ENSG Géomatique

ÉCOLE NATIONALE DES SCIENCES GÉOGRAPHIQUES

SAPEURS POMPIERS DE MONACO

Documentation développeur

Recherche et développement d'un viewer BIM opérationnel

FIRE:BIM

Firefighters' Interactive Roving Equipment: BIM

BRES Lucas, HUE Valentin, LABOURG Priscillia, PENG Yanzhuo

Chef de projet : BRES Lucas

Table des matières

1 Contexte

Le Corps des sapeurs-pompiers de Monaco, unité militaire de la Force Publique en Principauté avec les Carabiniers du Prince, est chargé de la lutte contre l'incendie et des risques de toute nature, du secours à personnes et de la protection des biens sur le territoire de la Principauté de Monaco.

Pour préparer leurs interventions, le Corps des sapeurs-pompiers de Monaco dispose d'un modèle 3D des bâtiments et de leurs intérieurs, et souhaiterait le rendre accessible à leurs agents en intervention.
L'apport de cet outil sur les lieux d'opération simplifiera la reconnaissance du terrain, et aura pour but d'accélérer et améliorer la prise de décision.

Ce projet de réalisation d'une visionneuse de bâtiment 3D sur un appareil mobile pourrait ainsi augmenter l'efficacité des interventions et réduire la prise de risque des sapeurs-pompiers. L'utilisation de ce plan 3D diminuera notamment les imprévus qui peuvent survenir durant des manoeuvres en conditions dangereuses. Pour cela, des tablettes tactiles sont utilisées, car elles sont faciles à transporter, et permettent aux agents une utilisation simple et intuitive.

Figure 1: Exemple d'équipements de Sapeurs-Pompiers

2 Comparaison entre les différentes solutions

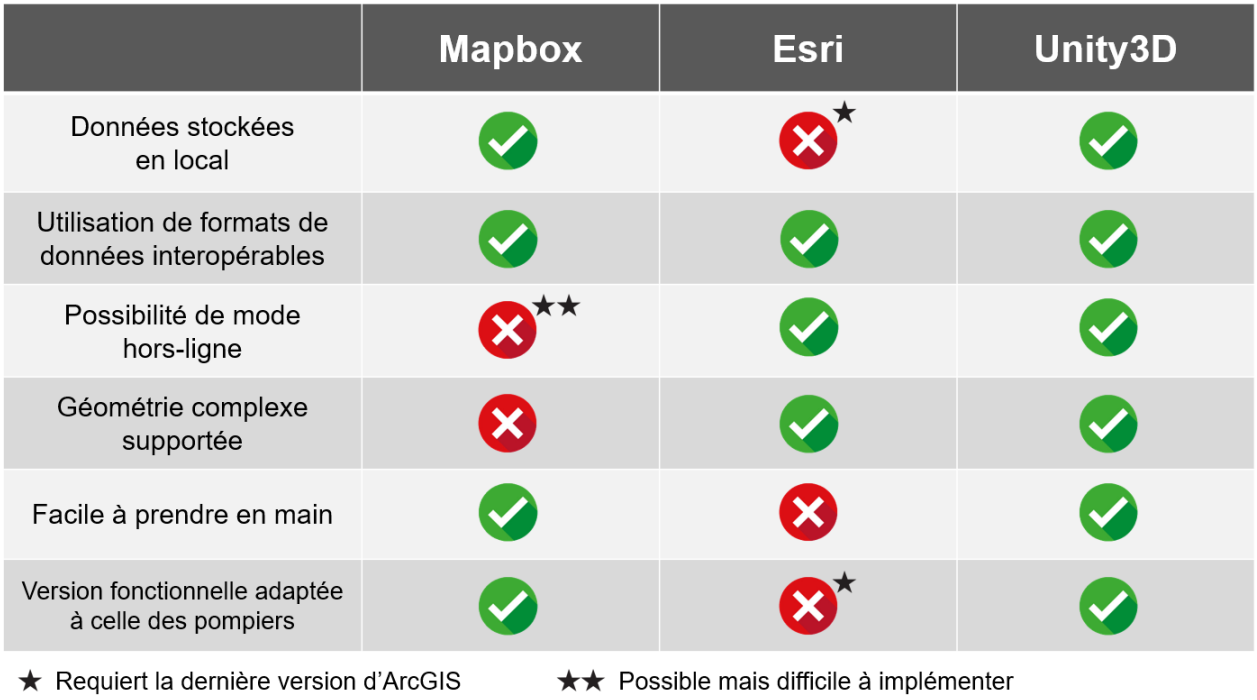

Figure 2: Tableau de comparaison des fonctionnalités des trois solutions Mapbox, Esri et Unity3D

Voici ci-dessus un tableau qui fait la comparaison, entre les trois solutions que nous avons étudiées, des plus importantes contraintes et fonctionnalités . Les solutions Esri et Unity3D sont toutes les deux viables, mais Esri requiert la dernière version d'ArcGIS. C'est pourquoi nous avons commencé à développer notre application sous Unity.

3 Présentation de la solution Mapbox

Le développement de la solution Mapbox s'effectue sous Android Studio. Le guide d'installation et de mise en place d'un premier projet est très bien expliqué. [1]
Après avoir initié notre projet en affichant un fond de carte, nous avons chargé un fichier GeoJSON depuis les ressources de l'application. [2] Bien que la composante verticale soit renseignée dans ce fichier, cette méthode d'import ne prend en compte que les coordonnées X et Y.
Pour ajouter la composante verticale, il semblerait qu'il faille rechercher dans le fichier GeoJSON les valeurs des coordonnées Z et les rajouter aux objets correspondants.

La contrainte majeure que présente Mapbox est la nécessité d'une connexion à Internet pour initialiser l'application, malgré le fait que l'on puisse télécharger des cartes en mode hors-ligne. [3]
Nous avons remarqué que si nous ouvrions l'application une première fois avec succès (donc avec une connexion internet), puis que nous désactivions le mode en ligne, si l'application est toujours dans le cache alors celle-ci peut être toujours utilisée (donc sans connexion internet). Il existe sûrement un moyen d'empêcher cet accès à Internet nécessaire pour lancer l'application, mais nous n'avons pas trouver de documentation suffisamment explicite à ce sujet.

4 Présentation de la solution Esri

L'entreprise Esri possède une plateforme de cartographie et d'analyse appelée ArcGIS, qui possède de nombreux outils intéressants pour la création d'un viewer BIM, notamment la modélisation de bâtiments 3D avec des géométries complexes et la récupération des attributs d'un objet. Le développement d'une application mobile peut se faire grâce à un SDK Android sous Android Studio. Des fonctionnalités intéressantes pour la modélisation 3D hors-ligne sur mobile ont été introduites début Avril 2019, avec la possibilité de créer des fichiers .mspk (*Mobile Scene Package*) à partir d'une *Scene* sous ArcGIS Pro. Ce fichier .mpsk sera ensuite importé sur un appareil. [4]

Il existe également une plateforme ESRI appelée AppStudio qui permet de programmer et de déployer des applications webs et mobiles, et qui possède donc les même spectres de possibilité que le SDK Android d'ArcGIS. Cependant, nous ne nous sommes pas assez penché dessus pour en fournir une documentation détaillée.

5 Présentation générale d'Unity

5.1 Le moteur de rendu 3D Unity

Unity est un moteur de rendu 3D. C'est-à-dire qu'il permet d'afficher des objets 3D de manière dynamique et performante. Le moteur de rendu est basé sur une technologie C++, mais le développement algorithmique se fait en langage C#.

5.2 Généralités sur le fonctionnement d'Unity

Unity est une plateforme de développement. Cette documentation présente l'utilisation du logiciel Unity version 2018.3 que vous pouvez télécharger via Unity Hub à l'adresse suivante : https://store.unity.com/. L'éditeur Unity est composé de plusieurs fenêtres:

- L'explorateur de fichiers ressources permet d'importer tout les fichiers qui seront utilisés par l'application. Deux dossiers sont présents à la création d'un projet Unity. Le premier dossier, nommé *Assets*, permet de stocker tout type de fichier (modèle 3D, scripts C#, images, fichiers texte, ...) qui seront employé par Unity. Le second dossier: *Package* sert uniquement à importer des librairies et extensions dans le projet courant. L'import de ces librairies s'effectue via l'onglet *Assets* en haut de la page.
- Une application peut être composée de plusieurs scène.La scène est la fenêtre de visualisation et de modification de la scène active. Notre application ne comporte qu'une seule scène. Les objets 3D peuvent être modifié selon les différents outils mis à disposition en haut à gauche de l'éditeur. Par exemple en faisant glisser les flèches selon les trois axes X,Y,Z avec l'outil *Move* sélectionné par défaut.
- La hiérarchie de la scène recense tout les objets présents dans celle-ci. L'import d'objet dans la scène se fait depuis l'explorateur de fichiers par un simple glisser-déposer. Ces objets sont appelés *GameObjects* selon la syntaxe d'Unity.
- La fenêtre d'inspection permet de voir les différents composants d'un *GameObject*. Ces composants sont appelés *Components* selon la syntaxe d'Unity. Il s'agit très généralement un script C#. Tout les *GameObjects* disposent d'un *Component* qui s'appelle *Transform*. Ce dernier configure la position, la rotation, et l'échelle de l'objet 3D.
- La fenêtre de rendu représente l'aspect final de l'application. En cliquant sur le bouton *Play* en haut de l'éditeur, l'éditeur compile une preview du projet qui est visible dans la fenêtre de rendu.
- La fenêtre de console permet d'afficher des erreurs de compilation et également des log. La fonction pour créer un log est : Debug.Log("Ceci est un message à destination de la console")

L'environnement d'exécution de l'éditeur Unity est à ne pas confondre avec l'environnement d'exécution Android. Par exemple, de nombreuses fonctions C# sont disponibles seulement dans l'éditeur car elles effectuent des actions sur le dossier *Assets*.

Figure 3: Présentation des différentes fenêtres de l'éditeur Unity

6 Présentation de l'application FIRE:BIM sous Unity

6.1 Hiérarchisation du projet Unity

L'application FIRE:BIM est composée de plusieurs *GameObjects*:

- d'une caméra.
- d'un modèle 3D (ici SketchUp)
- d'une interface, dotée d'une barre de recherche (zone de texte, menu déroulant, et bouton) et d'un encadré d'information.
- d'une source de lumière pour illuminer la scène.

6.2 Paramètres de construction de l'application mobile

Unity est un outil de développement multiplateforme, et permet donc de compiler le projet Unity sur différentes plateformes. Afin de créer une application à destination des terminaux Android, il est nécessaire d'installer le SDK Android au moment de l'installation d'Unity (sinon Unity vous proposera de le télécharger si vous voulez compiler sous Android).

Les paramètres de compilation se trouve dans l'onglet *File* > *Build Settings*. Une fenêtre apparaît et vous demande de choisir la plateforme de compilation.

Si vous avez choisi la plateforme Android, d'autres modifications sont nécessaires. Allez dans l'onglet *Edit* > *Project Settings*. Une fenêtre apparaît. Rendez vous dans la section *Player*.
Vous devrez renseigner dans cette section le nom du dossier qui contiendra votre application dans la mémoire du terminal mobile à la ligne *Package Name*. Il n'y a pas de restriction de nom.
Une autre option à modifier est la permission d'écriture ou *Write Permission* en anglais. Si la permission d'écriture est paramétrée sur *External (SD Card)*, l'application pourra accéder à des fichiers externes qui n'auront pas été compilé avec le projet Unity.

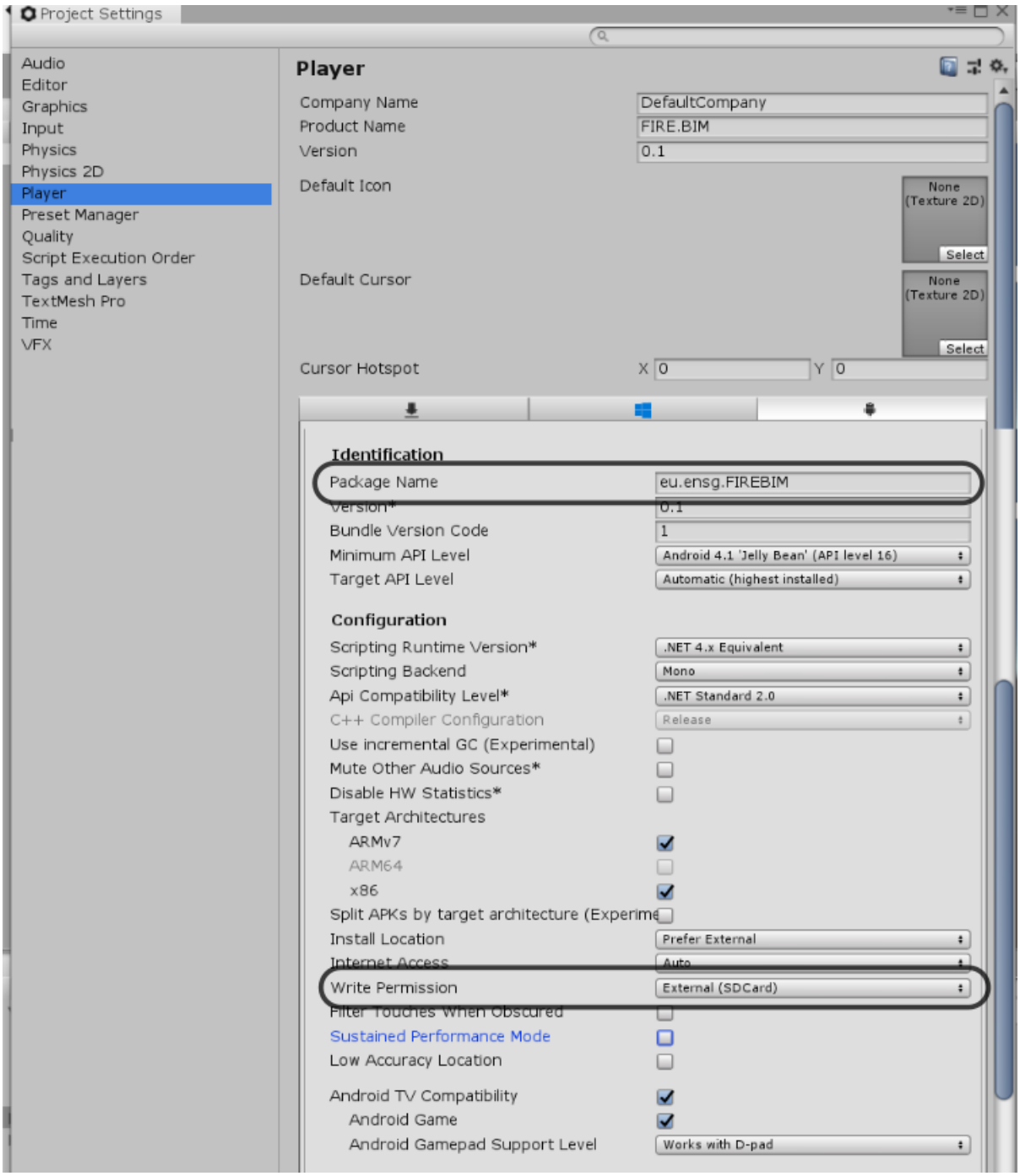

Figure 4: Fenêtre de paramétrage du projet

6.3 Compilation et installation de l'application

Après avoir fait les paramétrages sous Unity, il reste à installer l'application sur un appareil mobile Android (smartphone ou tablette). Pour cela, il faut brancher l'appareil à l'ordinateur, puis aller dans *File* > *Build and Run*. Unity vous demandera l'emplacement de sauvegarde du fichier APK qu'il va créer, et il suffira de choisir un emplacement sur votre PC. La compilation et l'installation de l'application se fera ensuite directement sur votre appareil, et l'application s'ouvrira automatiquement après son installation. Vous trouverez également une icône de l'application sur l'écran d'accueil de l'appareil.

6.4 Explication du fonctionnement de la caméra

Nous avons codé le déplacement de la caméra nous-mêmes. En effet, il n'existait pas d'exemple gratuit de déplacement de caméra qui correspondait à nos attentes. Avant d'expliquer le fonctionnement du code, nous allons parler du positionnement des objets dans Unity.

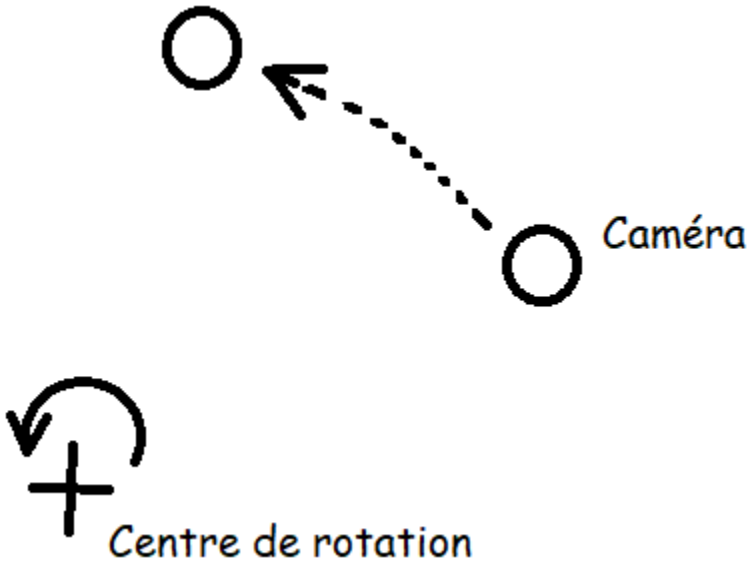

Figure 5: Rotation sur Unity

Lorsqu'un objet parent subit une transformation (rotation, translation, ou changement d'échelle) l'objet enfant subit les mêmes transformations et les coordonnées locales de l'enfant (par rapport au parent) restent inchangées. Ceci nous est très utile car nous ne souhaitons pas que la caméra tourne sur elle-même, mais qu'elle tourne autour d'un centre de rotation, comme le montre le schéma ci-dessus. À noter que la direction de prise de vue de la caméra ne doit pas tourner car sinon cela amènerait un tangage indésirable.

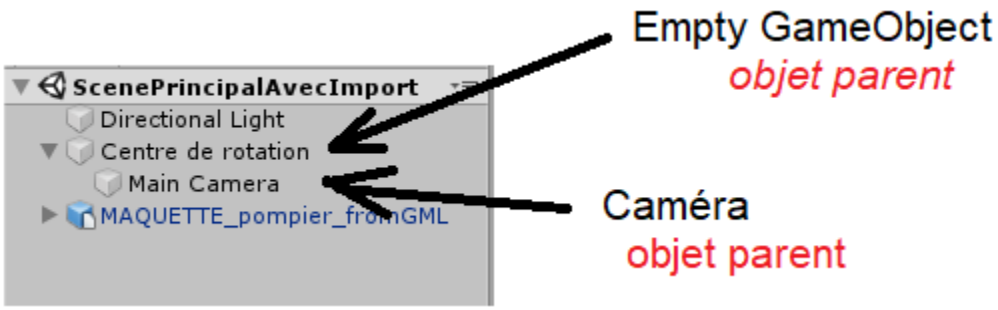

Figure 6: Objet parent, objet enfant

La figure ci-dessus décrit la hiérarchie de la scène du projet (cf. figure 3). La relation parent/enfant est assez intuitive vue dans cette hiérarchie. Il est également important de préciser que le centre de rotation est du type *Empty GameObject*, un objet qui a une position dans la modélisation mais aucune représentation.
Nous allons maintenant parler des parties importantes du code pour sa compréhension.

6.4.1 L'écran tactile

Nous avons choisi d'intégrer l'interface code/écran tactile dans le script contrôlant la caméra car il s'agit de la seule manière de la déplacer. Nous avons trois types de contrôles:

- appui court sur un objet: **centrage et lecture d'info**

- appui long et déplacement du doigt sur l'écran: **rotation de la caméra**
- pincement de l'écran: **zoom/dézoom**

Pour gérer les entrées nous avons utilisé la classe *Input* d'Unity avec la méthode *Input.GetTouch(id)* qui permet d'obtenir l'objet *Touch* par son id (qui représente son "ordre chronologique de touche") et la variable *Input.touchCount* qui donne le nombre de doigts touchant l'écran. Il est ensuite possible d'accéder à un attribut de *Touch* appelé *phase* qui permet d'effectuer des actions suivant les différents "types" de touche, par exemple *TouchPhase.Began*, *TouchPhase.Moved* et *TouchPhase.Ended*. Si le contrôle pour la rotation est relativement simple, les deux autres le sont un peu moins.

Pour le zoom, nous calculons la distance entre les positions des doigts d'abord à t puis à t+1, puis nous effectuons une différence sur ces deux distances qui nous donnera une valeur de zoom.

Pour la sélection d'objet, nous avons dû apporter une mesure de temps d'appui. Le temps d'appui permet de différencier la sélection de la rotation. Il est difficile d'exprimer le temps que dure un appui court car si le système a un temps de rafraîchissement long, nous pouvons ne pas détecter l'appui du tout, et au contraire sur un système puissant nous pouvons confondre la rotation et l'appui. Si la sélection semble ne pas fonctionner, revoir cette condition peut être un début de solution. Ensuite, il faut déterminer si nous appuyons ou non sur un objet. Pour cela *Camera.main.ScreenPointToRay(Vector3 position du point sur l'écran)* crée un objet *Ray* qui peut ensuite être utilisé par la fonction *Physics.Raycast(ray.origin, ray.direction, out hit)*. Cette dernière fonction renvoie un booléen qui peut être utilisé pour savoir si le rayon a touché un objet ou non. Le paramètre en sortie *hit* stock ensuite le *Collider* qui a interrompu le rayon.

6.4.2 Rotation

La rotation nous a posés beaucoup de problèmes au début, car nous utilisions les angles d'Euler qui se sont avérés peu pertinents. Pour mieux comprendre le problème, nous conseillons cette vidéo qui explique les rotations eulériennes. Une fois le problème compris, il a été simple d'y remédier avec la fonction *Quaternion.Euler* qui permet d'allier la puissance du Quaternion avec la simplicité des angles d'Euler. A noter qu'il faut toujours repasser par les angles d'Euler pour modifier le Quaternion.

6.4.3 Zoom

Pour comprendre le fonctionnement du zoom il faut se rappeler que les coordonnées peuvent être exprimées en local ou en global. Ainsi, plutôt qu'effectuer un déplacement en position absolue (global), nous l'effectuons en local, ce qui permet de s'affranchir de la position et de la rotation du centre de rotation. Nous avons hésité entre une translation (*camera.transform.Translate(new Vector3(0, 0, valeurZoom));*) et une augmentation de l'échelle (qui dilaterait ou contracterait la distance). La translation semble donner les meilleurs résultats. IL est à noter que la méthode de zoom peut être appelée dans d'autres scripts.

6.4.4 Translation du centre de rotation et centrage

L'action de déplacer le centre de rotation sur un objet a été découpé en deux actions, tout d'abord la détermination du nouveau centre (à la suite de la détection de collision) en accédant à *Collider.bounds.center*. Il faut également prendre en compte le fait que certains objets sont composés de plusieurs géométries, ce qui fait que le centre peut se retrouver entre ces géométries, ce qui est voulu.

Pour déplacer le centre de rotation nous avons d'abord modifié instantanément les coordonnées du centre de rotation pour qu'elles correspondent à celles du nouveau centre. Nous nous sommes rendus compte que cela pouvait être désorientant pour l'utilisateur et nous avons finalement utilisé la fonction *Vector3.MoveToward(Vector3 départ, Vector3 arrivé, float distanceMax)*. Cette fonction renvoie un Vector3 qui correspond à la position *départ* à laquelle nous avons ajouté la distance maximale dans la direction *départ* vers *arrivé*. Nous avons choisi de définir cette *distanceMax* en tant que la distance entre le départ et l'arrivée divisée par 4, ce qui fait que lorsque l'on déplace le centre de rotation, exactement 4 images seront affichées pendant la translation de la caméra, quelque soit la distance parcourue. Nous avons fait ce choix pour éviter d'avoir des durées de translation trop élevées qui ferait perdre du temps à l'utilisateur.

6.5 Explication du format des données

Les aspects géométrique et sémantique sont séparés dans Unity.
Nous avons donc créé un fichier SketchUp à partir du fichier CityGML qui représente l'aspect visuel des bâtiments (géométries, couleurs, ...). CityEditor, un plugin SketchUp, nous a permis cette conversion du format CityGML à un modèle SketchUp. Nous avons réussi à obtenir une clé d'essai en leur envoyant une demande. Cependant, le logiciel Unity ne conserve pas les données attributaires du fichier SketchUp en important son modèle 3D. Un fichier txt, contenant les attributs, a ainsi été créé à partir du même fichier CityGML.
Ce fichier CityGML a d'abord été converti en GeoJSON sous QGIS. Pour cela, ouvrez le fichier GML dans QGIS dans l'onglet *Couche* > *Créer une couche* > *Nouvelle couche shapefile*. Ensuite faites un clic-droit sur votre couche, qui est apparu en bas à gauche de la fenêtre de QGIS, puis choisissez *Enregistrez sous...*. Choisissez le format GeoJSON.

Ensuite à l'aide d'un script Python, un nouveau fichier texte au format JSON sera créé. Ce dernier regroupera les attributs de chaque objet. Il se trouve que les groupes du fichier SketchUp et du fichier txt ont les mêmes noms. Ces deux fichiers possèdent alors les mêmes noms pour les mêmes bâtiments, nous permettant ainsi de faire une jointure entre les deux, et donc d'accéder aux attributs des objets à partir du modèle 3D.

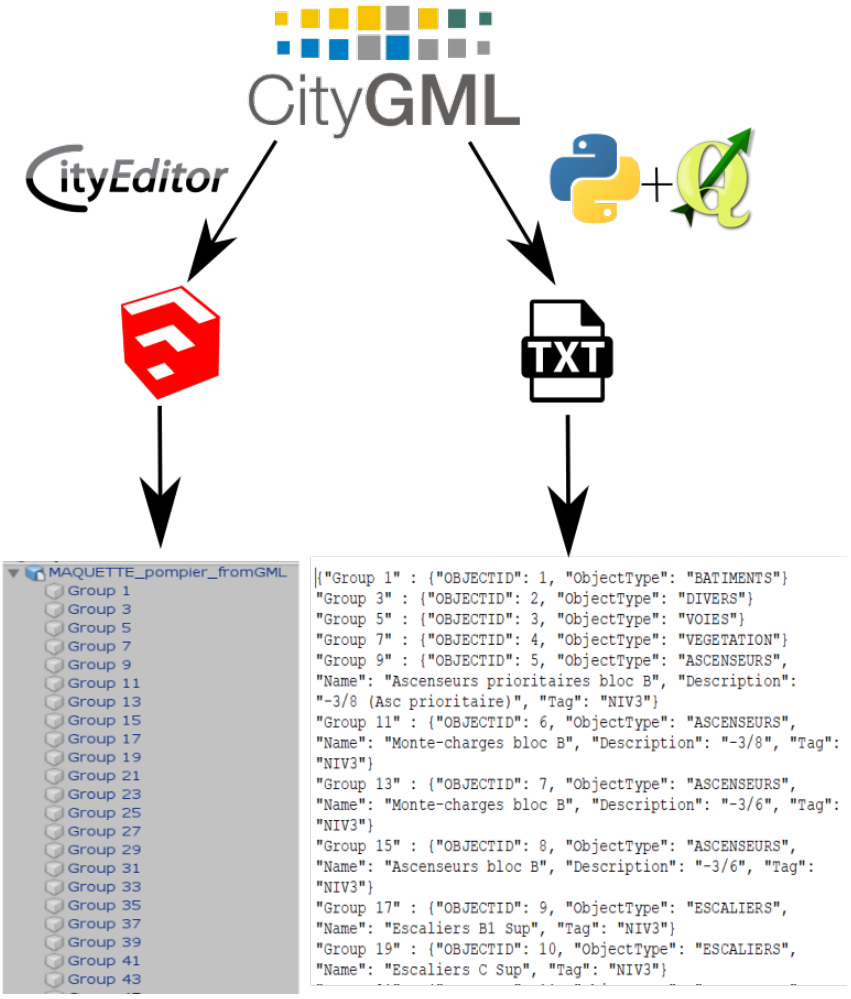

Figure 7: Processus de jointure entre la géométrie du modèle SketchUp et les attributs du fichier texte

Dans les faits, il est plus simple d'adapter le fichier texte aux noms des *GameObjects* pour que la jointure se passe bien.

6.6 Explication de la récupération des attributs

La description des objets étant stockées dans un fichier à part de la géométrie, nous disposons d'un script qui fait le lien entre les deux.
Lorsqu'un objet est sélectionné par une pression sur l'écran, on récupère le nom du *GameObject*. Avec ce nom, nous pouvons rechercher dans le fichier texte les attributs qui y sont associé et ainsi les récupérés.
Ceux-ci sont ensuite affichés dans l'interface via une fenêtre semi-transparente. On récupère donc les champs de la fenêtre et on actualise leurs valeurs.
Le nom permet également d'appeler la fonction sélectionner qui change la couleur de l'objet pour plus de lisibilité. Un seul objet peut être sélectionné à la fois via cette méthode.

6.7 Modification de l'interface

L'interface de l'application peut être enrichie avec de nouveaux panneaux contextuels, boutons, et autres. Pour ajouter un nouvel élément, il suffit de faire un clic-droit dans la fenêtre *Hierarchy*, et de sélectionner le type d'élément d'interface que vous voulez créer dans l'onglet *UI*.
Pour la modification d'interface, passer l'affichage en mode 2D rend plus intuitif les actions liées au positionnement.

6.7.1 Canvas

Tout les éléments d'interface sont liés à un élément plan: le *Canvas*.
S'il n'existe pas déjà, il est créé automatiquement avec le premier élément d'interface que l'on veut implémenter.Celui-ci, dans ses options possède un élément *RenderMode*. Cet élément défini la façon dont le canvas se comporte par rapport à notre vue. Ici il est réglé sur *ScreenSpace - Overlay*. C'est grâce à cela qu'il correspond à la superficie de l'écran. Tout les autres éléments d'interface sont placés par rapport à ce canvas.
On peut alors obtenir un aperçu du comportement de l'interface sur différentes résolution d'écran. Pour cela, sélectionnez en haut à gauche de la fenêtre *Game* la résolution que vous souhaitez. Vous pouvez également régler la taille du *Canvas* en cliquant au préalable sur un des objets qu'il contient. (*InputField*, *Dropdown*, *Search_Button*, *Panel*)

6.7.2 Position des éléments

Pour accéder à la modification de l'interface, vous pouvez cliquer sur le bouton *2D* en haut de la fenêtre *Scene*, puis sélectionnez par un double-clic le *Canvas* dans la fenêtre *Hierarchy*. Il est ensuite possible de modifier l'emplacement, et les dimensions des différents éléments composant l'interface grâce à des actions intuitives à la souris.
Tout les objets de l'interface, sauf le *Canvas*, ont une disposition relative et absolue qui sont modifiables respectivement par le *pivot* et les *anchors*.
Le déplacement, la rotation, et l'agrandissement/rétrécissement de l'interface s'effectuent grâce au *pivot*. Il suffit de sélectionner un objet, puis de modifier ses quatre coins, représentés par un cercle bleu.
Toutefois cette solution ne vaut que pour des résolutions fixes. Pour que l'interface créée s'adapte à toute les tailles, il faut utiliser les *anchors*. Elles permettent de positionner les éléments sur le *Canvas*, non pas à partir de l'origine du *Canvas* et des distances en pixel, mais via des pourcentages. De cette façon, quelque soit la taille de l'écran, la disposition de l'interface restera la même.
Les *anchors* des quatre coins de l'objet sont représentés par des triangles semi-transparents. Par défaut, ils sont joints au niveau du *pivot* de l'objet et ressemblent à une croix. [5]

6.7.3 Éléments d'interface et paramétrage

L'interface de l'application FIRE:BIM est composé d'une barre de recherche (*InputField*), d'un menu déroulant (*Dropdown*), d'un bouton de recherche (*Search_Button*), et d'un panneau contextuel (*Panel*).
Chaque élément d'interface possède des sous-élément. Par exemple, l'*InputField* possède un *PlaceHolder* qui représente la barre horizontale blanche, et un *Text_Input* qui représente le texte écrit dans la barre de recherche. De plus, chaque sous-élément possède ses propres paramètres qu'il est possible de changer via la fenêtre des options.

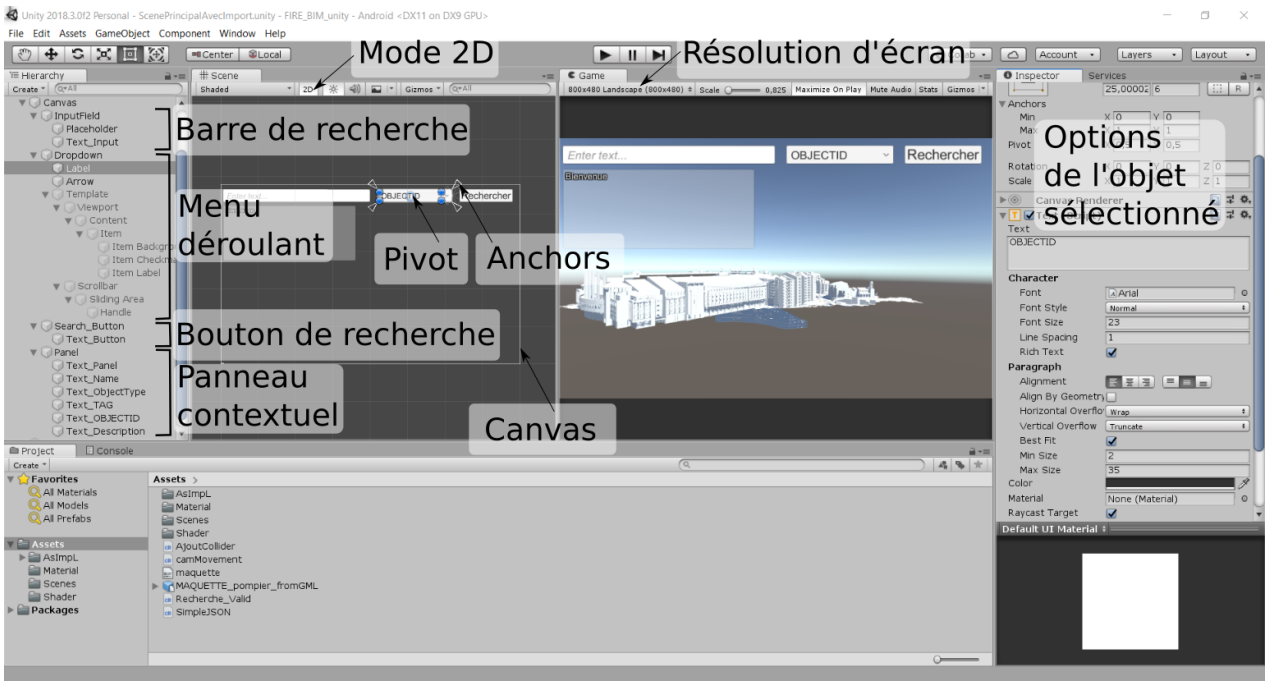

Figure 8: Présentation de l'interface de l'application dans Unity

Il est possible de modifier l'apparence de l'interface en changeant notamment l'image de fond (*Material*) de chaque élément. Il est également possible de mettre en forme le texte affiché à l'écran. L'option de texte la plus intéressante est *Paragraph* > *Best Fit*. Cette option permet d'adapter la taille de la police en fonction de la taille de l'écran, mais aussi en fonction de la longueur du texte afin qu'il ne dépasse pas de l'encadré de texte. L'option *Best Fit* nécessite de renseigner au préalable une taille de police minimale et maximale.

6.8 Fonctionnement de la barre de recherche

Les recherches se font via des requêtes sur le fichier texte contenant les attributs. Un menu déroulant permet de sélectionner l'attribut sur lequel on souhaite effectuer la recherche. La particularité de ce menu est que l'on peut y ajouter des options ce qui permet de rajouter ou retirer des attributs en cas d'évolution des données.

Les recherches se font via l'expression régulière suivante:
∧ (?i) + "texte utilisateur" + .*(?-i)

∧ Signifie que l'on recherche depuis le début de la chaîne de caractère ciblée.
(?i) (?-i) On ignore la casse entre ces deux balises.
. Indique un caractère quelconque.
* Indique un nombre quelconque incluant 0.
.* Indique que l'on recherche n'importe quel nombre de caractère quelconque.

L'expression recherche donc une chaîne de caractères de taille quelconque contenant à son début ce qui est demandé par l'utilisateur.
Elle met ensuite en surbrillance les objets qui correspondent à la recherche. Ceux-ci sont visible à travers les autres objets, au contraire de ceux simplement sélectionnés.

6.9 Surbrillance des éléments

La surbrillance des éléments consiste à changer la couleur du shader du GameObject choisi. Il faut opérer au cas par cas, car changer la couleur d'un matériau changerait la couleur de tout les *GameObjects* possédant le même matériau.
Pour la surbrillance des objets sélectionnés par une recherche, ces derniers sont visibles à travers d'autres objets de la maquette. Cet effet est permis grâce à un shader déjà existant. [6]

7 Amélioration de l'Application Unity

Nous avons implémenté les fonctionnalités les plus importantes du viewer BIM commandé par les sapeurs-pompiers de Monaco. Cependant il reste de nombreuses autres fonctionnalités à développer et quelques problèmes à résoudre.

7.1 Problèmes connus

Nous allons lister les problèmes que nous avons noté sur l'application et les éventuelles pistes de résolution que nous avons devisé :

- Lors d'un zoom poussé au maximum (vers la maquette, celle-ci disparaît. Il faut alors fermer l'application et la relancer.
- Si une recherche vide est effectuée, tout les objets sont sélectionnés.
  Pour corriger ça, il faut ajouter une condition non nulle à la recherche d'objet.
- La recherche est sensible aux accents. Il faudrait donc inclure les cas accentués et/ou modifier l'expression régulière
- Le zoom se comporte de façon erratique en cas de pincé trop écarté. Il est plus sensible sur les extrêmes (zoom maximum et minimum)

7.2 Chargement de fichiers 3D en *runtime*

Le processus d'import des données avec le modèle SketchUp nécessite que le modèle 3D soit présent lors de la compilation de l'application. Cela rend impossible l'utilisation d'un autre modèle 3D sans avoir à recompiler le projet Unity.
Pour pallier à ce problème, nous avons envisagé de charger un modèle 3D en *runtime*, c'est-à-dire pendant le fonctionnement de l'application. Nous avons pour cela trouvé un projet GitHub permettant de charger des objets 3D à partir d'un fichier .OBJ externe à l'application. Ce fichier .OBJ ne renseigne que la géométrie comme le modèle SketchUp, mais est plus lisible que ce dernier. Les attributs sont toujours renseignés grâce au fichier texte.
Nous avons donc converti notre modèle SketchUp en .OBJ grâce à SketchUp Pro et utiliser le projet GitHub disponible à l'adresse suivante : https://github.com/gpvigano/AsImpL

7.3 Interface utilisateur

Les proportions des différents éléments de l'interface peuvent être adaptées.
Les filtres demanderont une étude de la façon de les afficher. Nous avions imaginé un système d'icônes qui permettrait de remplir ce rôle.
La mise en place d'un menu déroulant qui permettrais d'afficher les type d'objets disponibles, associé à un outil de complétion automatique pour limiter les erreurs pourraient grandement améliorer l'expérience utilisateur dans l'utilisation de la barre de recherche.

7.4 Sélection des objets

La sélection d'objet pourrait être amélioré, en implémentant, par exemple, un bouton bloquant toute nouvelle sélection par un clic ou par une recherche. Il faut toutefois discuter de la pertinence d'un tel outil.

7.5 Implémentation et enregistrement de filtres

Pour faciliter l'utilisation de l'application aux pompiers, il serait envisageable de créer une fonction permettant l'implémentation et l'enregistrement de filtres sémantiques, par exemple pour afficher tous les ascenseurs ou tous les escaliers du bâtiment modélisé.

Cela pourrait augmenter l'efficacité des pompiers, surtout dans des moments critiques comme les interventions.
La mise en place de filtres peut s'inspirer du code de la barre de recherche.

7.6 Paramétrages de vues

Certaines fonctionnalités concernant les paramétrages de vues pourraient être développées à partir de notre application. Par exemple, lorsque l'on sélectionne un objet, les objets autour pourraient devenir transparents, donnant ainsi une vue plus claire sur l'objet. Un bouton pourrait également rendre le contexte transparent, n'affichant que le modèle 3D du bâtiment détaillé. Il serait également envisageable de paramétrer une navigation entre les différents étages du bâtiments à l'aide de boutons listant les différents numéros d'étages. Quand un étage serait sélectionné, les étages supérieurs seraient rendus transparents.

7.7 Barre de recherche

Plusieurs options sont manquantes sur la barre de recherche notamment au niveau de l'affichage des résultats.

Si les attributs sont présentement affichés dans un panel avec du texte, il est possible de mettre en place un scrollView. Cet élément d'UI Unity qui permettrait de naviguer dans les résultats et d'afficher l'intégralité d'entre eux.

La mise en place d'un outil de blocage de la sélection pourrait compléter l'utilisation de filtres lors des interventions.

Mettre en place un moyen d'afficher les ObjectTypes disponibles. L'idée serait d'itérer dans les ObjectTypes à l'initialisation de l'application pour que cela soit adaptatif.

References

[1] Documentation officielle de Mapbox. Premiers pas avec le SDK Android.

[2] Documentation officielle de Mapbox. Référence API pour le chargement de données GeoJSON.

[3] Documentation officielle d'Esri. Télécharger une carte hors-ligne.

[4] Documentation officielle d'Esri. Ouvrir un Mobile Scene Package avec ArcGIS Pro.

[5] Documentation officielle d'Unity. Introduction à la modification de l'interface.

[6] From Unify Community Wiki. Silhouette-Outlined Diffuse Shader.