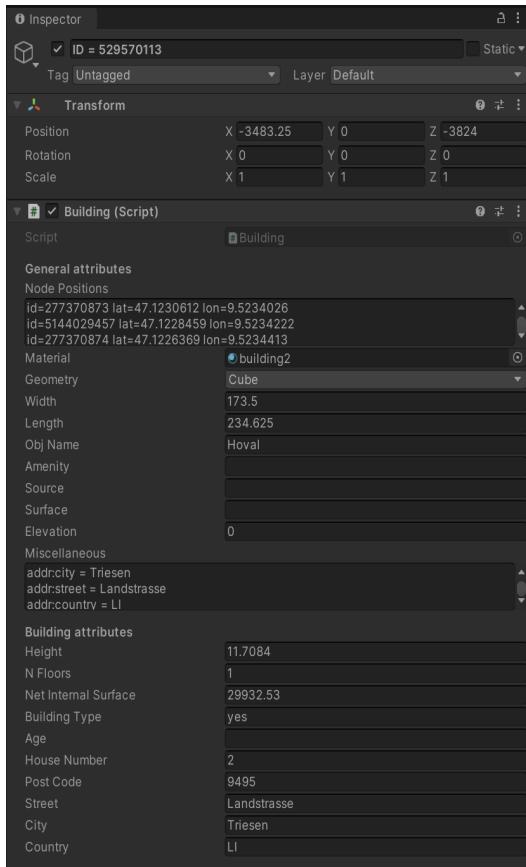


Fig. 74 à gauche : Fenêtre d'inspecteur de l'objet associé au bâtiment principal de l'entreprise Hoval à Triesen, d'identifiant 529570113

Fig. 75 à droite : Fenêtre d'inspecteur de l'objet associé au bâtiment principal de l'entreprise Swarovski AG à Triesen, d'identifiant 25208578

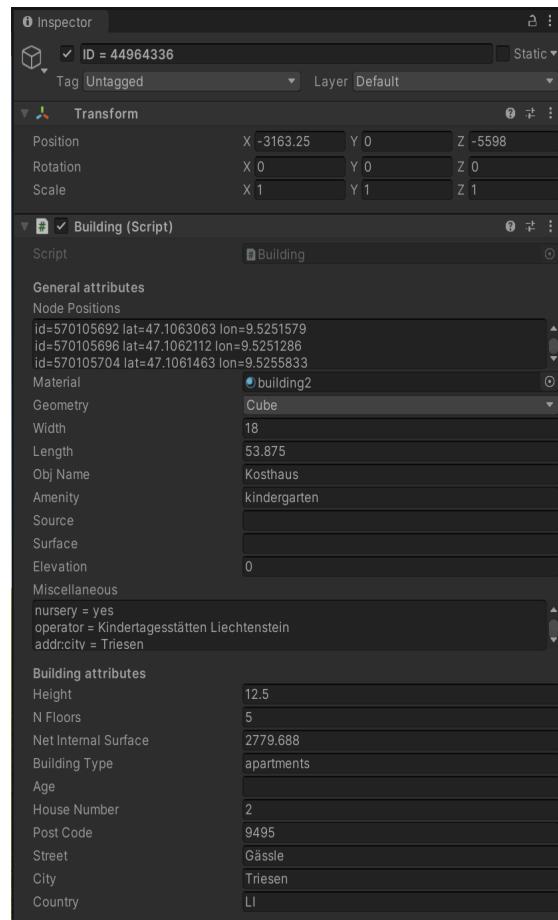
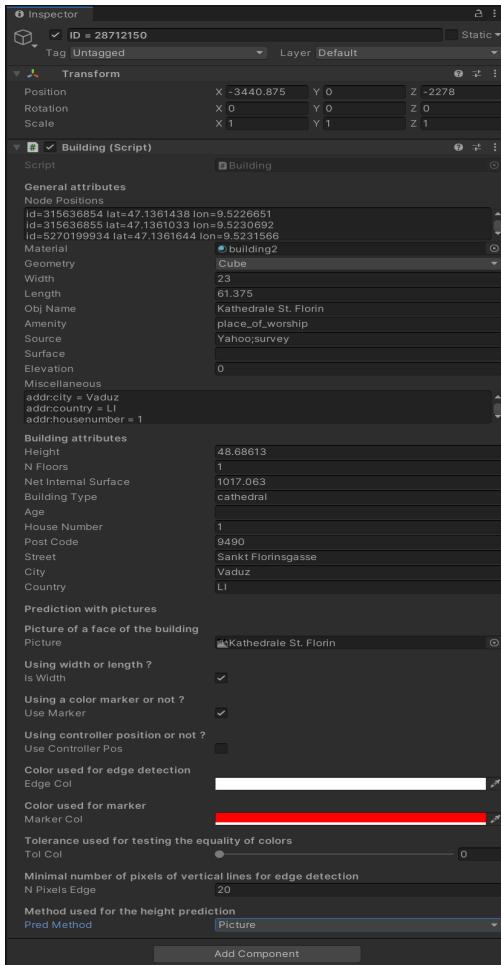


Fig. 76 à gauche: Fenêtre d'inspecteur de l'objet associé à la cathédrale de St-Florin à Vaduz, d'identifiant 28712150

Fig. 77 à droite : Fenêtre d'inspecteur de l'objet associé à la garderie de Kosthaus à Triesen, d'identifiant 44964336

VI/ Conclusion

En comparant en partie V/ les vues citadines et aériennes de bâtiments du Liechtenstein sur Google Maps/Earth avec les vues équivalentes de notre modèle, on peut en déduire que celui-ci est plutôt cohérent avec la réalité : la longueur et la largeur d'un bâtiment ont des valeurs exactes et la hauteur d'un bâtiment est cohérente avec la réalité dans la plupart des cas. Par ailleurs, les données sur les attributs de chaque bâtiment affichées dans la fenêtre d'inspecteur de l'objet associé s'avèrent très utiles pour récolter des informations diverses : il suffit alors à l'utilisateur de rechercher un bâtiment par son identifiant unique dans la fenêtre de hiérarchie de l'interface de Unity.

Finalement, notre modèle répond bien aux besoins des utilisateurs et des autorités des villes afin de s'informer et permettre la planification de projets de travaux urbains. Il constitue donc une alternative simple et intuitive aux principaux logiciels de reconstruction de modèles 3D urbains à partir de données OSM tels que : Google Maps, Google Earth, Wikimapia, etc... notre modèle a cependant quelques limites : d'une part, celui-ci n'effectue pas la reconstruction de façon automatique mais semi-automatique puisqu'elle demande certaines actions de la part de l'utilisateur.

comme : importer des données OSM, déplacer des fichiers texte ou XML dans le dossier *Assets*, ou encore prédire la hauteur de bâtiments avec un modèle de régression ou des photographies. D'autre part, notre modèle est moins complet que les autres alternatives existantes car il ne modélise actuellement que les bâtiments en LoD1 et rien d'autre... c'est pour cette raison que j'envisage d'ajouter d'autres types d'objets urbains dans notre modèle par la suite. Cependant, les bâtiments restent tout de même les constructions les plus communes dans les villes.

Notez qu'il existe également un logiciel payant très similaire au nôtre nommé *CityGen3D*, qui peut être téléchargé à partir de « l'Asset Store » de Unity et qui possède son propre site Internet :
<https://www.citygen3d.com/>

Pour résumer le processus de génération de notre modèle 3D urbain depuis le début, celui-ci se découpe en 3 parties : en partie **II/**, nous avions expliqué les étapes qui permettaient d'importer des données OSM notamment à partir de la librairie *OsmSharp*. En partie **III/**, nous avions présenté deux méthodes de prédiction de la hauteur de bâtiments utilisées par notre modèle : le modèle de régression par arbre de décision et la détection des contours sur des photographies de bâtiments. Enfin en partie **IV/**, nous avions décrit les étapes d'implémentation de notre modèle sur Unity, notamment les actions de l'utilisateur nécessaires à la construction et modification de notre modèle.

Pour terminer, je prévois d'ajouter de nouvelles fonctionnalités à notre modèle pour le reconvertir en un modèle LoD2 plus détaillé et avec plus de types d'objets urbains : ceci dans l'objectif de fournir une alternative viable et efficace à celles déjà citées plus haut. Je prévois notamment d'ajouter des toits, portes et fenêtres aux bâtiments reconstruits et de modéliser les routes, espaces verts, lacs, rivières, etc... à partir de données OSM : en somme, je prévois de modéliser le maximum d'objets urbains simplement à partir des attributs présents pour un élément OSM donné, tout en conservant les performances de notre modèle. Je prévois également de modéliser les éléments associés à une relation OSM tels que les multi-polygones, car notre modèle ne peut actuellement modéliser que les noeuds et chemins OSM associés à des bâtiments.

D'ailleurs, pour citer quelques statistiques sur la base de données OSM à partir du lien suivant <https://taginfo.openstreetmap.org/keys/building#overview>, nous pouvons observer que :

- 5.87% des données totales (tous les types d'éléments OSM) de la base correspondent à des bâtiments
- 0.70% des noeuds de la base sont des noeuds associés à des bâtiments
- 58.81% des chemins de la base sont des chemins associés à des bâtiments
- 9.27% des relations de la base sont des relations associées à des bâtiments
- environ 0.27% des éléments associés à des bâtiments sont des noeuds OSM
- environ 99.54% des éléments associés à des bâtiments sont des chemins OSM
- environ 0.18% des éléments associés à des bâtiments sont des relations OSM

Cela montre que la très grande majorité (plus de 99% !) des éléments associés à des bâtiments sont des chemins OSM : notre modèle n'est donc pas incomplet lorsqu'il s'agit de modéliser les bâtiments à partir de données OSM. En revanche, notre modèle ne modélise avec les bâtiments seulement 5.87% des données totales de la base ce qui est relativement peu... d'où l'utilité de modéliser d'autres types d'objets urbains dans notre modèle.

Dans tous les cas, on peut dire que notre projet de reconstruction de bâtiments à partir de données OSM a un avenir certain devant lui !