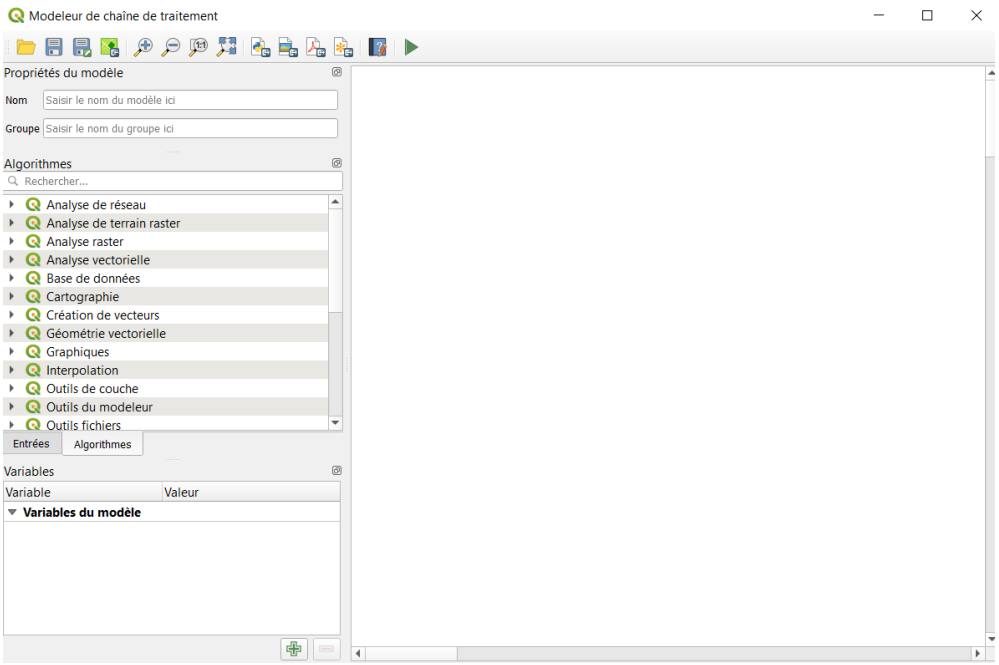
AUTOMATISATION DES GEOTRAITEMENTS SOUS QGIS 3

La programmation permet notamment d'automatiser certaines tâches répétitives. Or le domaine du SIG est précisément confronté à la répétition de certaines tâches ou à la réplication de certaines méthodologies permettant d'obtenir les résultats souhaités. C'est pourquoi Python est souvent utilisé pour automatiser des tâches sous QGIS. Sans surprise, on va surtout chercher dans un SIG à automatiser les géotraitements. Ce TD se focalise donc logiquement sur cet aspect. Néanmoins, l'automatisation est tellement courante dans le domaine des SIG que celle-ci n'est pas réservée à celles et ceux maitrisant des langages de programmation. En effet, de nombreux SIG proposent des « Model Builder » permettant à un public plus large d'automatiser certaines tâches et constituent une bonne alternative à la programmation. QGIS propose justement un « modeleur graphique » que l'on va donc présenter succinctement ici. Faut-il privilégier les « Model Builder » ou la programmation pour automatiser des tâches sous un SIG ou inversement ? Je n'ai pas la réponse. Je n'utilise jamais les « Model Builder », la programmation offrant plus de libertés que les « Model Builder », mais il n'y a pas de bonnes réponses à cette question. Ce sera à vous de voir en fonction des usages. ***Dans ce TD, on va chercher à réaliser une tâche courante du géomarketing, faire un calcul de potentiel.*** Pour cela, on dispose d'une couche de magasins (Leclerc), d'une zone géographique (Departements) et d'un fichier de potentiel économique (PIB).
Les données utilisées sont disponibles ici : http://sergelhomme.fr/data/Donnees.zip

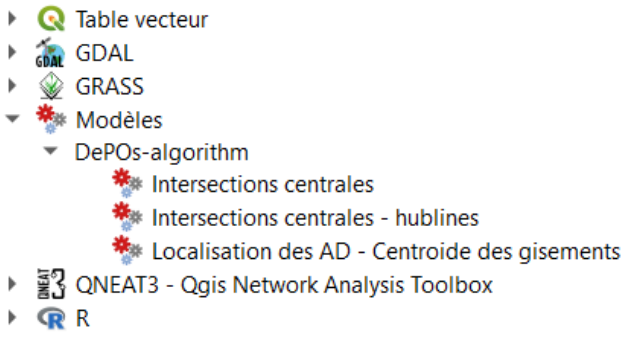
1) AUTOMATISATION SANS PROGRAMMATION : LES MODEL BUILDER (le modeleur graphique QGIS)

Le modeleur graphique de QGIS est disponible depuis la barre de menu dans l'onglet « Traitement -> Modeleur graphique… ». Ce modeleur se compose en haut d'icônes « Outils ». A gauche, on retrouve généralement trois panneaux : le panneau du nom et du groupe ; le panneau des algorithmes et des paramètres ; le panneau variable. A droite, c'est l'espace graphique du modeleur.



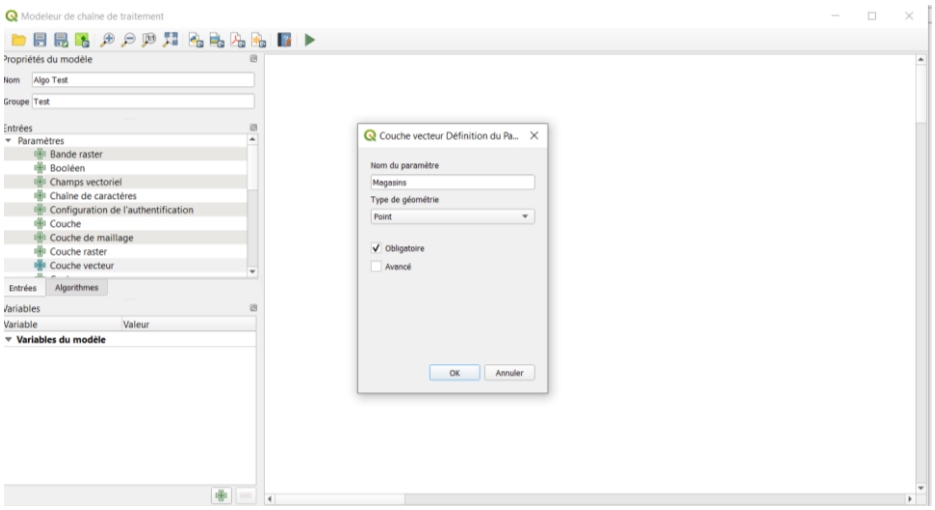
Le modeleur graphique de QGIS et ses panneaux.

On retrouve dans le panneau « Algorithmes », l'ensemble des algorithmes disponibles dans QGIS depuis l'onglet « Traitements -> Boite à outils… ». Lorsque vous aurez donné un nom et un groupe à votre modeleur (en haut à gauche sous les icônes), et une fois ces modifications enregistrées, vous pourrez charger le modèle dans la boite à outils en cliquant sur les roues dentées en haut à gauche de la boite à outils, puis sur « Ajouter un modèle à la boite à outils ». Les modèles seront alors directement disponibles dans l'onglet « Modèles ». Un exemple ci-dessous.

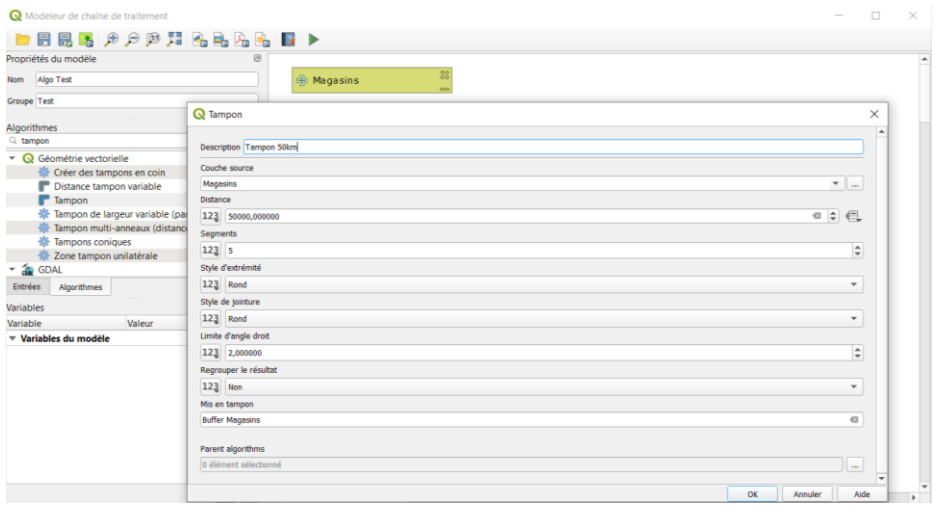


Les modèles développés et sauvegardés dans la boite à outils.

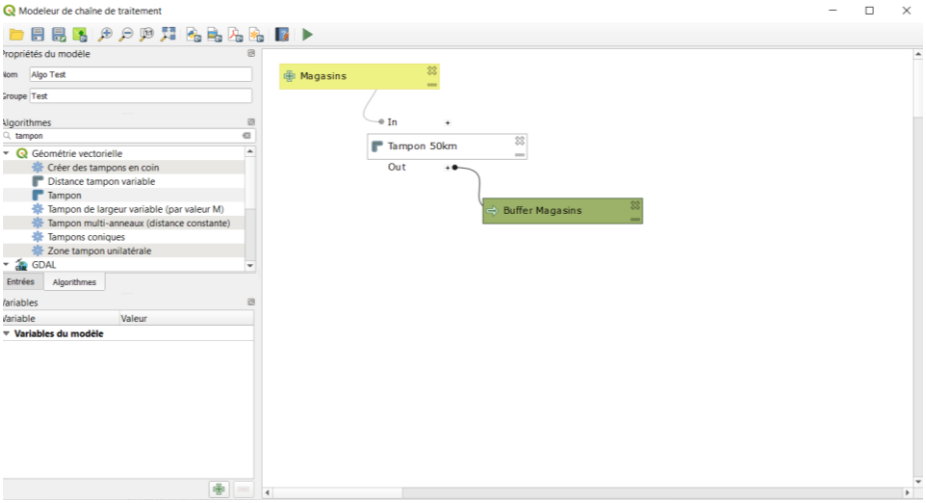
Pour utiliser le modeleur, c'est très simple, on cherche l'algorithme que l'on veut utiliser. Pour cela, le plus simple, c'est de connaitre son nom. Par exemple, si on veut faire des buffers de 50km autour des magasins Leclerc, dans QGIS l'algorithme s'appelle « Tampon ». On tape alors son nom dans l'espace de recherche « Algorithmes », puis à l'aide d'un cliquer-glisser, on le place dans l'interface graphique du modeleur, puis les éléments à remplir pour faire tourner l'algorithme apparaissent. Pour une zone tampon, il faut une couche source en entrée. On peut indiquer directement l'emplacement d'un fichier, mais ici en théorie ce n'est pas l'objectif, puisque le modeleur doit « en théorie » servir à répliquer des calculs sur différents fichiers… Il faut donc indiquer au modeleur que l'on va utiliser une couche entrée par l'utilisateur. On clique donc ici sur « Annuler » et on va définir une couche en entrée (« Magasins »), puis dans un second temps on configurera l'algorithme. Pour définir une couche en entrée, il faut aller dans « Entrées », puis faire cliquer-glisser dans le modeleur graphique l'élément « Couche vecteur » et enfin remplir une boîte de dialogue très simple.



Boite de dialogue d'une couche vecteur en entrée.

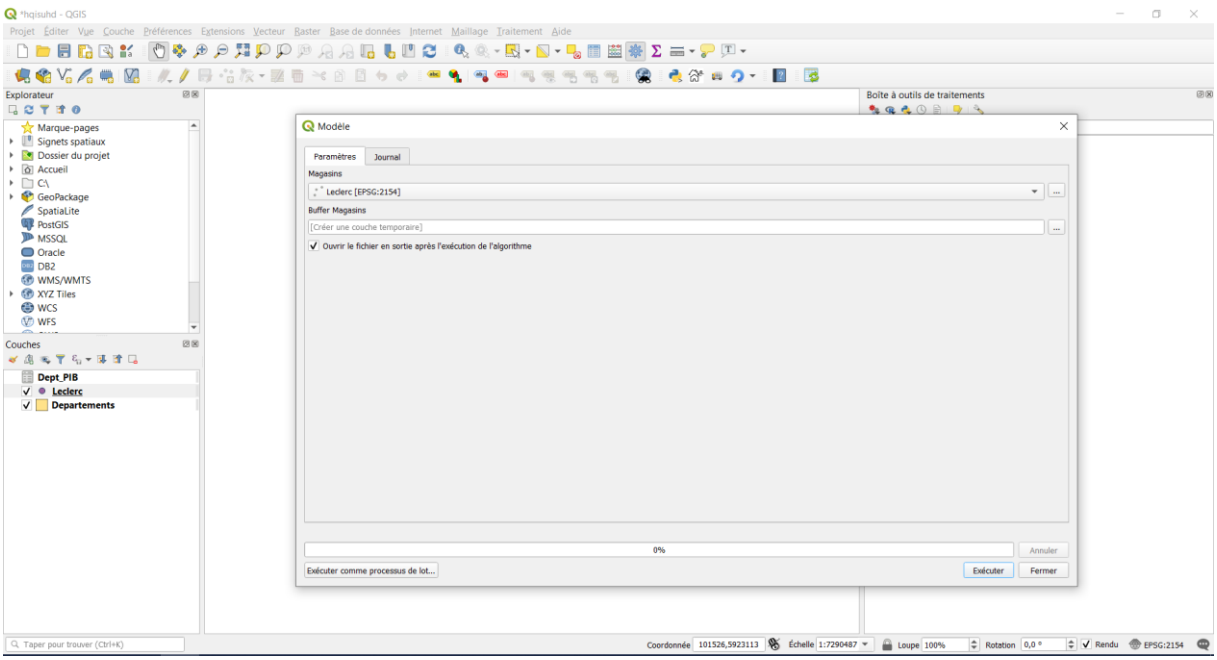


Boite de dialogue de l'algorithme « Tampon ».



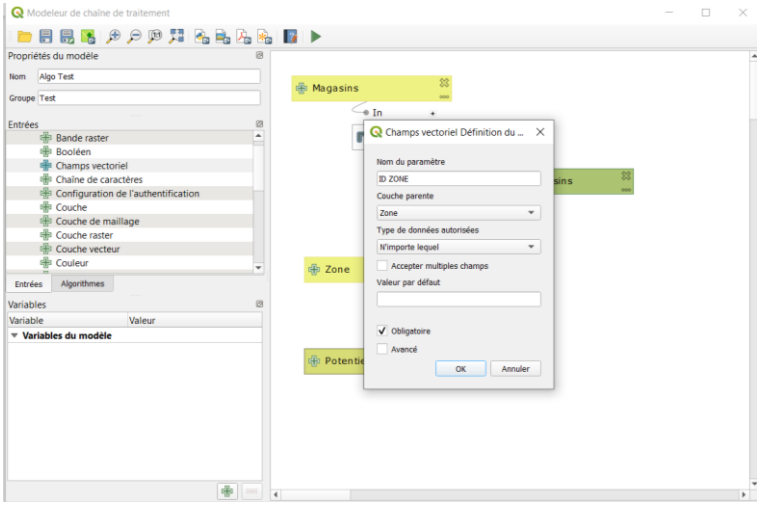
Le modeleur graphique avec une couche en entrée « Magasins », un tampon « Tampon 50km » générant une couche « Buffer Magasins ».

On peut alors cliquer sur le triangle vert d'exécution présent dans la barre d'outils. Cela exécute le traitement en simulant ce que l'utilisateur devra remplir pour exécuter l'algorithme. Dans les modèles de la boîte à outils, l'utilisateur pourra exécuter le modèle depuis le groupe « Test » et en choisissant le traitement « Algo Test » (ce qui est rentré en haut à gauche du modeleur graphique). À tout moment, vous pouvez modifier un élément du graphique en double-cliquant dessus. Vous pouvez supprimer un élément en cliquant sur la croix.

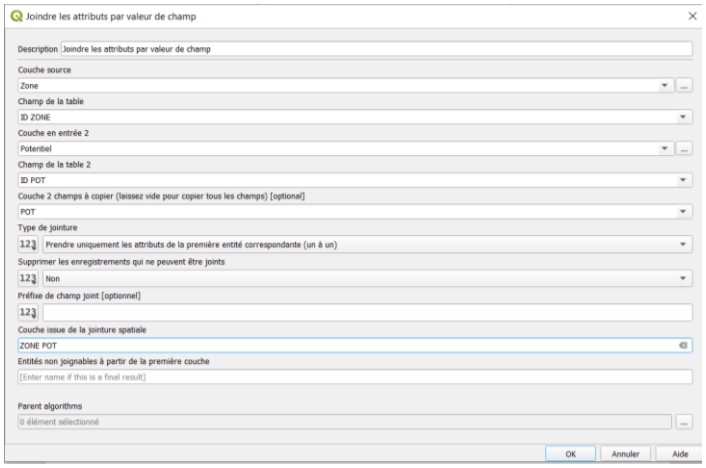


La boîte de dialogue à remplir pour exécuter le modèle de la zone tampon.

On peut aussi avoir recours à des algorithmes qui ne sont pas stricto sensu du géotraitement. On peut par exemple faire des jointures attributaires ou calculer de nouveaux champs à partir du modeleur. Ainsi, on peut chercher à joindre la couche PIB avec la couche « Departements », puis calculer la superficie de chaque département, dans le but d'intersecter nos départements avec les buffers des magasins pour faire ensuite une classique étude de potentiel. Pour effectuer la jointure, il convient de définir des champs sur lesquels s'appuyer. Pour cela, après avoir défini dans le modeleur les deux couches d'entrées nécessaires à la jointure (que l'on peut appeler avec des termes génériques « Zone » et « Potentiel »), dans le panneau « Entrées » il faut faire cliquer-glisser « Champs vectoriel » puis remplir les informations demandées. On peut alors créer un « ID ZONE », un « ID POT » et même rajouter un « ID MAGASINS » pour la couche précédente. On peut alors faire la jointure attributaire avec l'algorithme « Joindre les attributs par valeurs de champ ». A noter que si l'on souhaite copier uniquement le champ « PIB », il faudra aussi créer ce champ. C'est ce que l'on va faire ici, on va l'appeler « POT ». On retrouve bien ces champs définis dans la boite de dialogue de l'algorithme de jointure attributaire.

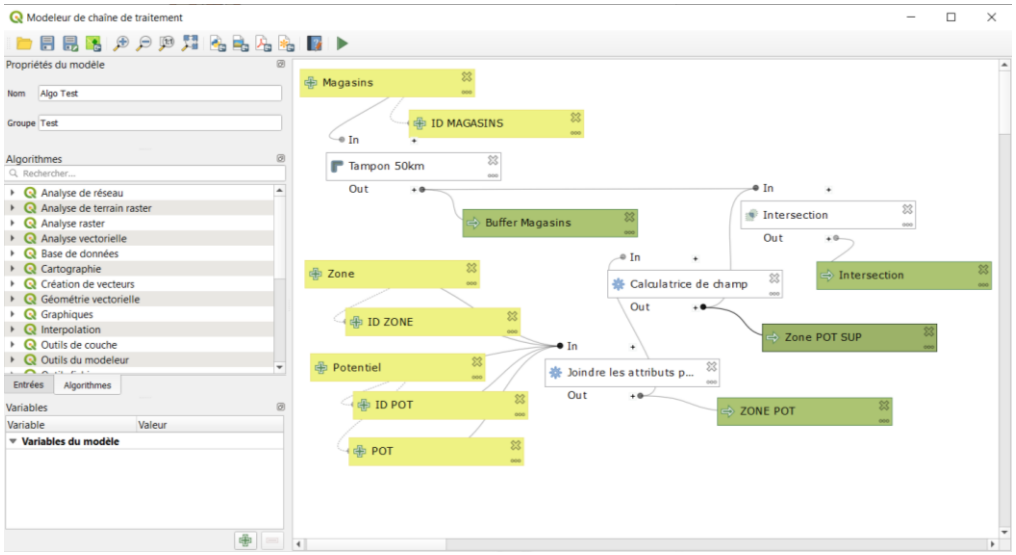


Boite de dialogue de définition de « Champs vectoriel ».



Boite de dialogue permettant d'effectuer une jointure attributaire.

A noter que dans cet algorithme de base de QGIS, le lien n'est pas fait entre les couches et les champs, utiliser cet algorithme est donc assez pénible alors même que nous n'avons que quatre champs et trois couches… A partir de la couche résultat « ZONE POT », on peut calculer la superficie des départements en utilisant l'algorithme « Calculatrice de Champs » et la fonction « $area ». A noter que la couche à utiliser s'appelle dans ce nouvel algorithme « 'Couche issue de la jointure spatiale' from algorithm 'Joindre les attributs par valeur de champ' » et non tout simplement « ZONE POT ». C'est assez lourd, car on n'a pas accès aux noms des couches créées en mémoire dans cet algorithme. Ensuite, on peut faire l'intersection entre la zone tampon et la jointure.

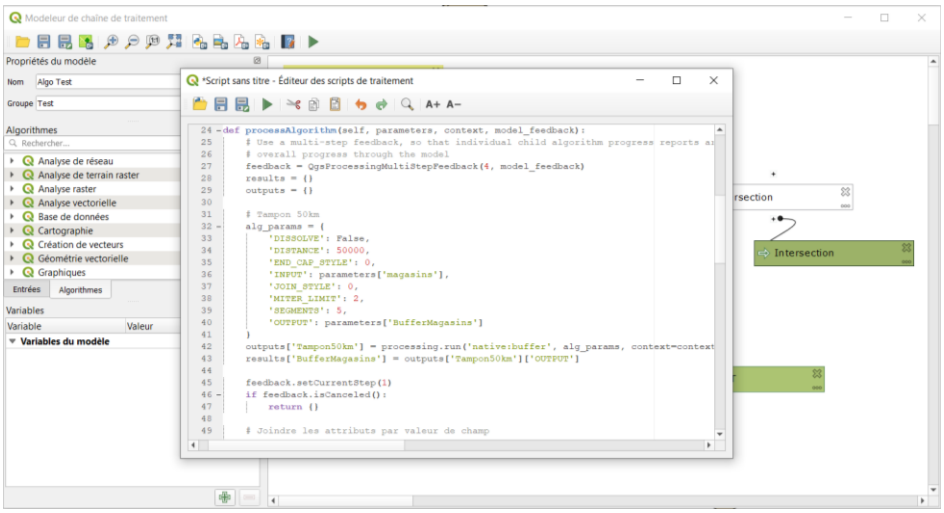


Au final on peut obtenir ce modèle.

Pour chaque objet de la couche « Intersection », il est désormais possible de calculer leur superficie, en mettant celle-ci en regard de la superficie du département correspondant et de son PIB. Il est donc en théorie possible d'estimer le PIB potentiel pour chaque objet à l'aide d'une règle de trois. Néanmoins, le champ « POT » n'est pas exploitable par la calculatrice de champs, il faudrait donc connaitre le nom du champ choisi par l'utilisateur pour effectuer ce simple calcul. On ne peut donc pas réaliser ce calcul, sauf à choisir des solutions non pertinentes pour une automatisation (dire à l'utilisateur d'utiliser un nom de champ précis pour le potentiel). A noter que si l'on souhaite faire une jointure spatiale entre les buffers pour étudier la concurrence, cela se fait très bien. Néanmoins, si on souhaite exploiter les champs calculés à partir de la jointure spatiale, cela sera de nouveau compliqué. Les algorithmes de regroupement ou d'agrégation auront la même difficulté à gérer les nouveaux champs. En cherchant bien, on peut souvent contourner ces problèmes, mais cela devient vite pénible quand même. Le modeleur évolue vite et peut-être que la version 4 de QGIS améliorera cela (à noter que d'autres modeleurs graphiques gèrent beaucoup mieux ces difficultés). Enfin, la petite icone « aide » du modeleur graphique permet de décrire à l'utilisateur comment fonctionne le modèle développé. Pour nous, ça se termine là.

2) LES BASES DU GEOTRAITEMENT AVEC PYTHON

Le modeleur graphique est intéressant, mais présente assez vite quelques limites qu'avec un langage de programmation on pourra toujours surmonter. Pour cela, il faut maitriser les bases du géotraitement avec PyQgis. Pour les personnes observatrices, il y a une petite icone Python dans le modeleur qui permet de récupérer le code Python exécuté par le modeleur. Ainsi, si vous arrivez à faire une séquence avec le modeleur, vous avez juste en théorie à récupérer le code produit. Certains commentaires reprennent le nom donné aux algorithmes utilisés. En dessous de ces commentaires la structure est souvent la même : alg_params = {} ; puis outputs[''] = processing.run() ; puis results[] = outputs[][]. C'est cela qu'on peut appeler la base du géotraitement dans QGIS avec Python…



Exemple d'un code Pyhton issu d'un modèle graphique.

3) REPRODUCTION DU MODEL BUILDER AVEC PYTHON

Pour exécuter les algorithmes de géotraitement de QGIS avec Python, il faut utiliser la méthode « processing.run() ». Tous les algorithmes disponibles dans la boite à outils sont utilisables avec PyQGIS. Il est souvent nécessaire d'identifier l'algorithme à l'aide de la méthode « processingRegistry() ». A noter que lorsque l'on exécute un algorithme avec QGIS dans le journal, on trouve aussi les paramètres du code Python utilisés lors de son exécution. La méthode « processing.algorithmeHelp() » est quoi qu'il en soit très utile pour utiliser les paramètres de l'algorithme.

```
>>> QgsApplication.processingRegistry().algorithms()
>>> len(QgsApplication.processingRegistry().algorithms())
>>> QgsApplication.processingRegistry().algorithms()[3]
>>> QgsApplication.processingRegistry().algorithms()[3].displayName()
>>> QgsApplication.processingRegistry().algorithms()[3].id()
>>> processing.algorithmHelp('qgis:buffer')
```

Tout l'enjeu est alors de savoir utiliser les paramètres pour faire fonctionner l'algorithme souhaité grâce à cette aide. Ici, pour un buffer de 50km, on peut utiliser le paramètre « DISTANCE », le paramètre « INPUT » permet de définir le nom de la couche en entrée (ou l'adresse d'un fichier), le paramètre « OUTPUT » défini le résultat, généralement placé en mémoire (ne pas oublier les « : ») ou sur une adresse précise. Le paramètre « DISSOLVE » est très important, il permet de regrouper ou non les buffers. En effet, par défaut, si l'algorithme opère un regroupement, ce n'est pas nécessairement toujours ce que l'on souhaite obtenir. Ensuite, on utilise « processing.run() » simplement et on ajoute le résultat au projet.

```
>>> param = {'INPUT' : 'Leclerc',
      'DISTANCE' : 50000,
      'DISSOLVE' : False,
      'OUTPUT' : 'memory:'}
>>> buf50k = processing.run('qgis:buffer', param)
>>> QgsProject.instance().addMapLayer(buf50k['OUTPUT'])
```

Par défaut les couches en sortie s'appellent « output ». On peut simplement utiliser la méthode « setName() » pour modifier cela.

```
>>> buf50k['OUTPUT'].setName('Buffer Magasins')
```

Ensuite, il n'y a plus qu'à enchainer les géotraitements ou les autres types de traitements d'ailleurs. Pour la jointure attributaire ce sera « native:joinattributestable », pour la calculatrice de champ « qgis:fieldcalculator » (pour cela on préférera en Python utiliser les éléments présentés dans le premier TD). Pour l'intersection, ce sera « native:intersection ». On trouvera ci-dessous un exemple pour effectuer la jointure attributaire.

```
>>> param = {'INPUT' : 'Departements',
          'INPUT_2' : 'Dept_PIB',
          'FIELD' : 'CODE_DEPT',
          'FIELD_2' : 'Num',
          'FIELDS_TO_COPY' : 'PIB',
          'OUTPUT' : 'memory:'
         }
>>> zonepot = processing.run('native:joinattributestable', param)
>>> QgsProject.instance().addMapLayer(zonepot['OUTPUT'])
>>> zonepot['OUTPUT'].setName('ZONE POT')
```

On peut ainsi sans trop de difficultés faire ce que l'on veut avec les algorithmes QGIS en Python. Il ne reste plus qu'à communiquer avec l'utilisateur pour récupérer les informations que l'on souhaite obtenir. En attendant de créer de belles interfaces, on peut utiliser des « QInputDialog.getText() » ou des « QInputDialog.getItem() ». On obtient alors le code suivant sans trop de difficultés et faire le calcul final de potentiel. On pourra même chercher à agréger les résultats pour obtenir une synthèse pour chaque magasin. A noter qu'il est possible de ne pas afficher toutes les couches, car ça devient vite lourd…

```
#Recuperation des infos
nomMag = QInputDialog.getText(None,"Mag","Entrez le nom de la couche Magasins :")
nomZone = QInputDialog.getText(None,"Zone","Entrez le nom de la couche Zone :")
nomPot = QInputDialog.getText(None,"Pot","Entrez le nom de la couche Potentiel :")
coucheMag = QgsProject.instance().mapLayersByName(nomMag[0])[0]
coucheZone = QgsProject.instance().mapLayersByName(nomZone[0])[0]
couchePot = QgsProject.instance().mapLayersByName(nomPot[0])[0]
providerMag = coucheMag.dataProvider()
providerZone = coucheZone.dataProvider()
providerPot = couchePot.dataProvider()
```

```
champsMag = providerMag.fields().names()
champsZone = providerZone.fields().names()
champsPot = providerPot.fields().names()
idmag = QInputDialog.getItem(None, "Couche Mag", "Champ Identifiant", champsMag)
idzone = QInputDialog.getItem(None,"Couche Zone"," Champ Identifiant ", champsZone)
idpot = QInputDialog.getItem(None,"Couche Pot"," Champ Identifiant ", champsPot)
pot = QInputDialog.getItem(None,"Couche Pot"," Champ Potentiel ", champsPot)
#Tampon
param = {'INPUT' : nomMag[0],
      'DISTANCE' : 50000,
      'DISSOLVE' : False,
      'OUTPUT' : 'memory:'}
buf50k = processing.run('qgis:buffer', param)
QgsProject.instance().addMapLayer(buf50k['OUTPUT'])
buf50k['OUTPUT'].setName('Buffer Magasins')
#Jointure
param = {'INPUT' : nomZone[0],
         'INPUT_2' : nomPot[0],
         'FIELD' : idzone[0],
         'FIELD_2' : idpot[0],
         'FIELDS_TO_COPY' : pot[0],
         'OUTPUT' : 'memory:'
        }
zonepot = processing.run('native:joinattributestable', param)
QgsProject.instance().addMapLayer(zonepot['OUTPUT'])
zonepot['OUTPUT'].setName('ZONE POT')
#Nouveau champ Superficie
param = {'INPUT' : zonepot['OUTPUT'],
         'NEW_FIELD': True,
         'FIELD_LENGTH': 10,
         'FIELD_NAME': 'Superficie',
         'FIELD_PRECISION': 3,
         'FIELD_TYPE': 0,
         'FORMULA': '$area / 1000000',
         'OUTPUT': 'memory:'
        }
zonepotsup = processing.run('qgis:fieldcalculator', param)
QgsProject.instance().addMapLayer(zonepotsup['OUTPUT'])
zonepotsup['OUTPUT'].setName('ZONE POT SUP')
#Intersection
param = {'INPUT': buf50k['OUTPUT'],
         'OVERLAY': zonepotsup['OUTPUT'],
         'OUTPUT':  'memory:'
       }
intersection = processing.run('native:intersection', param)
QgsProject.instance().addMapLayer(intersection['OUTPUT'])
intersection['OUTPUT'].setName('Intersection')
#Calcul final
param = {'INPUT': intersection['OUTPUT'],
         'NEW_FIELD': True,
         'FIELD_LENGTH': 10,
         'FIELD_NAME': 'Estim_PIB',
         'FIELD_PRECISION': 3,
         'FIELD_TYPE': 0,
         'FORMULA': '$area / 1000000 / superficie * '+ str(pot[0]),
         'OUTPUT': 'memory:'
        }
final = processing.run('qgis:fieldcalculator', param)
QgsProject.instance().addMapLayer(final['OUTPUT'])
final['OUTPUT'].setName('Final')
```

```
#Exercice final : Faire un regroupement (agregation) pour avoir le resultat agrege pour chaque magasin
```

4) LE GEOTRAITEMENT PYQGIS

Savoir utiliser les algorithmes de QGIS avec Python est très pratique. Néanmoins, c'est loin d'être la solution à tous les problèmes que l'on va se poser. Ici nous travaillons sur un « cas d'école », où par exemple le nombre d'objets géographiques est raisonnable. Or, parfois, les géotraitements peuvent être longs. Des difficultés peuvent alors apparaitre. En effet, si une commande Python est trop longue à s'exécuter, cela peut générer des erreurs. Cela peut se modifier en demandant à « Python d'être patient » en attendant que QGIS réalise le géotraitement. Néanmoins, cela pose une question centrale, le code précédent repose sur des traitements QGIS qui peuvent être considérés comme extérieurs à l'environnement Python et votre code Python n'est ainsi pas « indépendant ». C'est pour cela qu'il peut être pertinent de savoir réaliser des géotraitements en PyQgis. Dans ce cadre, il faut être en mesure de manipuler les géométries. Pour accéder à la géométrie, cela fonctionne un peu de la même façon qu'avec les attributs, mais cette fois-ci on utilisera la méthode « geometry() », une fois que l'on a récupéré un objet.

```
>>> layer = iface.activeLayer()
>>> objet = layer.selectedFeatures()[0]
>>> print(objet.attributes())
>>> print(objet.geometry())
```

Les géométries sont néanmoins un peu moins faciles à manipuler, car il existe différents standards pour les manipuler. Le WKT est très connu, mais il est aussi possible d'utiliser des fonctions particulières pour chaque type de géométries (Point, Polyligne, Surface).

```
>>> print(objet.geometry().asWkt())
>>> print(objet.geometry().asPoint())
>>> print(objet.geometry().asPoint().x())
```

Une fois une géométrie récupérée, on peut la manipuler pour l'utiliser dans une autre couche.

```
>>> vln = QgsVectorLayer("Point?crs=epsg:2154", "Nodes", "memory")
>>> prn = vln.dataProvider()
>>> prn.addAttributes( [QgsField("Nom", QVariant.String)])
>>> QgsProject.instance().addMapLayer(vln)
>>> fet = QgsFeature()
>>> fet.setGeometry( objet.geometry() )
>>> fet.setAttributes( [ objet.attributes()[0] ])
>>> vln.startEditing()
>>> prn.addFeatures( [ fet ] )
>>> vln.commitChanges()
```

Une fois que l'on sait manipuler de la géométrie, on peut faire des changements « à la main ». Principalement, on créera des points de type « QgsPointXY » et on pourra utiliser en fonction des besoins les méthodes « fromPointXY() » , « fromPolygonXY() » ou « fromPolylineXY() ». Mais on peut aussi utiliser le WKT avec la méthode « fromWkt() ».

```
>>> point = QgsPointXY(600000, 6500000)
>>> fet = QgsFeature()
>>> fet.setGeometry( QgsGeometry.fromPointXY(point) )
>>> fet.setAttributes([ " ? " ])
>>> vln.startEditing()
>>> prn.addFeatures( [ fet ] )
>>> vln.commitChanges()
```

Enfin, on pourra effectuer des géotraitements sur ces géométries, comme des buffers.

```
>>> geom = objet.geometry()
>>> buf = geom.buffer(500000,5) #distance puis precision
>>> buf2 = fet.geometry().buffer(500000,5)
>>> layer = QgsVectorLayer("Polygon?crs=epsg:2154", "Surface", "memory")
>>> QgsProject.instance().addMapLayer(layer)
>>> fet2 = QgsFeature()
>>> fet2.setGeometry( buf )
>>> fet3 = QgsFeature()
>>> fet3.setGeometry( buf2 )
>>> layer.startEditing()
>>> layer.addFeatures( [ fet2, fet3 ] )
>>> layer.commitChanges()
```

Ou encore des intersections.

```
>>> buf.intersects(buf2)
>>> intersec = buf.intersection(buf2)
>>> fet4 = QgsFeature()
>>> fet4.setGeometry( intersec )
>>> layer.startEditing()
>>> layer.addFeatures( [ fet4 ] )
>>> layer.commitChanges()
```

A noter que ces traitements sont très souvent conçus pour traiter des objets et non des listes d'objets. Si par exemple on veut regrouper les départements de la France métropolitaine, il faudra faire son propre algorithme qui prend les éléments un par un et les regroupe.

```
#Recuperation des geometries des departements
couche = QgsProject.instance().mapLayersByName('Departements')[0]
provider = couche.dataProvider()
fit = provider.getFeatures()
feat = QgsFeature()
tablegeom = []
while fit.nextFeature(feat):
      tablegeom = tablegeom + [feat.geometry()]

#Execution des regroupements
geom1 = tablegeom[0]
for i in range(len(tablegeom)-1):
    comb = geom1.combine(tablegeom[i+1])
    geom1 = comb

#Affichage du resultat
layer = QgsVectorLayer("MultiPolygon?crs=epsg:2154", "France", "memory")
fet = QgsFeature()
fet.setGeometry(comb)
layer.startEditing()
layer.addFeatures( [ fet ] )
layer.commitChanges()
QgsProject.instance().addMapLayer(layer)
```

A noter que cette façon de faire (prendre les départements un à un selon un ordre arbitraire) n'est peut-être pas la plus pertinente pour obtenir le regroupement des départements français. C'est pour cela qu'il est utile de se forger **« une culture des traitements géométriques »**. **Les SIG**, on l'oublie souvent lorsqu'on en a juste une utilisation « clic-bouton », **ce sont des mathématiques, principalement de la géométrie**.