Chaînes de traitement et procédures pour l'exploitation de données spatialisées par un simulateur multi-agents.

Quentin Baduel, **Jean Le Fur** et **Sylvain Piry**

avril 2009

Résumé

Ce document présente l'enchaînement et les procédures à suivre pour la récupération et l'implémentation de supports spatialisés dans la plate-forme de simulation SimMasto.

La chaîne gère des données issues de SIG (shapefiles), de cartes numérisées à partir d'images géoréférencées, d'applications type GoogleEarth ou de grilles théoriques.

Le rapport présente un schéma synoptique général de la chaîne de traitement associé à la description détaillée de chacun des modules et des modalités de leur réalisation.

Ce travail a été réalisé au CBGP dans le cadre du projet SimMasto et du stage de formation de DUT informatique de Q.Baduel.

Centre de Biologie et de Gestion des Populations, CS30016, 34988 Montferrier sur Lez
www.montpellier.inra.fr/CBGP/

Table des matières

Présentation et schéma synoptique de la chaîne de traitement

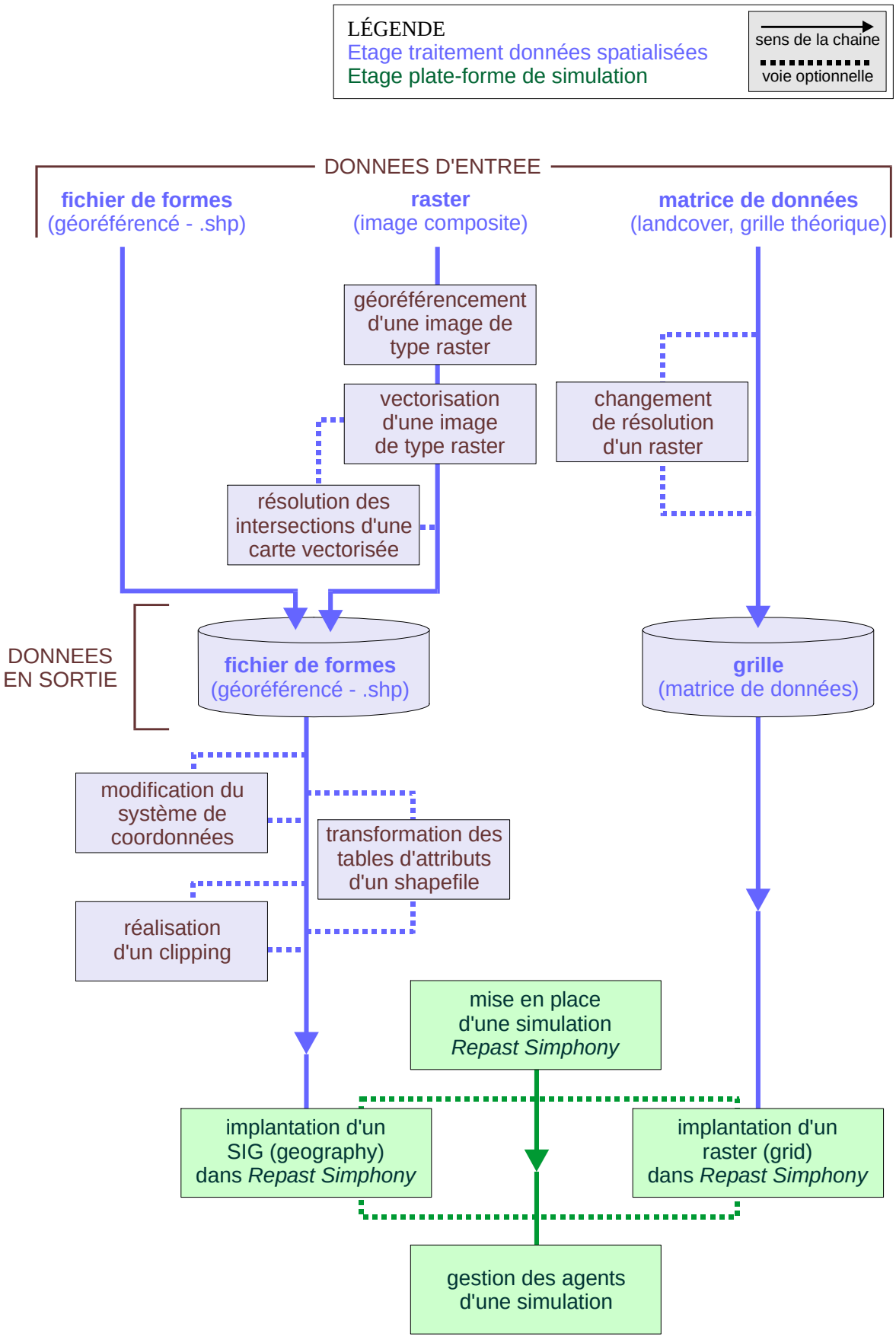
Ce travail s'inscrit dans le cadre d'un projet de recherche sur la dynamique des populations de rongeurs (http://simmasto.org). Il correspond à un module visant au développement d'une plate-forme générique de simulation multi-agents des rongeurs dans leur environnement. Dans le cadre de cette plate-forme, les agents simulés sont amenés à évoluer dans un environnement le plus réaliste possible. Les travaux réalisés dans le centre de recherche se fondent sur l'utilisation de cartes géographiques disponibles numériquement sous des formats et des échelles variées sous forme de systèmes d'information géographique, cartes raster, grilles théoriques, etc.

La chaîne de traitement qui est présentée permet la récupération des cartes et leur implémentation dans la plate-forme de simulation. Elle comprend les éléments permettant le traitement des images d'entrée (géoréférencement, détourage, changement de résolution, clipping, etc.) ainsi que les éléments de programmation liés à l'implantation et la mise en œuvre des supports dans le simulateur multi-agents.

Les procédures ont été élaborées dans le cadre d'un processus d'amélioration continue conduisant à l'élaboration de la chaîne qui est présentée. Cette chaîne prend en compte divers problèmes liés à la mise en forme et au traitement des données d'entrée. Chaque cas a fait l'objet d'un test concret jusqu'à sa résolution effective.

La figure présente les chapitres (modules) du document en situation dans la chaîne de traitement. Le lecteur pourra se reporter à la table des matières pour trouver le texte correspondant au module concerné.

Schéma synoptique de la chaîne de traitement



Module 1 Géoréférencement d'une image de type raster

Objectif :	Fournir un géoréférencement à une image (ex: carte satellitaire) au format raster.
Entrée:	Une image raster (au format .tif, .jpg , etc.).
Sortie:	Une image raster (au format .tif accompagnée d'un fichier de géoréférencement (au format .wld).
Outils utilisés:	*Google Earth, QGIS, (Gimp).*

Avant de pouvoir travailler sur une image type « raster » avec des logiciels SIG, nous devons la géoréférencer pour que les logiciels de traitement de SIG puissent localiser la zone de travail. Il est plus simple de géoréférencer l'image d'origine (dont on peut plus facilement extraire des points particuliers) et de transposer les coordonnées sur l'image modifiée que de géoréférencer l'image modifiée.

Pour obtenir un référencement, il faut connaître les coordonnées d'au moins trois points appartenant à l'image. Il est possible de retrouver la zone correspondante avec Google Earth et de relever les coordonnées de points remarquables. (Google Earth permet de retrouver une zone simplement avec un nom).

Pour faciliter le travail sur la carte, il est plus simple de travailler en mètres **à condition** de ne pas travailler sur des zones immenses (dans ce cas la zone recouvre plusieurs fuseaux UTM et il n'est pas possible de récupérer des coordonnées en mètres du fait de la courbure de la terre qui entraîne une déformation.). Nous allons donc référencer notre image avec des points récupérés sous *Google Earth* et exprimés avec des coordonnées en mètres :

- Pour commencer ouvrez Google Earth et positionnez vous sur votre zone de travail.
  - ☝ Vous pouvez régler le format d'affichage des coordonnées Google Earth dans le menu Outils=> Options.
- Sélectionnez l'onglet 'vue 3d'. dans le menu 'afficher lat/long', sélectionnez « projection transversal de mercator ».
  - ☝ Une fois cette manipulation faite, les coordonnées du curseur (affichées en bas à gauche de la carte affichée) sont exprimées dans une échelle en mètre.
- Ouvrez *QGIS* en parallèle et lancez le plugin « géoréférencer ».
- Dans la fenêtre qui s'ouvre, importez votre image d'origine.
  - ☝ Normalement, votre image doit apparaître dans une nouvelle fenêtre qui vous permet de définir des points de référence.

Vous devez choisir des points remarquables de votre carte (angle de bâtiment, arbre, ville) et prélever les coordonnées correspondantes exprimées en mètres dans *Google Earth*.

- Dans la fenêtre où votre image s'affiche, utilisez le bouton « ajouter des points », puis entrez les coordonnées de différents points en vous reportant à *Google Earth*.
  - ☝ Il est nécessaire de prendre au minimum 3 points, mais plus on en ajoute, plus la précision est importante. Toujours dans un souci de précision, il faut faire en sorte de les prendre les plus éloignés et les moins alignés possible.
- Une fois que vous avez positionné vos points, vous pouvez cliquer sur « créer » ; *QGIS* va alors générer un fichier « world » qui portera le même nom que votre image mais avec l'extension '.wld'

- Ce fichier world ne contient en fait que du texte (une suite de valeur numérique) servant à géoréférencer l'image. Lorsque vous essaierez d'importer cette image dans un logiciel de SIG, le fichier '.wld' sera lu en même temps et renseignera le logiciel sur les coordonnées de l'image.
- *QGIS* peut aussi créer un fichier « .point » qui contient les coordonnées des points que vous avez placés sur l'image. Une fois le géoréférencement effectué, ce fichier n'est plus utile et peut être effacé.

- *QGIS* propose d'importer l'image après son géoréférencement. Vous pouvez vérifier lors de son affichage que l'ensemble des points de la carte possède maintenant des coordonnées.

Maintenant que notre image est géoréférencée, nous pouvons faire correspondre le fichier '.wld' à toutes les images de même dimension (ce qui est intéressant pour géoréférencer les images traitées avec *Gimp* dont on ne reconnaît pas les points remarquables). On peut procéder ainsi car les images ont même taille, elles correspondent au même endroit et à la même échelle, seule la représentation change.

Pour faire correspondre le fichier '.wld ' d'une image à une autre de même dimension, il suffit de faire une copie de celui-ci à coté de l'image non géoréférencée et de le renommer en lui donnant le nom de cette image (sans modifier l'extension).

A partir de maintenant notre image est un 'raster' géoréférencé.

Module 2 Vectorisation d'une image de type raster

Objectif :	Cette partie consiste à créer un fichier au format vecteur depuis une image au format raster (voir le complément sur les rasters et vecteurs en annexe). Le fichier au format vectoriel pourra alors être interprété par des logiciels SIG et il sera possible d'effectuer des traitements ou de l'intégrer dans un simulateur.
Entrée:	une image satellitaire
Sortie:	fichier au format shapefile (.shp) qui soit assimilable par le simulateur.
Outils utilisés:	*Google Earth, QGIS, (Gimp), GRASS*

Présentation de deux méthodes possibles

Il existe différentes manières de procéder pour convertir un fichier du format raster vers le format vecteur. Les deux techniques présentées ont leurs propres avantages et inconvénients.

1. La première et sans doute la plus efficace pour créer un fichier-vecteur léger est de « dessiner » des polygones par dessus une carte satellitaire et d'attribuer à chacun une catégorie (bâtiment, terrain...).
    - Cette solution permet de créer rapidement des fichier de formes au format kml*.
    - Puisque les formes sont définies à la main, il s'agit généralement de polygones réguliers utilisant peu de points. Les fichiers générés nécessiteront donc peu ressources lors de la simulation ou d'un traitement quelconque.
    - La découpe de chaque forme peut prendre du temps et lorsque les cartes à numériser ont une grande surface la découpe à la main peut prendre beaucoup de temps.
    - Si l'on est amené à dessiner des polygones inclus dans d'autres polygones ou se superposant partiellement, on doit effectuer un traitement supplémentaire pour fusionner les éléments (un point sur la carte ne peut appartenir qu'à un seul polygone).
2. Une deuxième solution peut être envisagée, si l'on doit travailler avec un fichier raster (que ce soit une image ou une grille d'attribut) : il est possible d'utiliser des logiciels de vectorisation capable d'isoler les zones en fonction de leurs valeurs (pour une image, on peut découper une image selon les plages de couleurs présentes).
    - Cette méthode peut être relativement rapide grâce aux logiciels de traitement d'image.
    - De plus, en utilisant un système de calques puis en les fusionnant il n'y a pas de problème de gestion de polygones imbriqués (les zones visibles seront des zones indépendantes, il n'y a pas à s'inquiéter d'éventuels chevauchements des polygones).
    - Cette solution donne cependant rarement un résultat satisfaisant immédiat, puisque les couleurs des pixels d'une même zone ne sont presque jamais exact (il y a des dégradés ou des flous qui provoquent de nombreuses erreurs lors de la détection de couleur).
    - Il faut procéder à une étape de traitement supplémentaire en attribuant à chaque zone une couleur unique.
    - Cette méthode pose enfin un problème de taille: les polygones générés lors de la

reconnaissance des zones de même couleur sont découpés selon la même résolution que l'image d'origine, la précision est donc élevée, mais chaque polygone est défini par plusieurs centaines voire milliers de points. Le fichier est donc très lourd à gérer.

Il est possible d'effectuer des méthodes de simplification[1] ; cependant, ces opérations peuvent être très longues puisque le logiciel effectue des traitements récursifs sur tous les polygones touchés par une modification (cela peut prendre plusieurs heures et si le résultat n'est pas satisfaisant il faut recommencer en changeant le seuil). De plus ces opérations de simplification réduisent la précision et ne ramènent pas le fichier à un format de polygone aussi léger que celui d'un fichier créé avec la première solution.

Première méthode : numérisation par tracé de polygone

Remarque: Dans le cas où l'on travaille sur une image définie et pas sur une carte issue de *Google Earth*, on peut procéder aux mêmes étapes à condition d'importer l'image dans un logiciel de traitement SIG. Il est par exemple possible de dessiner des polygones au dessus d'une carte importée dans *QGIS.* L'image doit être géo-référencée (voir Module 1 , p.6).

Utilisation de *Google Earth:*

Google Earth (http://earth.google.fr/) est au départ un logiciel de cartographie, mais les options proposées permettent de faire quelques traitements de SIG (superposition de couches, ajouts d'éléments selon les zones).

Nous allons utiliser ses fonctionnalités pour obtenir une image présentant une vue aérienne puis nous allons faire un repérage des éléments intéressants en créant un fichier KML, c'est à dire en superposant une forme à chaque élément du relief et en établissant un lien entre les deux.

- Une fois que vous vous êtes familiarisé avec l'environnement du logiciel, positionnez vous sur la zone qui vous intéresse.
- Sur le panneau de gauche, sélectionnez l'objet « lieux ».
- Dans l'arborescence, faites un clic droit sur « Mes lieux préférés » et cliquez sur « ajouter un dossier ». C'est dans ce dossier que nous allons stocker les polygones (c'est à dire les parcelles de terrain).
    - Pour faciliter le travail, il vaut mieux commencer par délimiter la zone de travail (que l'on pourra modifier ultérieurement, il ne s'agit que d'un repère).
    - Pour conserver une carte régulière, on va ensuite en extraire un polygone régulier (grâce au clipping). Il faut donc avoir une zone suffisamment large pour pouvoir éliminer les marges sans toucher à la zone qui nous intéresse. Il est donc important de prendre une zone un peu plus grande que celle que vous voulez récupérer, car lors du changement de système de coordonnées de référence (voir Module 4 , p.24) la carte peut subir des déformations.
- Assurez vous que votre dossier est sélectionné puis dans la barre des outils, cliquez sur « ajouter un polygone »
    - Pour dessiner un polygone, il suffit de faire un clic gauche par sommet. Le clic droit supprime un sommet. Conserver le clic gauche enfoncé permet de tracer rapidement plusieurs sommets

[1] Pour la simplification d'un shapefile, dans la console *Grass*, il est possible d'utiliser la commande : `v.clean input= « nom du shapefile à simplifier » output= « nom du shapefile simplifié » tool=rmarea threast = « le seuil de tolérance ».`

de manière consécutive (utile pour dessiner des courbes).

Attention : il faut absolument respecter certaines règles lorsque vous dessinez vos polygones. Il faut éviter que vos polygones soient en « auto-intersection ». C'est à dire qu'une ligne entre deux points ne doit jamais entrer en intersection avec une autre ligne du même polygone.

Figure 1: Exemple de mauvais choix de découpe

Dans cet exemple, les deux bâtiments sont recouverts par un seul polygone, mais à l'endroit où les angles des bâtiments sont en contact, le polygone est en intersection avec lui même.

Pour la numérisation de l'arbre, l'erreur a été amplifiée pour la rendre visible , mais lors de la création du polygone, on risque d'insérer des « boucles ». en particulier lorsque l'on cherche a dessiner des formes courbes.

- Il faut également faire attention à ne pas insérer des éléments autres que des polygones dans le dossier. Par exemple Google Earth propose d'insérer des points de repère, mais ceux ci sont enregistrés comme des géométries de types « point ». Lorsque l'on essaie de le lire dans *QGIS*, le format n'est pas reconnu comme un format valide car un shapefile ne peut contenir que des éléments d'un même type (point ligne ou polygone).

- Lors de la création d'un polygone, une fenêtre s'ouvre, elle permet de définir les propriétés de l'élément que vous êtes en train d'éditer (note: il faut conserver cette fenêtre ouverte pendant que vous tracez le polygone).
  - Pour votre premier polygone délimitant la zone vous pouvez lui donner la description « fond » pour indiquer qu'il sera le polygone "Univers" c'est à dire le polygone qui représentera la zone par défaut. Il est ensuite possible de modifier son style pour afficher ses bordures et rendre le contenu transparent.

- Une fois cette zone délimitée, vous pouvez commencer à différencier les types de terrains.
  - A ce stade, il est conseillé de tracer les polygones catégorie par catégorie pour ne pas avoir à modifier le style à chaque fois.
  - Par défaut, les polygones crées sont blancs et opaques, mais il est possible d'éditer leur description et leur style via leur fenêtre de propriétés.
  - Il vaut mieux conserver un certain degré de transparence pour pouvoir gérer les polygones inclus dans d'autres polygones.
  - Il est **important** de donner une description relativement courte et explicite pour chaque polygone. On va utiliser la description comme repère pour retrouver les catégories. Tous les polygones d'une même catégorie doivent avoir la même description. **Si vous faites des erreurs de frappe vous risquez** de perdre des polygones lors du traitement (le mot écrit dans la fenêtre description correspond à la future catégorie du polygone).

- Pour enregistrer votre travail, faites un clic droit sur le dossier qui contient vos polygones et sélectionnez « enregistrer le lieu sous »
- Avant d'enregistrer votre travail, modifier l'extension en .kml
- Vous pouvez vérifier que le fichier est correctement enregistré en l'ouvrant depuis *QGIS*.

Attention: Selon la version utilisée, il arrive que *Google Earth* génère des fichiers .kml qui ne sont pas interprétés par *QGIS*. Dans ce cas, il suffit de modifier l'en-tête du fichier .kml en récupérant un en-tête valide d'un fichier .kml qui puisse être interprété par *QGIS*. Ci-dessous, exemple d'en-tête KML pour *QGis :*

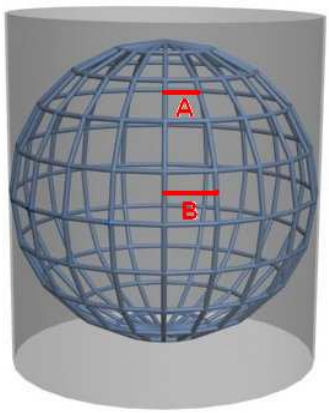
```
<?xml version="1.0" encoding="utf-8" ?>
<kml xmlns="http://earth.google.com/kml/2.0">
```

- Une fois que vous avez ouvert le fichier .kml sous *QGIS* il suffit de faire un clic droit sur la couche importée dans le panneau de gauche et de sélectionner enregistrer comme shapefile pour pouvoir sauvegarder le fichier de forme avec ses attributs, son CRS et ses coordonnées.

Le fichier créé est presque exploitable, mais deux problèmes doivent être résolus avant de continuer.

1. La première source d'erreur provient du format des coordonnées. En effet *Google Earth* enregistre les fichiers avec un géoréférencement en latitude-longitude ce qui n'est pas pratique pour le simulateur. Dans une simulation on travaille en mètre car une distance de 1 degré ne correspond pas à la même distance selon la position géographique.

   Sur l'exemple ci dessous, on peut voir qu'une distance entre deux coordonnées exprimées en latitude-longitude diffère selon la distance à l'équateur (une distance de un degrés en longitude à l'équateur est plus grande qu'une distance de 1 degré près des pôles.

   On remarque que les distances représentées par les segments A et B représentent le même écart en terme d'angle, mais ne correspondent pas à la même distance réelle du fait de la distorsion du repère. Plus la zone étudiée est grande, plus la distorsion des distances exprimées en angle devient importante.

   Même si le format latitude/longitude reste cohérent en terme de distance sur de petites surfaces, il est difficile d'y faire correspondre d'autres données comme l'aire d'un polygone ou des vitesses en kilomètres par heure. Il est donc préférable de convertir le repère pour travailler en mètre.

   Il faut donc convertir le CRS au format UTM. Il est possible de faire ce changement grâce à une fonction de *PostGIS*, mais il faut connaître le SRID (identifiant du système de référence) du fichier de départ et celui d'arrivée.

2. L'autre source d'erreur est invisible, mais il faut tenir compte de la superposition des polygones. En effet, dans le cas d'une simulation, il ne faut pas qu'un élément puisse se situer sur deux types de terrains en même temps.

Pour résoudre ces deux problèmes, il est possible d'utiliser *PostGIS* (voir Module 3 , p.19)

Deuxième méthode : vectorisation automatique par logiciel

Cette solution permet de créer un fichier shapefile en convertissant un image raster d'une vue satellitaire (il est possible d'en récupérer depuis *Google Earth*) ou d'une carte standard.

Pour mieux comprendre les étapes du traitement, nous allons expliciter les besoins puis tacher d'y répondre point par point :

a) Il faut pouvoir reconnaître la zone cible, c'est à dire qu'il faut géoréférencer l'image satellitaire récupérée pour pouvoir travailler (Il faut pouvoir effectuer une reconnaissance des différentes zones).

b) Après la reconnaissance « visuelle », il faut pouvoir isoler les éléments dans différentes catégories.

c) Les catégories étant toujours sous un format d'image, il faut les modifier pour que chaque élément puisse être reconnu en tant que zone (polygone) et puisse accepter des attributs.

d) Une fois les catégories créées et mises dans un format compatible avec les traitements des SIG, il faut donner des attributs propres à chaque catégorie

A.- Reconnaissance et choix des catégories

Pour numériser une carte, il faut pouvoir découper tous les éléments qui nous intéressent sous forme géométrique, c'est à dire transformer la carte du format raster au format vecteur, (voir en annexe p.69 le complément sur les rasters et vecteurs). Chaque polygone pouvant contenir plusieurs centaines de points et une carte contenant souvent plusieurs milliers de polygones, il n'est pas possible de les redéfinir point par point à la main. Nous allons donc **utiliser un logiciel qui va créer ces polygones en fonction des différentes zones de couleur** présentes sur la carte.

- Cette aide logicielle est pratique, mais le logiciel ne permet pas de remplacer l'œil humain et l'interprétation. En effet, le logiciel ne fait pas la distinction entre deux éléments qui ont la même couleur, ou qui ont une couleur proche. Ainsi la pierre et les bâtiments peuvent être confondus, de même que l'herbe et les arbres, il est parfois même nécessaire de faire des distinction entre élément visuellement semblable (type de bâtiment, de culture...).

Pour permettre au logiciel de fonctionner correctement, nous allons commencer par une étape de dessin, où nous allons faire une reconnaissance manuelle des zones qui nous intéressent. Nous allons donner une couleur unique pour chaque catégorie et peindre l'image de façon à ce que chaque élément soit recouvert par une zone colorée propre à sa catégorie. Pour différencier les zones d'une image, nous allons utiliser un logiciel de traitement d'image. Le plus célèbre est *PhotoShop*, mais nous allons nous servir de *GIMP* (http://www.gimp.org/) qui présente des fonctionnalités similaires et qui à l'avantage d'être libre. Nous allons créer un calque transparent et indépendant pour chaque catégorie et dessiner des polygones pour chaque élément. On va faire en sorte que les polygones se superposent aux éléments de la carte en dessinant sur le calque au dessus de l'image (voire la figure suivante).

Avant de commencer à découper l'image en différentes parties, il faut choisir quelles sont les catégories et les classer par ordre de pertinence. En effet il faut décider quels sont les éléments qui doivent recouvrir les autres (par exemple les bâtiments recouvrent les zones de terrain). Cette distinction est utile puisque lors de la découpe on va pouvoir étendre les zones qui ont une basse priorité d'affichage.

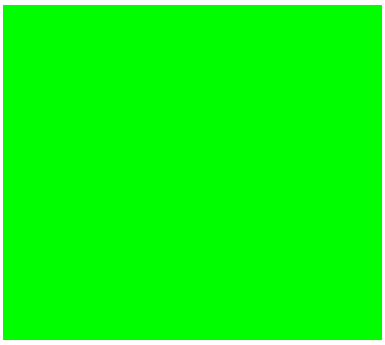
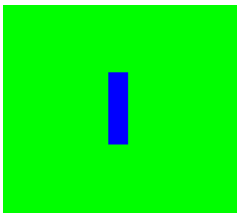




Image de départ | calque terrain | calque bâtiment

Il faut que chaque élément soit défini. Dans ce cas, si on ne décompose l'image qu'en deux catégories, on va définir un calque par défaut qui représente la plus grande partie de l'image, c'est à dire le terrain. On va donc entièrement recouvrir cette zone par un polygone de même dimension que l'image mais d'une couleur unique représentant le terrain. On va ensuite créer les éléments particuliers en utilisant d'autres calques. Dans cet exemple, on va considérer que le bâtiment central est une zone remarquable dont les propriétés écrasent celle de la couche inférieure (le bâtiment est sur le terrain).

Une fois les différents calques créés, il suffit de les fusionner pour obtenir une image avec des zones parfaitement délimitées et sans dégradés :



- On découpe les zones selon les calques
  - En procédant ainsi, on est sur de ne pas créer d'espace vide entre le terrain et les bâtiments, et on n'a pas à découper la forme du bâtiment dans le calque terrain

Note concernant la découpe : La numérisation d'une carte est un élément clé de l'optimisation pour la simulation. Il faut être suffisamment rigoureux pour que le terrain modélisé soit comparable au terrain réel. Plus on donne de détail plus la simulation sera précise. Mais il y a néanmoins un inconvénient à être précis, la taille du fichier numérique créé va devenir rapidement encombrante.

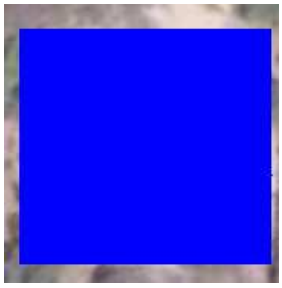
En effet lorsqu'on effectue la numérisation d'une carte raster vers une carte vecteur, on individualise chaque élément en tant que forme géométrique (polygone). Mais chaque élément polygone contient une définition de son apparence géométrique sous la forme d'une suite de points délimitant ses frontières. Par exemple un polygone carré pourra être défini par le code suivant :

Il faut 5 points, pour fermer le carré :

```
<Coordinates>
0,0
0,1
1,1
1,0
0,0
</coordinates>
```

Maintenant si nous voulons numériser un bâtiment ayant une forme particulière:

Nous avons plusieurs façons pour le numériser:

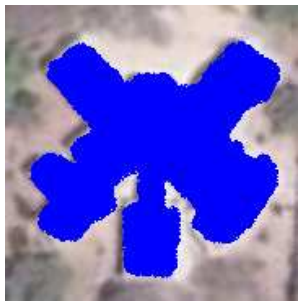


1

Représentation par un carré : le plus simple, mais peut suffire si on travaille à une échelle suffisamment grande et que la précision est réduite => utilise 4 points

2

Découpage « à la main » en utilisant des contours réguliers (découpage par polygones), celle utilisée dans la numérisation de la première méthode. Cette technique est la plus précise mais aussi la plus longue, surtout si on doit numériser un grand nombre d'éléments => utilise environ 35 points



3

Sélection par plage de couleur (baguette magique) cette technique est très rapide et très puissante, mais elles inclu parfois des zones non désirées et le polygone créé est très irrégulier => utilise plusieurs centaines voire millier de points

☝ Avant de commencer à découper une carte, il faut prendre en compte la surface a numériser, et l'utilisation prévue. Pour l'intégration du shapefile dans un simulateur, le fichier doit être suffisamment petit, la deuxième solution est plus « efficace » mais peut prendre du temps.

B.- Découpe

Pour découper une carte, plusieurs méthodes permettent de gagner du temps. Selon l'image sur laquelle vous travaillez, vous pouvez essayer d'**augmenter le contraste** avec les outils *GIMP* ou *PhotoShop* pour améliorer l'efficacité des outils de sélection utilisant la détection de zones ayant des couleurs proches.

Ensuite vous avez plusieurs moyens pour séparer les zones, la plus longue consiste à dessiner chaque polygone à l'aide de l'outil de **sélection au lasso** (raccourcis *GIMP* 'f'). Vous découpez une zone depuis le calque correspondant à la couche cible, puis vous appliquez le pot de peinture (raccourcis *GIMP* =' shift + b '(une couleur par calque)

Remarque importante sur le choix des couleurs: le logiciel que nous allons utiliser pour la reconnaissance des couleurs récupère des images et les traite en tant que fichier constitué de trois couches (rouge, vert, bleu). La reconnaissance des polygones ne s'effectue que sur une de ces trois couches au choix de l'utilisateur. La création des polygones est indépendante de la couleur, mais le logiciel attribue à chaque polygone la valeur de la composante du filtre choisi. On reconnaît ensuite la catégorie des polygones d'après cette valeur attribuée à la couleur de la composante. Il est donc très important que deux éléments disjoints n'aient pas la même valeur de la composante choisie pour la découpe.

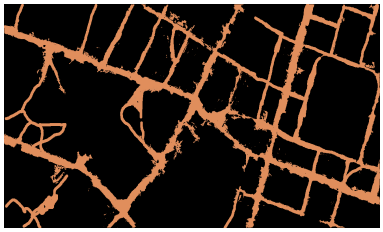
Pour mieux comprendre, si nous créons une carte très simple contenant deux catégories, (bâtiment et terrain), pour les distinguer nous choisissons deux couleurs (jaune et violet). Les codes couleurs en hexadécimal sont FFFF00 et FF00FF (voir la note sur le codage RGB dans le glossaire, p.73). Lorsque l'on va importer le fichier raster (dans *GRASS* par exemple), le fichier va être décomposé en trois couches. En prenant la couche « rouge » pour effectuer la découpe, les polygones seront créés, mais posséderont tous la valeur 255 (la composante maximale de rouge). Il ne sera alors pas possible de faire la distinction entre les deux couleurs de départ. Les polygones terrains et les polygones bâtiments seront confondus. Pour éviter ce problème, il est préférable de définir dès le début un tableau de couleurs en faisant en sorte que chaque couleur possède des composantes uniques.

- Une fois que vous avez défini votre palette de couleur, vous pouvez commencer à découper votre image :
  - Dans le cas de zone homogène , comme les champs ou les bâtiments, vous pouvez utiliser l'outil de sélection par plage de couleur, ou baguette magique (raccourci u), Cet outil se révèle très utile lorsque vous devez découper des régions de couleurs homogènes mais avec des formes complexes (comme une route). En réglant le seuil de sélection et en utilisant les touches ctrl et maj (pour fusionner des sélections par zones de couleurs ou les soustraire) pendant la sélection, le travail est grandement facilité. Enfin pour les formes régulières les outils de sélection par polygone permettent d'isoler rapidement les champs et les bâtiments rectangulaires.
  - Il est vivement conseillé de garder un enregistrement de l'image à un format conservant les calques, pour permettre de faire des modifications ultérieures.
- Une fois que toutes les zones sont découpées, si toutes les couches ne recoupent pas l'ensemble du terrain, on peut ajouter un fond uniforme qui représentera la zone « par défaut ».

Exemple: en partant d'une image satellitaire, on isole les éléments par couche (les zones noires sont transparentes). Puis on fusionne les calques.

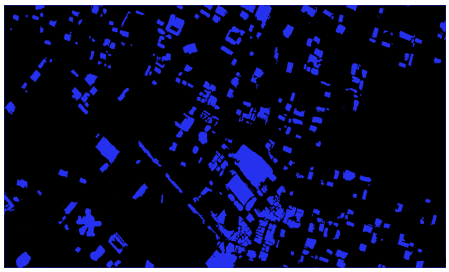
Image d'origine

Calque de fond



calque de route

calque de terrain



Calque de bâtiment

Résultat de la fusion des calques

- Quand la superposition de toutes les couches donne un résultat satisfaisant, vous pouvez exporter l'image obtenue au format « tiff».

Avant d'utiliser le logiciel permettant de créer les polygones en fonction des zones de couleurs, il faut géoréférencer l'image:

- Pour vérifier que votre image « tiff » est correctement géoréférencée, ouvrez la dans *QGIS* en utilisant « ajouter une couche raster ».
- Si rien ne s'affiche, vérifier que le CRS de votre projet est bien au bon format.
- Si vous ne savez pas quel CRS utiliser, reportez vous au Module 4 Modification du système de coordonnées, p.24.
- En déplaçant le curseur sur la carte, vous devriez pouvoir vérifier que les coordonnées correspondent à celles récupérées sur google earth.

C.- Utilisation de *GRASS*

- Pour effectuer des traitements, vous devez créer un jeu de donnée *GRASS* qui nous servira de répertoire de travail.
- Dans le menu Plugin sélectionnez GRASS puis « Nouveau jeu de donnée »
- Vous devez alors créer un répertoire.
- Vous devez ensuite indiquer le nom du secteur dans lequel vous travaillez (ville région ou zone selon l'échelle).
- Vous devez spécifier les propriétés de la projection dans laquelle vous allez travailler. Il faut indiquer le CRS (Coordinate Reference System) qui servira de référence.
- L'étape suivante consiste à délimiter par des cordonnées la région de travail, il suffit de cliquer sur « fixer l'empreinte courante de *QGIS* » puisque la carte qui est affichée dans *QGIS* est maintenant géoréférencée.
- Finalement, il ne reste plus qu'à nommer votre jeu de données et à terminer sa création.

Le jeu de données créé est encore vide, nous allons importer la carte

Remarque : il faut faire la distinction entre les cartes importées dans *QGIS* et celles importées dans le répertoire de travail de *GRASS*.

Attention : Pour certaines fonctions de Grass, il est nécessaire que les couches concernées soient affichées (ou sélectionnées) dans QGIS.

- Cliquez sur « ouvrir les outils grass »
  - La fenêtre qui s'ouvre va vous permettre d'utiliser les fonctionnalités de *GRASS*. L'onglet « Parcourir » vous permet de visualiser les éléments présents dans le répertoire de travail (pour l'instant il est vide)
- Cliquer sur l'onglet « arborescence des modules ». Une liste assez longue de module apparaît (*GRASS* permet de faire vraiment beaucoup de chose)
- Dans l'arborescence sélectionnez : file=>Import=> Import Raster=> R.in.gdal–import Gdal supported Raster.
  - La fenêtre ouvre alors un nouvel onglet qui vous permet d'utiliser le module.
- Dans le champ « raster file to be imported », chargez le fichier tiff que vous avez coloré et géoréférencé. Dans le champ suivant, indiquez le nom du fichier qui sera importé dans GRASS.
- Cliquez sur « lancer ».
- Dans l'onglet « parcourir », cliquez sur le bouton « rafraîchir », vous devriez voire apparaître un dossier raster dans votre espace de travail. Celui ci contient les différentes couches définissant l'image importée.

Avant de la convertir en vecteur, il est préférable de changer la résolution de travail pour ne pas perdre de précision durant la conversion.

- Sélectionnez une couche couleur « nom_fichier_raster.red » et ajoutez la dans *QGIS* (soit en double cliquant dessus, soit en la sélectionnant et en cliquant sur le bouton « ajoute la carte sélectionnée à la carte »
- Cliquez sur « région courante réglée sur la carte choisie »

- Dans l'onglet « arborescence des modules sélectionnez « shell – GRASS shell »

Grass ouvre alors un onglet qui vous permet d'accéder à la console de grass

- Dans l'invite de commande tapez : g.region rast= tuto.red en remplaçant tuto.red par le nom de la couche que vous avez sélectionné.
  - Cette commande va recopier les propriétés du raster (dont la résolution) dans les propriétés de la région courante. On va se servir de ces propriétés pour créer le fichier vecteur.
  - Si la commande marche correctement, l'invite de commande se réaffiche aussitôt sans erreur

(il n'y a pas de confirmation de la commande).

- Retournez dans l'onglet arborescence des modules
- Sélectionnez File=>Map Type Conversion=> r.to.vect.area-convert raster to vecteur areas
  - Dans le champs Raster Input file, seules les cartes que vous avez importé dans le plan de travail *GRASS* et sélectionné comme étant la région courante apparaissent (vous ne devriez en avoir qu'une seule, qui correspond à la couche que vous avez sélectionné).
- Lancez la conversion.
- Une fois terminé, vous pouvez fermer la fenêtre d'outils *GRASS*.
- Dans les options du plugin *GRASS* sélectionnez « ajouter une couche vectorielle grass ».
  - Normalement les paramètres sont par défaut les paramètres courants de *GRASS* sinon sélectionnez le nom du projet, et du secteur. Le nom de la couche que vous venez de créer doit être accessible, ajoutez-la dans *QGIS*.
  - A ce stade, vous devez pouvoir voir votre couche vectorisée, les polygones ont été créés.
- Tous les éléments sont de la même couleur, mais ceci est normal, vous pouvez vérifier que la distinction existe en allant dans les propriétés de la couche. Rendez-vous dans l'onglet « convention des signes » puis dans le champs « type de légende » cliquez sur « valeur unique ». Pour le clan de classification, choisissez « value » (cette valeur correspond à la quantité de la composante de la couche qui a servi pour la vectorisation (r, g ou b))
- Cliquer sur « classer » puis appliquez le résultat.

Si tout c'est bien passé, vous avez à présent une carte vectorisée où chaque élément est associé à une catégorie. Vous pouvez définir le nom et les attributs des polygones d'une des catégories en utilisant une jointure sur l'attribut value (la procédure pour réaliser une jointure est décrite dans le Module 1 , p.6).

- Pour finir, vous pouvez enregistrer cette couche sous la forme d'un shapefile : clic droit sur le nom de la couche dans le panneau de gauche => sauvegarder comme shapefile.

Module 3 Résolution des intersections d'une carte vectorisée

Objectif :	Eliminer les superpositions des polygones dans un fichier shapefile.
Entrée :	Un fichier shapefile (ensemble de polygones)
Sortie :	Un fichier shapefile dont les polygones ont été modifiés pour éliminer les superpositions
Outils:	*PostgreSQL / PostGIS* (descriptions p.66), *QGIS*
Prérequis:	Avant de commencer cette étape, vous devez disposer d'un serveur de données PostgreSQL (http://www.postgresql.org/) et vous devez avoir installé PostGIS (http://postgis.refractions.net/). Il faut également connaître les commandes de base permettant de créer et d'accéder à une base de données.

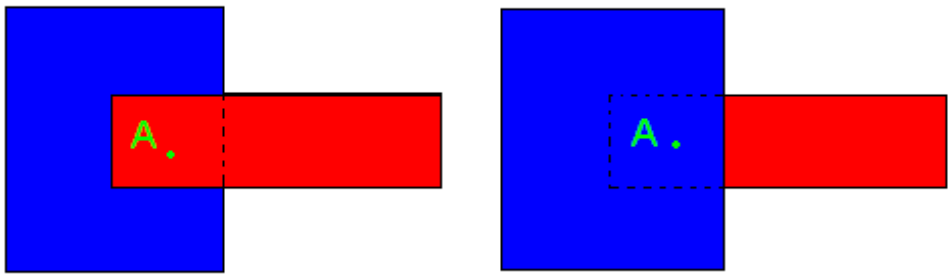
☝ La connaissance du langage SQL est utile pour comprendre la logique de fonctionnement de postgres et vous permettre de concevoir vos propres requêtes spatiales.

Présentation du problème

Dans un simulateur, un agent pourvu de coordonnées uniques ne peut pas se trouver sur deux types de terrain en même temps (bâtiment et champs) ; à la question « quel élément se trouve à cette coordonnée » on ne veut avoir qu'une seule réponse possible.

Lors de la création du shapefile, même si les polygones existent et sont bien placés et définis, leur superposition pose donc problème.

Dans l'exemple ci dessous, le logiciel ne fait pas de distinction entre les deux figures. Le point A appartient donc aussi bien au rectangle rouge qu'au rectangle bleu.



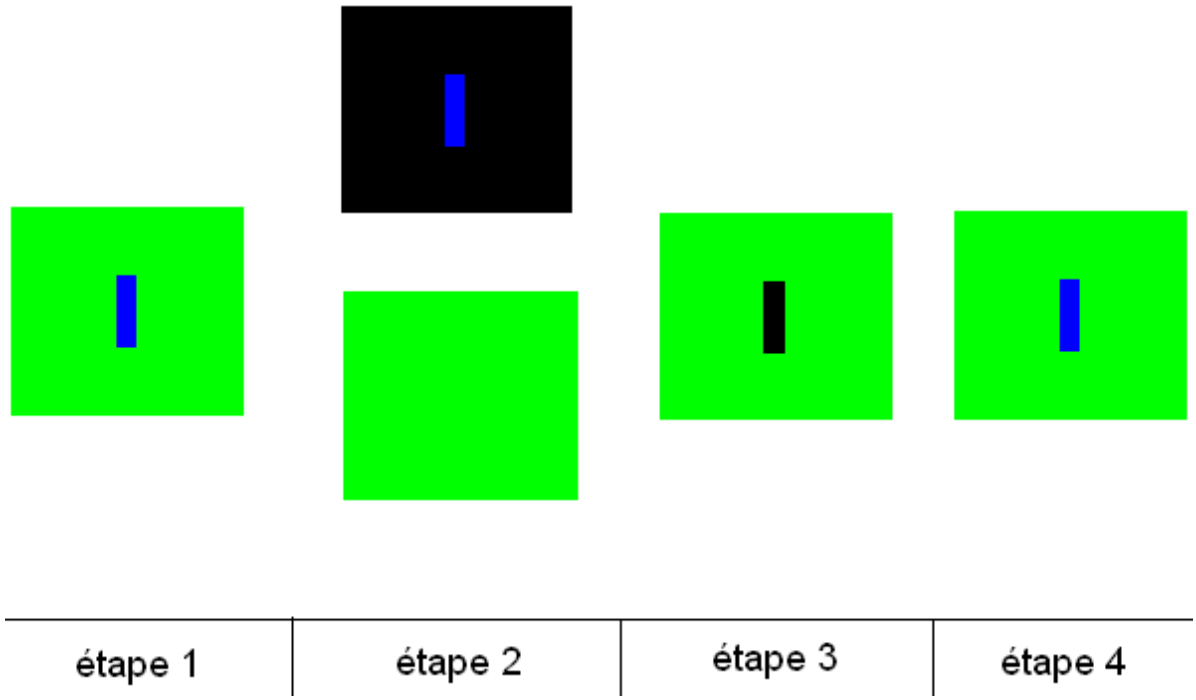
☝ Si l'on prend l'exemple d'un bâtiment dans un terrain, lors de la numérisation on se contente de créer deux rectangles superposés (le tracé de *Google Earth* ne permet pas de créer des polygones avec des « trous »). Le polygone terrain existe donc « sous » le polygone bâtiment (dans le cas où il ne masque pas celui ci).

Ce problème peut être résolu visuellement en utilisant des priorités de couches et de degré de transparence, mais elle n'est pas très satisfaisante pour effectuer des traitements et des requêtes spatiales.

Nous allons donc effectuer un traitement sur le shapefile pour faire en sorte qu'un point donné n'appartienne qu'à un seul polygone.

Découpage d'une couche par une autre

Pour effectuer ce traitement sur un cas précis, il est possible de découper chaque zone en fonction des couches qui lui sont supérieures.



Nb : noir = vide

On découpe à l'aide d'une fonction d'intersection une couche « champs » pour créer des trous dans les polygones là où se trouvent des bâtiments

- étape 1 : on différencie les couches en faisant des extractions en fonction des attributs (on peut le faire en utilisant une jointure pour ne conserver que les éléments d'un certain type)
- étape 2 : On utilise une fonction « geoprocessing » de différence pour soustraire la couche bâtiment à la couche terrain. On obtient ainsi un terrain qui ne sera pas défini là où se trouvent les bâtiments
- étape 3 : Il faut ensuite réunir les couches après modifications en utilisant une fonction d'union.
- étape 4 : Le shapefile résultant ressemble exactement à celui de départ, mais il ne contient plus qu'une seule épaisseur.

Cependant cette solution n'est pas adaptée lorsque l'on travaille sur un fichier shapefile contenant plus de trois ou quatre couches. Même si les traitements à effectuer sont relativement simples, il faut faire un nombre important d'opérations pour soustraire toutes les couches puis les réunir.

Pour simplifier ce traitement, il est possible d'utiliser des fonctions de *PostGis* qui permettent de réaliser rapidement des opérations complexes.

Utilisation de PostGIS

Pour commencer, il faut importer votre fichier shapefile en tant que table de données dans *PostGIS*. Pour créer cette table nous allons utiliser une requête SQL.

1. Dans la console, entrez :

```
shp2pgsql carte.shp carte_init > creer_table_carte.sql
```

- cette commande va créer un fichier nommé *creer_table_carte.sql* qui va contenir la requête permettant d'intégrer les données contenus dans le fichier *carte.shp* dans une table qui s'appellera *carte_init*.

2. Il faut ensuite lancer l'interpréteur de requête SQL et vous connecter à la base de donnée. Pour lancer l'interpréteur, il suffit de taper :

```
psql « nom_de_la_base »
```

   - Il peut être nécessaire de rajouter des paramètres si vous travaillez sur une base distante, à savoir votre nom d'utilisateur, éventuellement votre mot de passe, l'hôte sur lequel vous voulez vous connecter et le port correspondant.

Vous pouvez ensuite accéder aux commandes disponibles en tapant « \? » (NB: \d liste des tables dans le serveurs, \d nom_table, description de la table nom_table)

Pour entrez des commandes concernant le shell normal, il faut précéder la commande par « \! »

3. Pour lancer la requête SQL que vous avez créé précédemment, il faut entrez la commande

```
« \i » «  le chemin du fichier contenant la requête sql »
```

   - si vous êtes toujours dans le fichier ou vous avez enregistré le fichier « *creer_table_carte.sql* » il suffit donc d'entrer :

```
\i creer_table_carte.sql
```

Règle de découpage

Pour pouvoir produire un code facilement adaptable quel que soit le nombre de couches et leur ordre, nous allons nous imposer une règle:

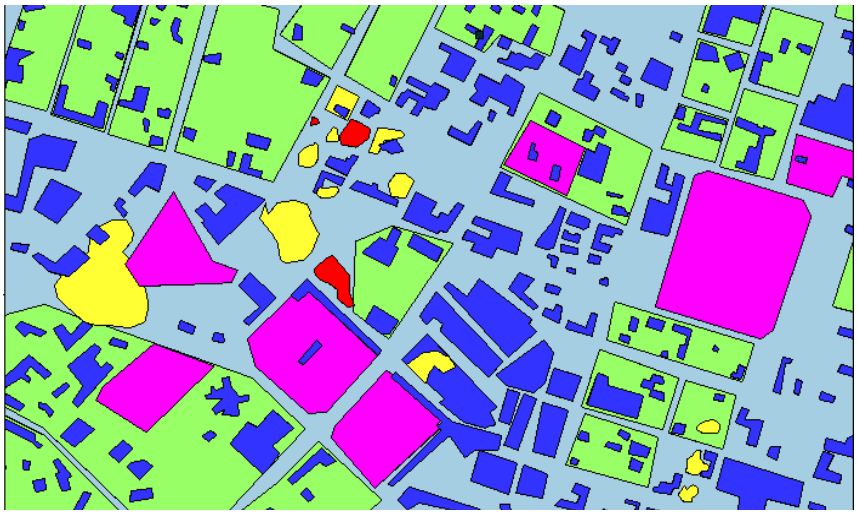
Dans le cas d'une superposition de polygones, on découpe les éléments de l'intersection des deux polygones concernés, et on rajoute la partie « découpée » du polygone qui contient la plus petite aire.

Ces conventions permettent de conserver la logique de répartition et de valoriser les plus petites zones. **De manière générale les petits éléments recouvrent les plus grands**. On peut alors afficher les couches par ordre logique.

→ Le polygone de fond qui est le plus grand sera écrasé par les polygones terrains, à leur tour écrasés par les polygones arbres.

Ainsi, il n'est plus nécessaire de créer des dizaines de couches intermédiaires pour faire attention aux priorités d'affichage. On peut donc traiter rapidement des fichiers relativement complexes:

Dans l'exemple suivant, on retrouve de nombreux cas de superposition, mais les zones en conflit sont attribuées aux polygones ayant la plus petite surface. On conserve ainsi une plus grande précision, on ignore juste les parties recouvertes des polygones « conteneurs ».



Cette façon de procéder permet de donner une priorité plus haute aux éléments contenus en découpant les éléments qui les englobent.

4. Une fois que vous avez intégré votre fichier shapefile dans une base de données *PostgreSQL*, vous pouvez utiliser la requête SQL suivante (soit en l'enregistrant dans un fichier avec comme extension .sql et en le lançant dans l'invite de commande en console de Psql, soit en passant par l'interface graphique de *PGAdmin* (un gestionnaire de base de données *PostgreSQL*). :

Les codes suivants sont commentés pour vous aider à comprendre la technique mise en place, il suffit de l'adapter selon votre fichier. Il faudra modifier le nom de « kedougou » par la table qui contient votre shapefile et éventuellement modifier les attributs concernés.

```
DROP TABLE IF EXISTS table_resultante ; --on supprime la table dans le cas
                                ou elle existe déjà depuis un test
                                précédent

CREATE OR REPLACE VIEW kedougou_auto_join AS SELECT
    kedougou.gid as gid,
    kedougou."name" as "name",
    kedougou.descriptio as "descriptio",
    GeomUnion(kedougou.the_geom) AS full_geom,
    GeomUnion(kedougou_bis.the_geom) AS shared_geom
    FROM  kedougou ,kedougou AS kedougou_bis
     -- on récupère tous les éléments de la table kedougou dans deux --
        collections pour pouvoir les comparer.
    WHERE
      --on vérifie la validité des polygones  testés
      ST_IsValid(kedougou.the_geom) AND ST_IsValid(kedougou_bis.the_geom)
      --on filtre pour conserver les polygones en intersection
      AND intersects(kedougou.the_geom,kedougou_bis.the_geom)
      --on élimine les polygones en intersection avec eux même
      AND kedougou.gid <> kedougou_bis.gid
          --pour une intersection de 2 polygones, on n'en garde qu'un ( le
          plus petit)
      AND Area2d(kedougou_bis.the_geom) < Area2d(kedougou.the_geom)
          --puisque l'on fait des "unions", il faut effectuer un
          regroupement sur les autres attributs
    GROUP BY kedougou.gid,kedougou."name" , kedougou.descriptio ;
```

/*

On va créer une table qui va contenir le résultat final en soustrayant à la carte complète les polygones de la table précédentes (on supprime ainsi les zones en conflits).

Une fois la soustraction faite, il suffit de rajouter les polygones manquants à savoir ceux qui étaient en conflit mais avec une taille inférieur aux polygones soustraits. En procédant ainsi, on est donc sûr de conserver la surface initiale mais en supprimant zones en plusieurs épaisseurs
*/

```
    CREATE TABLE table_resultante AS SELECT
    gid,
    "name",
    descriptio,
```

/*
On soustrait les "petits polygones" en intersections à la couche contenant tous les polygones; on supprime ainsi les parties en conflits
*/

```
    multi(difference(full_geom,shared_geom)) as the_geom,
    area2d(full_geom) as area
    FROM kedougou_auto_join
    WHERE ST_IsValid(full_geom) AND ST_IsValid(shared_geom)
```

/*
On doit ensuite rajouter les "petits polygones" que l'on vient de soustraire pour « boucher » les trous crées.
*/

```
    UNION
      SELECT
      gid,
      "name",
      descriptio,
      the_geom,
      Area2d(the_geom)
      FROM kedougou
      – c'est à dire ceux dont le gid  n'est  plus présent
      WHERE gid NOT IN (SELECT gid FROM kedougou_auto_join);
```

/*
on redéfinit ensuite une clé primaire. Même si cette étape n'est pas nécessaire pour créer la table, elle est nécessaire pour que *QGIS* puisse l'interpréter (il lui faut un index).
*/

```
ALTER TABLE table_resultante ADD CONSTRAINT pk_table_resultante PRIMARY KEY
(gid);
```

5. Aller ensuite dans *QGIS*,

   → choisir l'icone 'ajouter une couche PostGis',

   → choisir la table table_resultante (dans ce cas),

   → vérifier le bon détourage des polygones,

   → sauvegarder la nouvelle couche au format .shp.

Module 4 Modification du système de coordonnées

Objectif :	Modifier le CRS (Coordinate Reference System) d'un shapefile pour que ses coordonnées soient exprimées en mètres.
Entrée:	Un fichier Shapefile
Sortie :	Un fichier Shapefile
Outils :	*QGIS*, *Google Earth* ou *Ramses Conversion*.

Importance du choix du système de coordonnées de référence

Pour pouvoir travailler avec une carte dans un simulateur,il est préférable voire obligatoire de travailler en mètre. Hors une grande partie des cartes récupérées au format shapefile ont des coordonnées exprimées en angle selon leur latitude et longitude (en degrés minutes secondes ou degrés décimaux).

Il existe une quantité impressionnante d'autres formats possibles pour exprimer des coordonnées, et il est souvent nécessaire d'effectuer des conversions.

Il est très important de choisir et de sélectionner le système de référence de coordonnées le plus tôt possible; sinon, dès que l'on cherche à effectuer des conversions, on risque de provoquer des déformations et d'engendrer des erreurs et des imprécisions.

Pour mieux comprendre le problème posé par le passage de coordonnées géographiques à des coordonnées de projection cartographique, vous pouvez consulter les liens suivants:

http://fr.wikipedia.org/wiki/Système_de_coordonnées_géoréférencées

et

http://fr.wikipedia.org/wiki/Projection_cartographique

Plus spécifiquement pour le format UTM que nous allons utiliser:

http://fr.wikipedia.org/wiki/Transverse_Universelle_de_Mercator

Modifier le CRS

Pour pouvoir définir le bon format, il faut connaître les fuseaux dans lesquels on travaille

- On peut retrouver le fuseau en utilisant des utilitaires de conversion comme le logiciel « *Ramses conversions*» ou en utilisant une nouvelle fois *Google Earth*.
  - Dans les options de *Google Earth* (outils -> options -> vue3D ->afficher Lat/lon -> projection transverse de Mercator), il faut utiliser le mode d'affichage de projection transverse de Mercator les deux premiers chiffres permettent d'indiquer la zone UTM sur laquelle on se situe. Il faut ensuite relever si on se situe au sud ou au nord de l'équateur (la lettre N ou S apparaît à la fin des coordonnées)

- Une fois ses coordonnées prélevées, vous pouvez ouvrir *QGIS* et importer la couche dont vous voulez modifier le CRS.
- Pour modifier le système de coordonnée d'une carte déjà géoréférencée, on peut utiliser *FTOOLS* en choisissant dans le menu Data management tools le module « export to a new projection ». Choisissez "système de coordonnées projetées" puis dans la liste sélectionnez « Universal Transverse Mercator (UTM)".

  ☝ Le système UTM permet de travailler en mètre sur n'importe quelle zone à condition que celle ci ne recouvre pas plusieurs fuseaux. Dans le cas où la zone de travail est très grande, il faut utiliser une autre projection.
- Dans la liste qui apparaît, les éléments qui nous intéressent sont vers la fin. Sélectionnez l'élément « WGS 84/ UTM zone XXX » (les deux premiers x correspondent à la zone prélevée dans Google Earth et le dernier est la lettre N ou S selon votre position à l'équateur).
- Il suffit ensuite de confirmer le format et de lancer la conversion en spécifiant un shapefile de sortie (il est préférable de conserver le premier au cas ou la conversion ne donne pas un résultat satisfaisant).

Module 5 Transformation des tables d'attributs d'un shapefile

Objectif :	Les données d'un fichier .shp ne sont pas toujours en adéquation avec les besoins de la simulation. On peut avoir besoin d'ajouter des attributs à une table, de changer le champ de référence, de supprimer des éléments qui ne nous intéressent pas ou de reclasser les catégories dans une autre classification, etc. Tous ces besoins peuvent êtres résolus par l'utilisation d'une jointure.
Entrée :	Un fichier shapefile
Sortie :	Un fichier shapefile modifié à partir de changement effectué sur le fichier DBF associé.
Outils:	*QGIS*, Un tableur prenant en charge les fichiers .dbf

Exemples d'applications

A.- 1) Modifier des attributs et éliminer certains éléments

Lorsque l'on dispose d'un fichier shapefile, il est parfois nécessaire d'effectuer des modifications pour le rendre plus explicite et pour éliminer des éléments superflus.

Par exemple après la numérisation d'une image satellitaire avec GRASS (chap.Module 2 C.- p.16), le fichier shapefile créé contient parfois des polygones parasites de un ou deux pixels et la table des attributs du fichier shapefile possède une structure (fichier .dbf) ressemblant à celle ci :

id	cat	value
1	1	255
2	2	153
3	3	215
4	4	78
5	5	32
6	6	12
7	7	0
,	,	,

Chaque ligne représente un polygone, les types de terrain sont alors référencés par la valeur d'une composante de couleur, enregistrée dans le champ « value » avec une valeur entre 0 et 255.

Cette distinction, bien que unique pour chaque type de terrain n'est pas explicite, il faut utiliser une table de correspondance pour retrouver la nature du terrain.

De plus si des erreurs se sont introduites lors de l'enregistrement de l'image, lors d'un changement de repère ou lors de la conversion en vecteur par exemple, on observe des catégories supplémentaires (qui ont une couleur particulière mais qui ne sont associées qu'à un seul pixel).

L'utilisation de la jointure permet de créer une nouvelle colonne d'attributs en fonction d'une autre existante et à modifier.

On peut ainsi faire correspondre la valeur contenue dans la colonne « value » à un type de terrain défini explicitement et supprimer les éléments ayant une valeur ne correspondant à aucun terrain.

B.- 2) Regroupement par catégories

Si dans un shapefile, les polygones ont un attribut « zone » pouvant prendre de nombreuses valeurs « prairie, friche, forêt, blé, orge, colza, jachère... », on peut vouloir les trier et les regrouper en un nombre plus restreint de catégories (reclassement) pour simplifier la mise en place de la simulation (libre, culture...).

Il est bien sur possible dans le simulateur de récupérer les attributs zones et de faire correspondre chaque zone à une catégorie au moment de leur création, mais à chaque démarrage du simulateur, il va falloir effectuer plusieurs tests sur chaque zone, ce qui peut se révéler une solution lourde et lente. Pour éviter d'avoir à effectuer ces tests, nous pouvons modifier le .dbf en ajoutant un nouvel attribut « Catégorie » à chaque zone.

- Attention: nous avons appelé l'attribut « Categorie » et non pas « Catégorie » car il peut apparaître des problèmes si vous décidez d'utiliser votre .shp sous différentes plateformes (par exemple si vous créez un attribut nommé « Catégories » sous linux, si vous essayer de le lire sous windows, celui ci aura été transformé en «CatÃ©gorie »).
- De manière générale il est déconseillé d'utiliser des accents ou des caractères spéciaux.

Pour modifier les attributs d'un shapefile, il faut modifier le fichier .dbf qui lui est associé. Comme le fichier .dbf sert de base de données contenant les informations propres à chaque élément géométrique du shapefile, on peut directement le modifier si on se contente de rajouter des champs ou d'éditer le contenu de certains champs (sauf celui qui sert d'index).

En revanche si l'on veut ajouter ou supprimer des enregistrements, il faudra recréer un shapefile pour conserver la relation entre le nombre de formes et le nombre d'enregistrements du fichier .dbf.

De manière générale, la modification d'un shapefile est plus rapide en utilisant une jointure, mais on peut également modifier un .dbf en l'éditant directement avec un tableur. Cette dernière solution est à éviter, mais il peut être utile de la connaître si vous n'avez à modifier que quelques lignes.

Solution avec un tableur

- Si vous n'arrivez pas à ouvrir un fichier .dbf avec *OpenOffice*, assurez vous que votre version est à jour et contient *OpenOffice* « bases de données ».

- Vous pouvez ouvrir le fichier .dbf associé à un shapefile avec un tableur comme *OpenOffice* ou *Excel*.
- Une fois le .dbf ouvert avec le tableur, les attributs doivent apparaître en ligne.
  - Chaque ligne représente un élément. Il y a normalement une colonne d'identifiant (qui sert à faire correspondre le fichier .dbf avec les autres fichiers du shapefile) et des colonnes d'attributs.
- Pour ajouter un champ d'attributs il suffit de rajouter une colonne.
- Si vous voulez attribuer les colonnes en fonctions d'un autres attributs, vous pouvez effectuer un tri de la manière suivante (les noms des options correspondent au tableur *OpenOffice*): dans le menu « Données », sélectionnez « trier ». dans la fenêtre qui apparaît, vous pouvez choisir le champ cible et l'ordre (croissant ou décroissant)
  - Une fois le tri effectué, les tuples (lignes d'attributs) sont regroupés selon la valeur de l'attribut cible. Il est alors possible de faire correspondre une valeur du nouvel attribut à toutes les lignes ayant la même valeur de l'attribut ayant servit pour le tri.
  - Pour récupérer l'ordre de départ, on peut ré effectuer un tri en ordre croissant sur la colonne de l'index.
  - Attention: Il est important de penser à fermer le tableur après l'édition d'une table, sinon les autres programmes (dont *QGIS*) pourront ne pas y avoir accès. Exel par exemple verrouille les feuilles de calcules ouvertes et bloque leur accès à d'autres programmes.

Solution avec une jointure

Si le fichier est important, il est plus simple d'utiliser une jointure. Le principe de la jointure est de créer une table par fusion de deux tables indépendantes en fonction d'un attribut commun. *QGIS* permet d'effectuer simplement des jointures et de recréer des fichiers shapefile indépendants résultants de la jointure.

QGIS possède deux extensions utiles pour les manipulations des tables d'attributs, et dont nous pouvons nous servir pour effectuer la jointure.

- Le plugin « Table Manager » n'est pas directement utile pour la jointure, mais permet d'éditer la structure d'une table en ajoutant ou en supprimant des champs. Il peut être utile de modifier une table avant ou après jointure (pour supprimer les champs en double par exemple).
- Si vous avez installé le plugin *FTOOLS*, celui ci contient des fonctions permettant la gestion des attributs d'une table (menu tools => Data Management tools).

Avant de faire la jointure, il faut créer une table de correspondance. C'est à dire qu'elle doit posséder un attribut qui possède des valeurs communes avec un autre attribut du fichier .dbf cible. *QGIS* permet de prendre comme table de correspondance la table d'attribut d'un autre shapefile, ou d'utiliser une table seule. Nous allons utiliser la deuxième solution qui correspond mieux à notre besoin.

- La façon la plus simple de faire est d'ouvrir un tableur, de créer les différentes champs et de renseigner les éléments de jointure, puis d'enregistrer le fichier au format .dbf
  - des erreurs peuvent s'afficher lors de l'enregistrement mais le fichier .dbf est créé.
  - Votre table de correspondance doit posséder un attribut en commun avec votre table de départ, cet attribut va être la clé de la jointure. Dans notre exemple cet attribut sera « zone ».

id	zone	surface	localisation
1	foret	X	X
2	blé	X	X
3	prairie	X	X
4	orge	X	X
5	colza	X	X
6	friche	X	X
7	foret	X	X
8	prairie	X	X
9	blé	X	X
10	blé	X	X

Table 1: exemple d'une table .dbf (on veut associer une catégorie en fonction de la zone):

zone	Categorie
foret	libre
blé	culture
prairie	libre
orge	culture
colza	culture
friche	libre

Table 2: exemple de table de jointure:

L'attribut utile à la jointure (ici 'zone') est utilisé en clé primaire, on associe à chaque valeur une catégorie. (ce tableau est un copié-collé du contenu du .dbf créé avec un tableur). Une fois que l'on a défini la table contenant les attributs que l'on veut rajouter, on peut procéder à la jointure grâce à *FTOOLS*.

- Dans le Menu Tools, sélectionnez « Data Management Tools » puis « join attributes »
- Dans le champ « Target vector Layer », vous devez sélectionner le shapefile dont vous voulez récupérer les données, et indiquer dans le champ « target join field » l'attribut selon lequel la jointure doit s'effectuer (ici « zone »).

- Dans la partie « join data », vous pouvez utiliser un des shapefiles chargés (présent dans le projet), ou utiliser seulement un fichier .dbf en sélectionnant « join dbf table » et en indiquant le chemin de la table de jointure que vous avez créé.

  ☝ Vous devez également préciser le champs de la table qui doit correspondre au premier (il est souvent plus pratique de leur donner le même nom). Dans notre exemple, on va donc faire la correspondance entre les champs « zone » de notre SIG avec le champ « zone » du fichier .dbf contenant les correspondances.

- Il faut ensuite indiquer l'emplacement du shapefile dans le champ « output shapefile » qui sera créé avec la concaténation des deux tables d'attributs et préciser dans la partie «output table» que l'on veut conserver (ou non) les enregistrements de la table de départ qui n'ont pas de correspondance dans la table de jointure.

id	zone	surface	localisation	categorie
1	foret	X	X	libre
2	blé	X	X	culture
3	prairie	X	X	libre
4	orge	X	X	culture
5	colza	X	X	culture
6	friche	X	X	libre
7	foret	X	X	libre
8	prairie	X	X	libre
9	blé	X	X	culture
10	blé	X	X	culture

Table 3: résultat de la jointure

On a maintenant défini une catégorie pour chaque élément du fichier dbf.

Module 6 Réalisation d'un Clipping

Objectif :	Extraire une partie d'un shapefile à partir d'une forme de découpe choisie par l'utilisateur.
Entrée :	Un fichier shapefile
Sortie :	Un fichier shapefile représentant un extrait du fichier en entrée avec une forme définie.
Outils :	*QGIS*

Attention: si vous devez procéder à un changement de Système de référence de coordonnées sur votre shapefile, assurez vous de la faire avant d'effectuer le clipping, sinon votre zone de travail risque d'être déformée

Pour réaliser une capture d'un ensemble de données géographiques, il existe différentes manières de procéder. On peut récupérer le SIG dans son intégralité et procéder à une sélection en la programmant dans le simulateur grâce aux API, ou effectuer une découpe depuis un logiciel de SIG.

Dans le cadre de cette chaîne de traitement, nous allons utiliser une méthode de clipping détaillée pour *QGIS* mais dont le principe est identique sur plusieurs logiciels de traitement de SIG.

Installation d'un plugin de QGIS

- Dans les Plugins de *QGIS*, il faut d'abord installer *FTOOLS*, cette action est très rapide grâce au gestionnaire de Plugins:
- Il faut commencer par activer le Plugin de gestion de Plugin: dans le menu plugin cliquez sur « Gestionnaire de plugin ».
  - ☝ Une fois activé, un élément « récupération des extensions python... » apparaît dans le menu plugin. Il propose alors un certain nombre de modules téléchargeables.
- Plusieurs modules peuvent être utiles, mais celui qui nous intéresse s'appelle « ftools for *QGIS* 1,0 ».sélectionnez le puis cliquez sur « installer/mettre à jour l'extension ». L'installation du Plugin se fait automatiquement, il faut ensuite l'activer en passant par le gestionnaire d'extensions.
  - ☝ Si ftools est correctement installé et activé, Un menu « Tools » doit apparaître.

Pour procéder au clipping, nous avons besoin d'un SIG à découper (celui dont on veut récupérer une partie) et d'une forme géométrique superposable au SIG. Cette forme va servir de filtre de découpe (il peut s'agir d'un fichier de formes géométriques simples ou un autre fichier shapefile). Le shapefile résultant du clipping possédera les polygones et les attributs du SIG, mais sa taille et ses contours seront ceux de la formes géométriques.

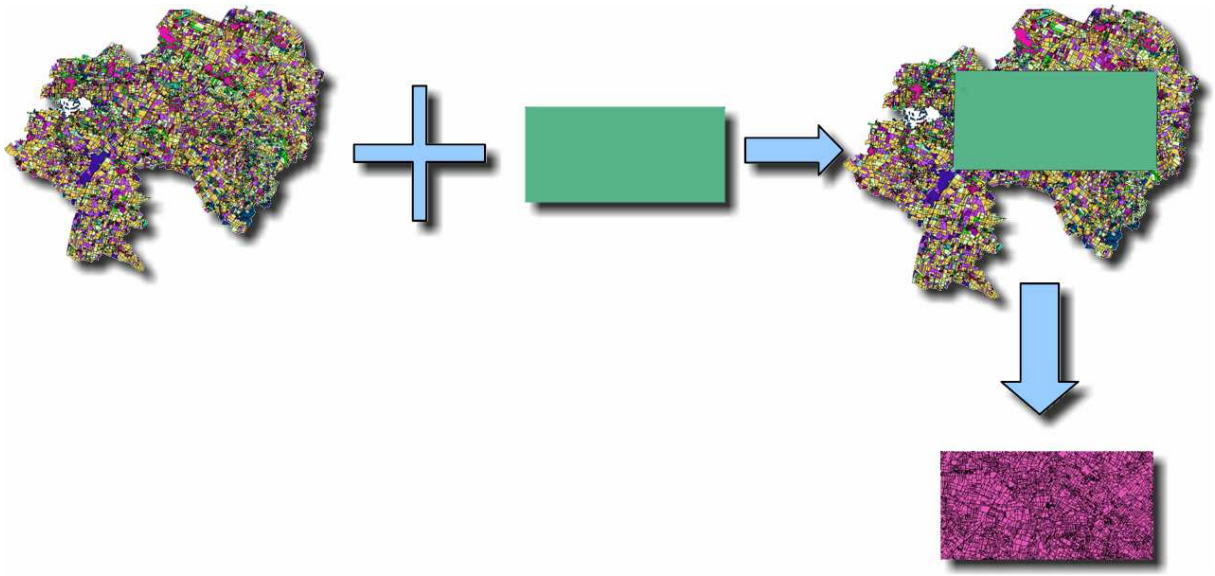


Figure 2: schéma explicatif du principe du clipping

Il existe deux façons de procéder pour créer la forme de découpe. Vous pouvez la créer vous même, en la dessinant, ou vous pouvez l'importer depuis un fichier de forme. Les deux manière de faire sont détaillées ci dessous.

Création de la forme de découpe « à la main » depuis QGIS:

Au moment de la création d'une couche il est nécessaire de lui donner au moins un champ avec un type de valeur.

Une fois le Plugin *FTOOLS* installé et le SIG de base (fichier shp) chargé, on va créer une nouvelle couche vide:

- cliquez sur « ajouter nouvelle couche » de type polygone
- Sélectionnez la couche nouvellement créée dans l'explorateur à gauche.
- Passez en mode édition
- Dans le menu « édition » sélectionnez «capturer le polygone »
  - On va « dessiner » la forme de découpe que l'on va appliquer sur la couche « SIG » , il est possible de dessiner un polygone directement avec la souris.
  - Cette solution permet des formes de découpe très libres. mais il n'est pas évident de créer rapidement un rectangle régulier.
- Pour dessiner le polygone, il suffit de cliquer une fois sur le SIG pour commencer la sélection, puis de faire un clic gauche par sommet en terminant la série par un clic droit.
  - *QGIS* vous propose alors de renseigner les attributs du polygone créé. Cette action est facultative et ne nous sert pas dans notre cas puisque seule la forme du polygone nous intéresse.
  - Attention: Avant de réaliser le clipping , assurez vous que toutes les modifications sur les couches vectorielles soient enregistrées, tant que vous n'avez pas quitté le mode d'édition, la couche contenant la forme de découpe sera considérée comme vide et le clipping ne marchera pas.

Importer un fichier de formes

L'autre solution (si vous devez faire un certain nombre de clipping ou si vous avez besoin de coordonnées précises) est d'importer une forme existante depuis un fichier.

Pour faciliter la création de fichier de forme servant à la découpe, nous avons réalisé une mini-application (le code source (disponible dans le dossier des fichiers sources, fichier `createur_carre_KML`) qui va créer un fichier KML représentant un rectangle depuis un couple de coordonnées (angles supérieur droit et inférieur gauche).

- Lancer l'application
- relever les coordonnées en pointant le curseur sur deux endroits du SIG et de les utiliser pour créer un rectangle.
- Il faut ensuite importer le fichier kml généré comme une couche vectorielle. (bouton 'ajouter une couche vecteur')
  - ☝ Attention à ce que les coordonnées de la forme de découpe correspondent aux coordonnées du SIG et qu'elles soient dans le même format.

Application de la forme de découpe

Une fois que la forme de découpe est créée, on va utiliser une option de *FTOOLS* qui s'occupe de faire le découpage proprement dit du fichier shapefile.

- Dans le menu Tools, sélectionnez « Geoprocessing tools » , puis « Clip ».
- Dans le champs « input vector layer », vous devez sélectionner la couche que vous voulez découper
- Dans le champs « clip layer », vous devez choisir la couche qui contient la forme de découpe.
- Pour finir vous devez sélectionner ou créer un fichier shapefile qui contiendra le résultat du clipping.
  - ☝ Une fois le clipping réalisé, vous pouvez visualiser le résultat en choisissant d'afficher la couche shp créée. Vous pouvez la mettre en évidence en désactivant l'affichage des autres couches grâce au panneau de gauche.

Module 7 Changement de résolution d'un raster

Objectif :	Modifier la résolution d'une image raster (on va simplifier une image et réduire le nombre de pixels utilisés pour la définir).
Entrée :	Un raster (géoréférencé).
Sortie:	un raster simplifié.
Outils:	*QGIS, GRASS*

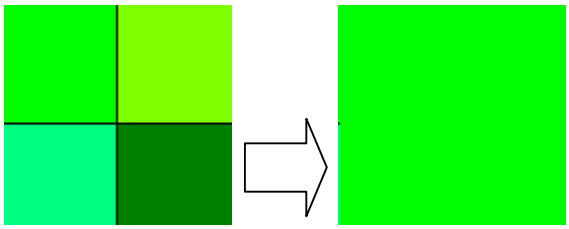
Présentation du problème

Lorsque l'on souhaite travailler avec des rasters, on se trouve parfois confronté à des problèmes de dimensions. Si l'on travaille avec des rasters contenant des données géographiques, on peut rapidement atteindre des dimensions de plusieurs milliers de pixels.

Dans le cadre d'une simulation, une telle précision n'est pas toujours nécessaire, car la gestion d'un tel raster suppose la gestion de millions de cases.

Pour optimiser la rapidité d'exécution de la simulation, on va réduire la résolution de ce fichier raster.

Pour réduire la résolution d'un raster la logique consiste à représenter un ensemble de pixel dans un seul pixel.



Exemple : à un groupe de 4 pixels on associe 1 pixel

Choix de l'algorithme

Pour effectuer cette simplification, il y a plusieurs algorithmes possibles. Une des méthodes consiste à prendre comme valeur du pixel de sortie la moyenne des valeurs des autres pixels.

Cette méthode pose cependant un problème, dans le cas où on travaille avec des valeurs discrètes significatives (1 signifie herbe, 3 signifie arbre, ...) si on effectue la moyenne on peut récupérer des valeurs qui ne correspondent à rien.

L'algorithme de la fonction que nous allons utiliser renvoie au pixel de sortie la valeur du pixel proche du centre de la grille qui contient les pixels que l'on va regrouper.

1	2	3
4	5	6
7	8	9

La valeur du pixel de sortie sera « 5 ».

Cet algorithme est surtout efficace dans le cas où on travaille sur des valeur continues, mais il pose problème dans certains cas ; par exemple dans le cas suivant :

0	0	0
0	9	0
0	0	0

La valeur renvoyée pour le pixel est 9, ce qui n'est pas représentatif des valeurs voisines, la perte de précision est grande.

Pour pallier à ce problème, **avant de faire appel à la méthode de changement de résolution**, nous allons rendre notre image plus floue en harmonisant les valeurs voisines.

On va utiliser une grille plus ou moins fine, que nous allons faire passer sur chaque pixel du raster pour attribuer à chaque pixel la valeur majoritaire des cellules avoisinantes. Ainsi la grille précédente est transformée en celle-ci :

0	0	0
0	0	0
0	0	0

"floutage" de l'image par attribution de la valeur majoritaire

Après cette transformation, les valeurs sont unies, on peut donc prendre comme valeur du pixel de sortie, une valeur située vers le milieu de la grille, l'erreur sera réduite.

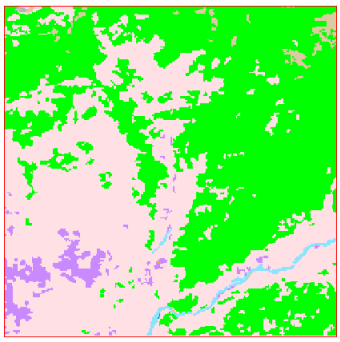
Changement de la résolution avec GRASS.

1. Ouvrir *QGIS*.
2. Si vous n'avez pas de jeu de donnée *GRASS*, vous devez en créer un.
   - La procédure pour créer un nouveau jeu de données est détaillée dans la procédure pour vectoriser un raster p.16.
3. Avant de commencer le traitement, importez votre raster dans *GRASS*. Il faut toujours faire attention à ce que votre jeu de données soit cohérent avec votre image (vérifier le CRS, voir module Modification du système de coordonnées, p.24). Nous allons nous servir de *QGIS* pour la visualisation, il faut donc que celui ci soit aussi dans le bon CRS.

- Pour importer votre raster, ouvrez les outils *GRASS* et dans l'onglet Arborescence des modules choisissez «File->Import->Import Raster-> r.in.gdal»Vous devez ensuite importer votre raster et préciser quel sera son nom dans le jeu de données *GRASS*. Si *GRASS* détecte correctement le raster, il crée une couche unique, si il s'agit d'une seule image,il va créer quatre couches contenants chacune les composantes RGB de l'image.

4. Vous pouvez vérifier que l'import c'est bien passé en ouvrant le raster depuis *QGIS* avec l'option « ajouter une couche raster *GRASS* » dans les options de *GRASS*.

   - Pour la suite de la procédure, nous allons utiliser un raster exemple dont la représentation est la suivante:



5. Nous allons utiliser la console GRASS. Dans l'arborescence des modules, ouvrez la console *GRASS* (le premier module, « shell. grass shell »).

6. Il faut commencer par configurer les paramètres de la région de travail, entrez la ligne de commande:

```
g.region rast= « nom du raster importé dans le
plan de travail »
```

   - Cette commande va associer les paramètres du raster comme paramètre de travail par défaut (coordonnées, résolution...)

   - Il est possible de visualiser les attributs d'une région en utilisant l'option -p, ce qui donne `g.region -p`

7. On va ensuite rendre l'image floue pour que le changement de résolution produise moins d'erreur. Dans la console, entez :

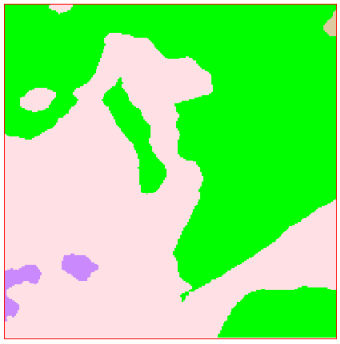
```
r.neighbors input= « le nom du raster importé »
output=« nom de la nouvelle carte » method=mode
size=21
```

   - Dans cette ligne de commande, on a choisi comme méthode « mode » qui consiste a prendre la valeur la plus fréquente.

   - L'attribut size lui correspond au nombre de pixel utilisé pour la grille. Cette valeur correspond au coté de la grille. Pour qu'on puisse la centrer sur une case il faut donc que cette valeur soit impaire. Pour ne pas rendre le calcul trop lourd, la valeur maximum de size est 25. Cependant il est possible d'appliquer plusieurs fois la méthode si nécessaire.

Il faut faire attention à ce que la valeur choisie pour rendre les pixels voisins homogènes entre eux soit cohérente avec le facteur de changement de résolution. Il n'est pas nécessaire de choisir une grille trop grande si le changement de résolution est faible. Au contraire si on rend l'image trop floue on risque de perdre en précision.

Pour mettre en évidence la transformation effectuée, on a exagéré la transformation sur l'exemple: il a maintenant cet aspect :



Résultat du floutage - On s'aperçoit que si la taille de la grille choisie pour rendre le raster flou est trop grande, on risque de perdre des informations (ici le fleuve en bas à gauche était trop fin, il a été écrasé par les pixels environnants qui sont majoritaires).

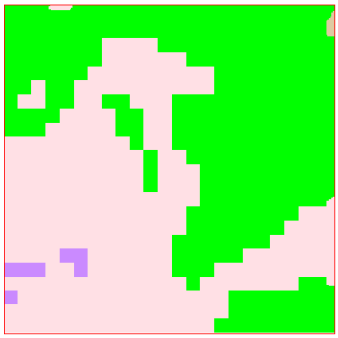
8. A partir de cette image, on va pouvoir effectuer le changement de résolution. Dans la console *GRASS*, il faut changer la résolution de travail par la commande

```
g.region res= « un entier correspondant à la nou-
velle résolution »
```

9. On peut ensuite procéder à la transformation de l'image en utilisant la commande

```
r.resample in= « nom de l'image rendue floue »
out= « nom de l'image de sortie ».
```

10. Vous pouvez alors importer l'image à la nouvelle résolution dans *QGIS* et vérifier la cohérence:



Résultat du changement de résolution : L'image en sortie possède beaucoup moins de pixels, elle conserve cependant un certain équilibre dans la répartition des types de zones.

Module 8 Mise en place d'une simulation Repast Simphony

Objectif : Présenter la logique de la plate-forme *Repast Simphony* et les principes retenus pour l'implémentation.

Outils: *Repast Simphony* et les modules nécessaires à son fonctionnement sont disponibles à l'adresse suivante : http://repast.sourceforge.net/. Un tutoriel sur ses fonctions de base est également disponible à http://repast.sourceforge.net/docs/tutorial/SIM/

Avant de se lancer dans la conception d'une simulation dans *Repast Simphony*, il est nécessaire de se familiariser avec la logique du logiciel. Dans un premier temps, nous allons donc étudier la structure d'une simulation et expliciter la syntaxe de *Repast Simphony*. Pour simplifier les explications, les termes anglais transparents seront traduit, et on appellera « *Repast Simphony* » « *Repast* » pour faire plus court (attention à ne pas confondre avec le logiciel « *Repast* » qui est le prédécesseur de *Repast Simphony*).

Une même simulation peut être programmée de nombreuses manières différentes. *Repast* prévoit deux types d'approche:

1. Soit en utilisant une interface graphique permettant de programmer les interactions et le comportement des agents à l'aide de schémas.
2. Soit en programmant les agents et leurs comportements à l'intérieur de classes *Java*. Cette dernière solution permet de programmer de façon plus proche des agents et permet plus de flexibilité. En effet la programmation en *Java* est orientée objets et permet de mettre en place des interactions entre les agents assez facilement.

La logique de programmation pour la simulation ne nécessite pas beaucoup d'apprentissage si vous avez déjà fait de la programmation orientée objets: On va isoler le gestionnaire de lancement de simulation dans une classe, le gestionnaire de terrain dans une autre classe et enfin les agents eux même seront définis par une classe. La partie programmation consistera à définir les caractéristiques de ces « objets » et leurs interactions.

Pour pouvoir définir une simulation dans *Repast Simphony*, il est nécessaire de mettre en place un certain nombre d'éléments :

Création d'un projet

1. ouvrir Eclipse. Nous allons créer un nouveau projet repast simphony: `File => New => Others => repast simphony=> repast simphony project`.
2. Si tout se passe bien, vous devez voir apparaître un certain nombre de dossiers dans le volet d'exploration à gauche, si il n'est pas visible, vous pouvez l'ouvrir en cliquant sur: `Window => Show view => project explorer.`

   ☝ 4 types de fichier doivent apparaître dans votre nouveau projet :

   a) le dossier contenant votre code source (src)
   b) les archives contenants les librairies
   c) les dossiers utilisés par *Repast Simphony*
   d) des fichiers textes.

Pour l'instant nous allons nous intéresser à un fichier en particulier :

Le fichier model.score

3. Un dossier doit porter le nom de votre projet avec l'extension « rs ». Dans ce dossier, ouvrez le fichier « model.score ».

   - Cette page contient un arbre qui va représenter l'ensemble des éléments que vous allez définir pour votre projet. Pour l'instant il ne doit contenir qu'un élément de type dossier possédant le même nom que votre projet. Il s'agit du contexte global (pour l'instant il ne possède que deux sous objets (attributs et style).
   - Le dossier présent représente le contexte mais il n'est pas « instancié », il est possible de le relier à une classe java que l'on peut programmer pour définir soit même le contexte.

Dans tous les cas, les éléments définissants la simulation (terrains, acteurs, et paramètres de la simulation) doivent apparaître dans un fichier nommé « model.score » situé dans votre projet dans un dossier qui porte le même nom que votre projet mais avec comme extension « .rs ». Ce fichier permet de connaître la structure globale de la simulation et donne une hiérarchie sous forme d'arbre des éléments qui la composent.

Le contexte

L'élément le plus abstrait et le plus basique est le contexte (dans *Repast* « context ») dans lequel les agents vont évoluer.

- Le contexte est immatériel mais englobe l'ensemble de la simulation. C'est l'infrastructure minimum permettant de stocker des agents sans ajouter de notion d'espace ou de relation.
- Un contexte peut posséder des attributs représentant les « lois » qui régissent les agents et l'environnement. On peut voir l'écoulement du temps, la gravité et les lois de la physique en général comme des éléments de notre contexte.
- De plus un contexte peut contenir des sous contextes, en effet l'application des lois change selon le lieu.
- De manière générale, tous les éléments du modèle vont appartenir au contexte principal ou à ses sous contextes.

4. Faites clic droit sur le dossier du contexte et sélectionnez « `show properties` »

   - Un seul attribut nous intéresse et par défaut n'est pas renseigné, ils s'agit de la ligne « class name » qui correspond à la classe java où le simulateur va aller chercher le contexte. On va l'appeler « ContextCreator » qui est assez explicite.

Remarque: A cet endroit il y a deux façons de procéder, soit on repère une classe qui va créer le contexte (cette classe doit implémenter l'interface « ContextBuilder ») soit on repère une classe qui représente le contexte lui même (cette classe doit étendre la superclasse « DefaultContext »).

5. Une fois que l'on a renseigné dans model.score la classe où l'on va créer le contexte, il faut créer la classe correspondante. Sélectionnez le dossier « src ». pour l'instant il ne doit contenir qu'un package vide portant le même nom que votre projet. Faite un clic droit dessus puis sélectionnez `=> new => Class`

   - Éclipse ouvre alors une fenêtre pour la création de votre classe, vous devez recopier à la lettre près le nom que vous avez mise pour le class name du contexte dans le fichier model.score.

Une fois la classe créée vous devez préciser qu'elle doit implémenter l'interface « ContextBuilder ».

- Eclipse va souligner en rouge ce qu'il considère comme faux ou inexistant, heureusement, la plupart du temps il propose une aide en cliquant sur l'icône à gauche de la ligne soulignée.

6. Ici il faut dans un premier temps importer l'interface « ContextBuilder » puis ajouter les méthodes non implémentées (à savoir celle qui retourne le contexte).

```
package tuto;
import repast.simphony.context.Context;
import repast.simphony.dataLoader.ContextBuilder;
public class ContextBuilder implements ContextBuilder
{
    @Override
    public Context build(Context context)
    {
          return context;
    }
}
```

Par défaut, cette fonction renvoi null, nous allons modifier le contexte passé en paramètre et l'utiliser comme paramètre de sortie.

Nous allons y ajouter une projection de type géographie qui servira de modèle pour la simulation.

Les projections

Une fois le contexte défini, il faut définir une zone réelle dans laquelle les agents pourront évoluer. On appelle ces zones des projections. Ce sont des supports permettant aux agents de se déplacer et d'interagir entre eux ou avec leur environnement. Toujours en ramenant ce concept à notre réalité, nous pouvons dire que nous évoluons sur un plan continu en forme de sphère (le globe terrestre).

Dans *Repast Simphony*, il existe quatre types de projections différentes :

1. Les espaces continus

Il s'agit de plans où la position des agents est représentée par des coordonnées réelles (décimales). On peut choisir le nombre de dimensions dans lequel les agents évoluent (1D, 2D , 3D ou ND). Il est peu probable que les positions de deux agents se superposent exactement, mais on peut les faire interagir entre eux selon la distance qui les sépare.

2. Les grilles

Leur fonctionnement est semblable à celui des espaces continus. On peut choisir le nombre de dimensions de la grille, qui va en fait correspondre aux dimensions d'une matrice de cases indexées par des coordonnées entières. L'utilisation des grilles est plus simple que celle des espaces continus, notamment en ce qui concerne les déplacements et les interactions entre agents (il peuvent interagir lorsqu'il arrivent sur la même case par exemple)

3. Les géographies

Ce mode de représentation est plus compliqué à utiliser que les deux précédents mais il permet une gestion de l'affichage des SIG. Dans cet espace, tous les éléments sont représentés par une géométrie (dans *Repast* « geometry », voir glossaire, p.71) particulière. Les éléments du même type doivent apparaître avec la même géométrie.

Pour illustrer cette représentation, nous pouvons prendre pour exemple un SIG simple sur lequel évolue un type d'agent. La construction se fera comme si il y avait deux couches, celle regroupant les éléments terrains (qui

auront une géométrie de type polygone) et celle regroupant les agents d'un même type (que l'on prendra la plupart du temps de type point).

> ☝ Remarque : La gestion des géométries à été reprise de la librairie de JTS Topology Suite. Plus d'informations et de documentations sont disponibles à l'adresse http://www.vividsolutions.com/JTS/JTSHome.htm.

Note : Un quatrième type n'a pas été traité ici, il s'agit des graphes ou réseaux.

Gestion de plusieurs types d'espace/projections

A.- Interface Ground_Manager

La manipulation des agents et leurs interactions avec leur environnement ne se fait pas de la même façon selon le support sur lequel ils évoluent mais les rôles attendus sont les mêmes. Ainsi, le terrain doit pouvoir remplir les services suivants :

- Retrouver les coordonnées d'un élément
- Donner la liste des objets et agents présents dans une portion donnée
- Permettre le déplacement de certains objets
- Gérer les frontières de la simulation

Dans la suite de cette chaîne de traitement, nous allons essayer de conserver un maximum de généricité en déclarant une interface « Ground_Manager » qui contiendra les méthodes abstraites permettant de remplir les rôles que l'on attend du terrain. Cette astuce permet de **faire en sorte que les agents puissent faire des requêtes sur le terrain sans connaître la nature de celui ci**. Pour changer le support de la simulation il suffira donc d'envoyer aux agents une référence sur un gestionnaire de terrain particulier.

Lors de la simulation, l'agent pourra demander au terrain :

- «Quelle est ma position (quelles sont mes coordonnées) ? »
- «Je veux me déplacer de x mètres dans la direction y »
- «Quels objets puis-je percevoir depuis ma position ? ».

Les agents devront donc posséder des caractéristiques indépendantes du terrain (par exemple un déplacement en mètre par heure et un champ de vision exprimé en mètres). Le gestionnaire de terrain s'occupe ensuite des conversions nécessaires pour le traitement.

B.- Traitements propres à chaque type

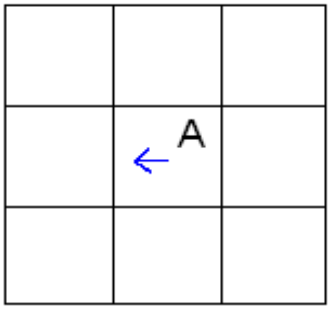
1. Dans un espace continu, les coordonnées des agents sont exprimées par une suite de nombres décimaux. Il faut donc :
   - rajouter des tests pour savoir à quelle distance sont les agents les uns des autres,
   - effectuer des parcours pour tester la présence des agents dans une zone de l'espace continu...

De plus si l'on veut faire correspondre des données terrain, il faut définir chaque zone. Dans ce cas, il est donc plus simple d'utiliser une grille.

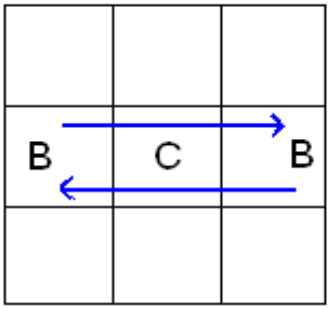
2. Dans le cas d'une grille :
   - on se repère grâce aux cases et aux éléments qu'elles contiennent.
   - Les coordonnées sont donc exprimées par des valeurs entières délimitées par les dimensions de la

grille.

- Les agents doivent avoir un déplacement en cases et chaque case doit représenter un certain nombre de mètres (ou d'une autre unité).
- De plus leur vitesse doit correspondre au terrain, ce qui peut se révéler délicat car les déplacements en cases évoluent si l'on augmente ou diminue la vitesse des agents en mètres et si l'on modifie l'intervalle de temps représenté par un tic.



Situation 1



Situation 2

Dans la situation 1, si le déplacement de l'agent en mètres est trop faible, l'agent ne bougera pas d'une case car son déplacement est inférieur à la taille d'une case. Donc ses coordonnées ne vont pas évoluer (les coordonnées sont entières, un déplacement inférieur à 1 sera ignoré).

Dans la situation 2 : l'agent a un déplacement qui correspond à deux cases, L'agent B ne pourra jamais atteindre son objectif au point C, il oscillera donc indéfiniment.

3. Dans le cas d'une projection de type Géographie, le traitement est moins intuitif, la gestion se fait principalement par traitement géométrique.
   - Tous les agents sont associés à des géométries. Il n'est pas possible de les déplacer directement. Il faut déplacer leur géométrie.
   - Si l'agent est un point, on peut récupérer ses coordonnées, si il représente un polygone terrain, on peut récupérer son centroïde (c'est à dire un point situé aux coordonnées moyennes de tous les points) ou un point à l'intérieur du polygone, ce qui est plus réaliste dans certains cas.
   - Pour tester leur interaction, on peut rechercher le terrain qui possède une géométrie associée en intersection avec la géométrie d'un agent.
   - Pour tester les élément vus par un agent, on créé un polygone en forme de rond autour de l'agent et on récupère la liste des objets dont la géométrie associée est en intersection avec ce rond.

     ☝ Par défaut, *Repast Simphony* ne permet pas d'utiliser directement des fichiers .shp comme support, mais il est possible d'utiliser les API de *Geotools* (intégrées à *Repast Simphony*) pour pouvoir intégrer rapidement le SIG que nous avons récupéré. Une aide détaillée sur l'utilisation de Geotools et de la gestion des SIG est disponible à l'adresse http://docs.codehaus.org/display/GEOTDOC/Home

C.- Combinaison des types

Il est possible d'utiliser dans une même simulation plusieurs projections différentes et de différents types (espace continu, grille ou géographie). Cependant, l'interface graphique impose des restrictions et créer plusieurs configurations est soumis aux règles suivantes :

- Elles peuvent êtres de 3 types : représentation 2D, 3D ou GIS

- Les représentation 2D ou 3D peuvent représenter simultanément des projections grilles et espaces continus mais pas de géographie. Cependant, le cumul de différents supports risque de masquer certains éléments.
- Le type de représentation GIS ne peut contenir qu'une seule projection de type géographie mais ni espace continu ni grille.

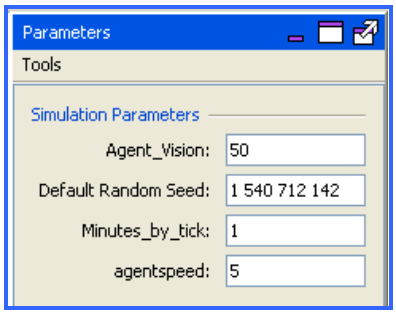
Les différentes projections ont toutes leurs avantages et inconvénients. Pour représenter des agents se déplaçant sur différents types de terrains, deux solutions sont envisageables :

1. La première solution serait de combiner les types de projections 'grille' et 'espace continu'. Ce qui nous permet de bénéficier des qualités de la grille pour référencer rapidement les emplacements des terrains et des qualités des espaces continus qui est beaucoup plus souple pour la gestion des coordonnées.
2. La deuxième solution est d'utiliser le mode de représentation GIS et de travailler sur les données d'un shapefile.

Créer et modifier des paramètres pour la simulation

Pour rendre une simulation souple et étudier différents cas, il est préférable de pouvoir modifier certains éléments de la simulation, par exemple la vitesse de déplacement des agents et le temps représenté par un « tick ».

Dans le Fichier model.score, on retrouve un élément « Attributes ». Il permet de stoker les paramètres utiles lors de la simulation. Pour créer un paramètre, il suffit de faire un clic droit et de choisir « create member – scalar attribute ». Vous pouvez ensuite modifier le nom de ce paramètre, son type et lui donner une valeur par défaut. Ces paramètres sont ensuite modifiables depuis l'interface graphique avant d'initialiser une simulation en cliquant sur l'onglet « parameters » (voir exemple ci-dessous).



Module 9 Implantation d'un SIG (geography) dans Repast Simphony

Objectif: réaliser une simulation basique basée sur un support SIG (fichier shapefile).
Entrée : un fichier shapefile
Sortie : un display dans le GUI de *Repast Simphony*
Outils : *Repast Simphony*

Création d'une projection

7. Faites un clic droit sur le dossier présenté dans model.score et cliquez sur « `create Member »=> projection – Geography`.

   ☝ Normalement un nouvel élément apparaît dans l'arborescence du dossier et s'appelle « geography ».

8. Vous pouvez le renommer en faisant un clic droit dessus => `show properties` et en modifiant la ligne « Label »

> Attention A ce stade il est important de choisir une convention pour les noms. Il est préférable qu'ils soient explicites, relativement courts, sans caractère spéciaux et sans espace.
>
> Il faudra établir la correspondance des éléments de fichier model.score avec le code correspondant par son nom, mais si celui ci est mal écrit, repast n'indiquera pas d'erreur.

Nous avons inscrit une projection dans le contexte, mais **elle est seulement déclarée**, il faut la « construire » dans le contexte.

Pour conserver la généricité du code, on va utiliser un « gestionnaire de terrain ». Pour commencer nous allons utiliser un terrain de type geography.

Nous allons donc créer une classe « SIG_Manager » qui va implémenter l'interface « gestionnaire de terrain » (voir Ground_Manager, p.54) et qui sera spécialisée dans le traitement de projection de type géographie.

La classe « SIG_Manager » devra donner une implémentation de toutes les méthodes définies dans l'interface. Mais avant de définir les méthodes de traitement relatives au terrain, nous allons créer le terrain. Nous allons donc rajouter la méthode « create_ground » qui va recevoir en paramètre le nom du contexte dans lequel on va ajouter la projection « geography » et la chaîne de caractère qui définir le nom de la projection (il doit être identique à celui mentionné dans le fichier « model.score ».Une fois que l'on aura ajouté la projection au contexte,on peut renvoyer le contexte modifé.

```
public class GIS_Manager implements Ground_Manager
{
public Context create_ground(Context context,String GeographyName)
      {
      }
@Override
      public void add_agents(Context context, int nb_agent) {
      }
      @Override
      public ArrayList<Object> find_object(Agent a, double radius) {
           return null;
```

```
        }
        @Override
        public Coordinate get_internal_point(Field_Agent fa) {
            return null;
        }
        @Override
        public Coordinate give_object_coordinate(Object o) {
            return null;
        }
        @Override
        public void move_Object(Agent a, Coordinate c) {
        }
    }
```

Pour créer la géographie, ajoutez le code suivant dans la méthode « create_ground » :

```
//on crée les paramètres par défaut de la geography
GeographyParameters geoparam = new GeographyParameters();
/*puis on crée la géographie elle même en fonction de son nom, du
contexte et des paramètres. On utilise la classe GeographyFactory-
Finder pour créé une nouvelle « usine à projection de type géogra-
phie» qui va à son tour crée la geography voulu en fonction des
paramètres.*/
Geography geo = GeographyFactoryFin-
der.createGeographyFactory(null).createGeography("Geography",conte
xt,geoparam);
```

La géographie est créée mais elle est encore vide, on va y ajouter le SIG à partir d'un fichier shp

Importation du shapefile

Repast Simphony comporte plusieurs librairies qui parfois ont des fonctions similaires, il existe plusieurs façon d'importer un shapefile, en particulier en utilisant directement les classes gérant les shapefiles (ShapefileReader), mais il n'est pas facile par la suite de rattacher les données géométriques des polygones avec leurs attributs associés, on va donc utiliser un objet « **Datastore** » qui va permettre de regrouper les données du shapefile (données et attributs) dans une variable se comportant comme une base de données. Le principe ressemble à celui utilisé dans *PostGIS*, on transforme chaque élément en ligne d'attributs.

```
//on importe le fichier shp en indicant son url
  File file = new File("data_sig/essai_clip.shp");
//on effectue un test pour s'assurer que le fichier est bien chargé
        if ( file == null)
        {
        System.out.println(" erreur de chargement du fichier");
        }
//on utilise un objet "map" pour enregistrer les paramètres qui pôurront
êtres utiles lors de la création de la database, en particulier l'url du
fichier shp
        Map<String,Serializable>
        connectParameters = new HashMap<String,Serializable>();
        connectParameters.put("url", file.toURI().toURL());
          connectParameters.put("create spatial index", true );
        //on crée la database.
          ShapefileDataStore dataStore = (ShapefileDataStore) DataSto-
reFinder.getDataStore(connectParameters);
```

L'objet datastore contient toutes les données, du fichier shp et du fichier dbf et conserve leur relation.

Chaque objet du datastore est appelé « **feature** », chaque feature contient la geométrie qui lui est associée (polygone) et les attributs correspondants.

Pour pouvoir les incorporer dans le SIG, on va récupérer ces features et effectuer un parcours sur chacun d'entre eux en les associant avec des agents de type terrain que l'on appellera « Agent_Parcelles ».

Avant de faire cette association, on va déclarer et créer le type Agent_Parcelle

9. Dans le fichier model.score faite un clic droit sur le contexte et créer un nouveau membre agent
10. Dans les propriétés de l'agent crée, modifiez le label en « Agent_Parcelle ».

Une fois que l'agent est créé dans model.score, vous devez rajouter au moins un attribut

11. Dans les sous éléments de l'agent créé, faites un clic droit sur « Attributes » et choisissez « `create_member – scalar attribute` ». vous devez alors renseigner le nom de l'attribut, son type et éventuellement une valeur par défaut.

    ☝ Une fois l'agent déclaré, il faut créer la classe correspondante dans les sources.(clic droit sur le package portant le nom du projet dans le dossier src, `new => class`). Il faut donner le même nom à la classe que celui donné dans model.score.

Attention: Lorsque vous créez un agent, qu'il représente un terrain ou un agent mobile, vous devez impérativement lui donner au moins un attribut dans sa classe java avec les « getters » et « setters » correspondants avec un attribut associé à l'agent dans le fichier model.score. Si vous ne le faites pas vous ne pourrez pas donner de style à vos agents et lorsque vous lancez la compilation vous risquez d'obtenir l'erreur suivante

```
log4j:WARN No appenders could be found for logger (MessageCen-
ter.INTERNAL.repast.simphony.ui.RSApplication).
log4j:WARN Please initialize the log4j system properly.
```

Cette erreur revient régulièrement, elle peut avoir plusieurs origines

- soit vous avez mal associé les noms (paramètre agent ou projection) entre le fichier model.score et les classes
- soit vous avez mal définit les types d'agents lors de la configuration de l'affichage (display)
- soit vos agents ne possèdent pas d'attributs avec les accesseurs correspondants.

Maintenant que les agents Parcelles existent, nous allons les utiliser pour créer les différents éléments du SIG. il faut reprendre la méthode « create_ground » là où nous l'avions laissé :

```
//on récupère les types de fichier stocké dans la database
        String[] typeNames = dataStore.getTypeNames();
        //on récupère le 1er type ( celui qui nous interesse)
        String typeName = typeNames[0];
        System.out.println("Reading content " + typeName);
//on créé un objet featuresource qui va nous permettre de récupé-
rer les différents éléments du fichier shp (chaque instance de
feature va contenir un polygone avec les données qui lui sont as-
sociées)
    FeatureSource fs = dataStore.getFeatureSource(typeName);
//on crée une collection pour stocker les "features"
    FeatureCollection collection;
//on crée un objet iterator pour effectuer un parcours sur tous
les features
    FeatureIterator iterator ;
    collection = fs.getFeatures();
    iterator = collection.features();
```

```
//on enregistre les dimensions de l'enveloppe du shapefile pour
pouvoir //délimiter les frontières (on en aura besoin pour placer
et contenir les //agents
    Envelope envelope = collection.getBounds();
```

- Dans cette portion de code, on récupère les dimensions occupées par la collection, ces dimensions sont utilisées pour créer l'enveloppe qui contient le SIG. Ces dimensions peuvent être utiles si l'on veut placer des agents dans le SIG ou les faire se déplacer à l'intérieur sans qu'ils puissent sortir. Mais il faudra effectuer des tests supplémentaires si le shapefile n'a pas une forme rectangulaire.

Une fois qu'on a crée un objet « itérator », il faut effectuer un parcours pour insérer chaque élément

```
while (iterator.hasNext())
      {
      //on récupère un élément du shapefile
      Feature feature = iterator.next();
      //on récupère le polygone qui y est associé
      Geometry geometry = (Geometry) feature.getDefaultGeometry();
      //à ce stade, on peut accéder aux attributs des éléments du
      //shapefile en utilisant les méthodes de feature:
      //on récupère les attributs qui nous intéressent:
      int x =          Inte-
ger.parseInt(feature.getAttribute("Catégorie").toString());
      //et on construit l'agent correspondant qui aura comme géométrie le
      //polygone du SIG qui lui est associé et les attributs dont
      // nous pouvons nous servir (les caractéristiques de la zones)
      a = new Agent_Parcelle(x);
      geo.move(a,geometry);
      //cette méthode ne déplace pas réellement un agent
      //mais l'associe avec une géométrie qui elle possède des cordonnées.
      }
```

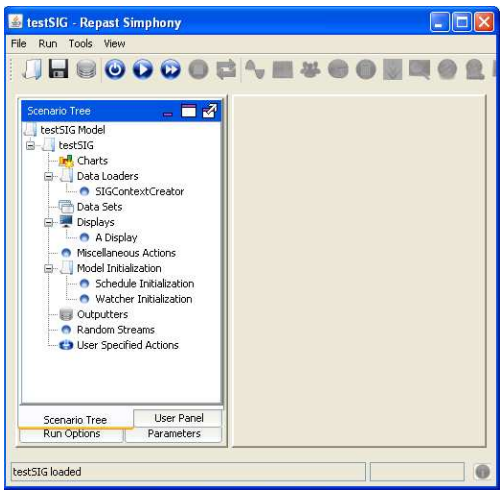
- si vous ne connaissez pas les noms des attributs de votre shapefile, vous pouvez utiliser la ligne: *System.out.println(feature.toString())*. Chaque feature va alors afficher tous ses attributs dans la console. Cette méthode peut être pratique pour vérifier que les noms des attributs n'ont pas changé. lorsque vous changez de plateforme (*Linux* vers *Windows* par exemple) certains caractères ne sont pas bien reconnus, par exemple l'attribut « Catégories » peut devenir « CatÃ©gorie » et si l'on essaie de récupérer la valeur de l'attribut « Catégorie », celui ci n'existe plus, ce qui provoque une erreur.

Affichage du SIG dans le simulateur

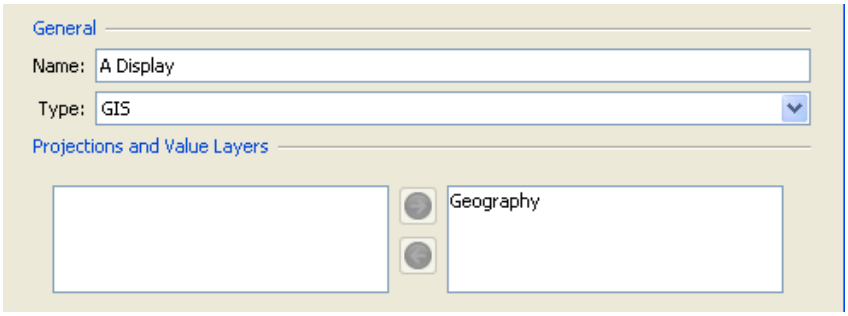
A ce stade, il est normalement possible d'afficher le SIG dans son intégralité

12. Cliquez sur la flèche à coté bouton de lancement et sélectionnez 'Run « votreprojet » model'.

- Au bout de quelques secondes la fenêtre du simulateur doit s'ouvrir.



13. Faites un clic droit sur DataLoaders et utilisez setDataLoaders.
14. Dans la nouvelle fenêtre sélectionnez « a specific java class » puis le nom de votre constructeur de contexte (« ContextCreator »).
15. Il faut ensuite configurer l'affichage, faites un clic droit sur « Displays » puis cliquez sur « add display »
16. Sélectionnez le type d'affichage GIS et faite passer votre geography dans le champ de droite :



17. cliquez sur suivant
    - 'Normalement' dans la colonne à gauche vous devez voir les agents que vous avez crées. Il faut les changer en types polygones.
    - En éditant le styles des agents il est possible de leur donner une échelle de couleur en fonction d'un attribut.
18. Cliquez sur suivant puis « finish »
19. Pour vérifier votre model il suffit de l'initialiser en cliquant sur le bouton « initialize run » : 
    - Le modèle met un certain temps à se charger en fonction de la taille du projet et du shp.
    - Une fois chargé, le contenu du fichier doit s'afficher.

Module 10 Intégration d'un raster (grid) dans Repast Simphony

Objectif: réaliser une simulation basique basée sur un support raster.
Entrée : un raster dans un format image sans perte (png ..)
Sortie : un display dans le GUI de *Repast Simphony*
Outils : *Repast Simphony*

Présentation

Les rasters sont très utiles pour représenter des couches de données. On considère que chaque pixel représente une surface et que les valeurs de ces pixels ont une signification (hauteur, température, chlorophylle...).

La solution la plus simple pour utiliser un raster comme support est d'utiliser une grille de même dimension que le raster et d'associer à chaque cellule un agent parcelle construit à partir de la valeur du pixel correspondant dans le raster. On peut alors positionner les agents à l'intérieur de ces agents parcelles.

Cependant plusieurs problèmes apparaissent rapidement:

1. Le premier problème est propre aux grilles, les agents se déplaçant en mètre, leur associer un déplacement en case n'est pas évident (surtout quand on peut faire varier la taille d'une case, la vitesse de l'agent et la période représentée par un tick de temps).
2. Le deuxième problème vient de la lourdeur d'affichage. En effet si l'on souhaite afficher la grille d'agent mobile, le logiciel va parcourir chaque agent terrain à chaque pas de temps pour le redessiner (même si l'agent terrain ne fait rien). Un petit raster 120 pixels par 120 pixels contient 14.400 pixels. Si le simulateur doit redessiner 14.400 agents à chaque pas de temps, la simulation sera très lente.

Pour résoudre ces deux problèmes, nous allons combiner différents procédés.

1. Pour pouvoir conserver la gestion des agents avec des conversions simples, nous allons utiliser une projection de type espace continu, pour contenir les agents « mobiles ».
2. Pour conserver les interactions entre les agents et le terrain, nous devons conserver les agents parcelles. Nous allons donc les stocker dans une matrice qui aura les mêmes dimensions que le raster. Mais ces agents parcelles ne seront pas représentés.
3. Pour conserver une représentation du terrain, nous allons utiliser un objet « GridValueLayer ». Il fonctionne comme une grille normale, mais ne contient que des valeurs simples. Une fois que cet objet sera construit, on pourra l'afficher comme une image raster.
    - ☝ En revanche, il va falloir définir une classe de style pour faire correspondre à chaque valeur une couleur.

Attention: Il est important de toujours penser à faire correspondre les données des classes java avec le schéma défini dans le fichier model.score. Il faut donc ajouter la projection de type espace continu et le GridValueLayer dans les éléments du contexte défini dans model.score. Il faut également faire en sorte que les libellé se correspondent entre le nom donné dans le fichier model.score et la chaine de caractères servant à repérer l'élément dans le code.

L'en tête de la classe Raster_manager aura donc cet aspect:

```
public class Grid_Manager implements Ground_Manager
{
  private static    Map<Integer,Color> map;
  private double  size_of_one_box = 3;
  GridValueLayer grille = null;
  ContinuousSpace space = null;
  Field_Agent[][] matrice_terrain;
        public Context create_ground(Context context,String
                              url,String geographyname )
  {
            //we load the data in an integer matrix from the
            file at the       specified URL
            int[][] matrice = img_raster_loader(url);
            //building of  the projection's elements
            .
            .
            .
            return context;
  }
}
```

Lecture des données d'un raster

Avant de construire la projection, on va récupérer les données qui nous intéressent dans une matrice :

```
private int[][]img_raster_loader(String url)
    {
    int[][] matrice = null ;
    int[] pixel = new int[3];
    try {
         //we get back the bufferedImage which represent a raster
         BufferedImage img =  ImageIO.read(new File(url));
         if (img == null)
         {
               System.out.println(" l'image est null");
         }
         //we can take the colormodel if we want to draw the raster with the
         same color //as the original image
         IndexColorModel c =(IndexColorModel)img.getColorModel();
         //we transform the colormodel in a map
         Map<Integer,Color> colorMap =new HashMap<Integer,Color>();
         for (int i = 0; i < c.getMapSize() ; i++)
         {
               colorMap.put(i, new Color(c.getRGB(i)));
         }
         map = colorMap;
         //we read the raster from the bufferdImage
         Raster r = img.getData();
         //we make a matrix with the same Dimension than the raster

         matrice = new int[r.getWidth()][r.getHeight()];

         //for each cells of the matrix, we associate the value of the pixel
         // here we know befor which data we were going to read, but it may
         // be recorded in another format
         for (int i = 0 ; i < matrice.length; i++)
         {
               for (int j = 0 ; j < matrice[0].length; j++)
               {
               Object tab = null;
```

```
                Object o =  r.getDataElements(i,j,tab);
                r.getSampleModel().getDataType();
                if ( o != null && o instanceof byte[])
                {
                byte[] tab_byte = (byte[])o;
                            matrice[i][matrice[0].length -j -1] =
                            (int)tab_byte[0];
                }
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        return matrice;
}
```

- Lors de la création, on lit l'image du coin supérieur gauche au coin inférieur droit. On doit donc adapter le parcours pour construire un tableau cohérent (puisque l'on parcourt les tableaux du coin inférieur gauche vers le coin supérieur droit).

Création du sol

Maintenant que nous avons récupéré nos données, nous pouvons compléter notre classe Create_Ground:

```
public Context create_ground(Context context,String url,String
geographyname )
      {
      //we load the grid in an array of integer
      int[][] matrice = img_raster_loader(url);
      //we create a matrix which will contains the Field_Agent
          matrice_terrain = new
          Field_Agent[matrice.length][matrice[0].length];
      Dimension dim = new Dimension(matrice.length,matrice[0].length);
      //we create the gridValueLayer
      grille = new GridValueLayer("ScalarField", true,
             new repast.simphony.space.grid.WrapAroundBorders(),
                                                    dim.width,
dim.height);
      context.addValueLayer(grille);
          ContinuousSpaceFactory continousfactory =
          ContinuousSpaceFactoryFinder.createContinuousSpaceFactory(new
          HashMap());
          space = continousfactory.createContinuousSpace("space", context,
          new SimpleCartesianAdder(),new
          WrapAroundBorders(),grille.getDimensions().getWidth(),grille.getDi
          mensions().getHeight());
      //on parcours tous les éléments de la grille pour les faire
          //correspondre avec les valeurs enregistrées dans la matrice que
          // l'on a chargé.
      for (int i = 0; i <grille.getDimensions().getWidth();i++)
            {
                        for (int j = 0; j
                        <grille.getDimensions().getHeight();j++)
                {
                      grille.set(matrice[i][j], i,j);
                      //on créé un agent terrain avec la
                                  //valeur correspondante
                                  matrice_terrain[i][j] = new
                                  Field_Agent(matrice[i][j]);
                }
```

```
            }
            return context;
      }
```

Redéfinition du style

Nous allons ensuite définir le style permettant de représenter notre « GridValueLayer ».

La plupart des méthodes sont des méthodes de *Repast* que l'on redéfinit.

- On utilise le constructeur pour associer le style à notre valueLayer.
- Puis on charge la table de correspondance que l'on a récupéré lorsque l'on a chargé le fichier pour associer une couleur à chaque valeur.

```
Public class Style2d_terrain implements ValueLayerStyle
{
      protected ValueLayer layer;
      Map<Integer,Color> colorMap;
      public Style2d_terrain()
      {
      //we get back the colorModel we have read during theinitialisation
            colorMap = Grid_Manager.getmap();
      }
             @Override
      public void addValueLayer(ValueLayer layer)
      {
            this.layer = layer;
      }
             @Override
      public int getRed(double... coordinates)
      {
            return colorMap.get((int)layer.get(coordinates)).getRed();
      }
             @Override
      public int getBlue(double... coordinates)
      {
            return colorMap.get((int)layer.get(coordinates)).getBlue();
      }
             @Override
      public int getGreen(double... coordinates)
      {
            Return colorMap.get((int)layer.get(coordinates)).getGreen();
      }
             @Override
      public float getCellSize()
      {
            return cell_size;
      }
             @Override
      public Paint getPaint(double... coordinates)
      {
            return colorMap.get((int)layer.get(coordinates));
      }
}
```

Configuration de l'affichage

Une fois tous les éléments mis en place, il faut configurer l'affichage:

- Lancez la simulation puis faites un clic droit sur display puis sélectionnez « `new Display` ».
  - ☝ l'affichage voulu est de type 2D.
- On va afficher l'espace continu et le GridValueLayer. Faites donc passer ces deux éléments dans la colonne de droite.
- L'étape suivante correspond au style des agents. Si vous en avez créé, vous pouvez éditer leurs styles et gérer les priorités d'affichage.
- Il faut ensuite donner des paramètres de l'espace continu. Il n'y a pas grand chose à modifier, mais il faut faire attention à ce que l'échelle soit la même pour le plan continu et pour la grille de valeur (l'échelle du plan continu modifiable grâce au champ « `unit size` », et la taille d'une cellule de la grille est modifiable dans la classe de style avec la méthode « `getCellSize()` ».
- L'affichage suivant propose de donner un style au GridValueLayer. Dans la liste déroulante, sélectionnez le style que vous avez créé dans une classe java.
- Finalement, vous pouvez cliquez sur `next` puis `finish`.
  - ☝ Normalement, si vous lancez la simulation, votre image doit apparaître en arrière plan.

Module 11 Gestion des agents d'une simulation (ajout, mouvement, interaction)

Objectif:	Organiser les interactions et les déplacements des agents sur le support
Entrée :	Une simulation *Repast Simphony* qui intègre les fonctions de traitement du module précédent.
Sortie :	Une simulation qui prend en charge des agents capables de remplir un minimum de fonctions.
Outil :	*Repast Simphony*

Une fois que l'on a défini le support de la simulation, nous allons créer les acteurs de la simulation : les agents.

Quelques remarques sur les contraintes de généricité

1. **Pour conserver une certaine généricité, nous allons faire en sorte que les agents ne connaissent pas la nature de leur support**. Nous allons donner aux agents une référence sur un objet « ground_manager » qui s'occupera de toutes les requêtes de l'agent concernant son environnement. L'agent ne doit pas avoir « conscience » des éléments de programmation qui l'entoure, il possède juste une référence sur les actions possibles. Le ground Manager est chargé d'appliquer ces actions.
2. Pour pouvoir utiliser différents types d'agents (exemple : loup, mouton) en réutilisant certaines méthodes (exemple : se déplacer). On utilise généralement l'héritage, mais un problème apparaît avec la gestion de *Repast Simphony*. En effet en mode de représentation GIS, tous les agents du même type doivent posséder le même type de géométrie. Cette contrainte s'applique même sur les classes héritées. On ne pourra pas donc pas utiliser des méthodes d'héritage pour créer des agents de type différent. Puisqu'il n'est pas possible d'utiliser l'héritage, on va donc utiliser une interface « Agent » qui va servir de marqueur.

   ☝ On peut utiliser le nom d'une interface comme type de variable, ce qui signifie que l'on accepte en paramètre toutes les classes qui implémentent cette interface. Ainsi, on pourra utiliser des méthodes d'interactions des agents avec d'autres agents ou des agents avec le terrain sans tenir compte du nom de la classe (il suffit que celle ci implémente l'interface « agent »). On peut créer autant de classe représentant un type particulier d'agent que nécessaire, ce seront toujours des « Agents ».
3. De la même façon, pour pouvoir traiter différents supports, nous allons donner aux agents, dans une procédure d'initialisation, une référence sur un objet dont la classe implémente l'interface « Ground_Manager ». Les agents pourront appeler les méthodes définies dans l'interface Ground_Manager sans connaître la classe de l'objet appelé (et donc la nature du support).

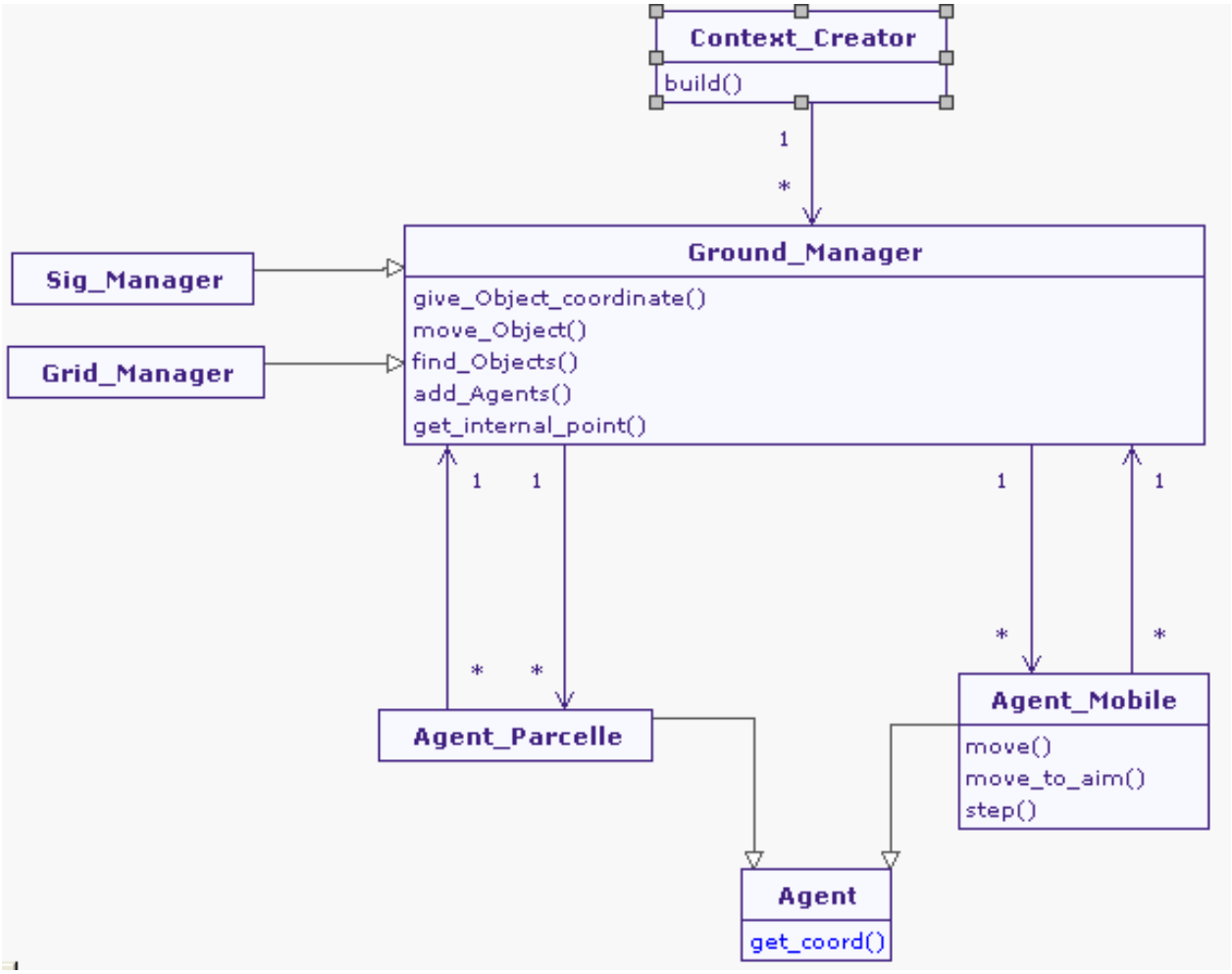


Diagramme de classe représentant la structure des éléments de la simulation

Définition de l'interface Ground_Manager.

- Avant de commencer à programmer le comportement des agents et les méthodes correspondantes dans leur gestionnaire de terrain, il faut définir quels sont les rôles que doivent remplir les agents et le gestionnaire de terrain.
- Les agents sont les éléments les plus sujets à des modifications, on va essentiellement définir les interactions de base permettant aux agents de connaître leur environnement et de pouvoir interagir avec. Une fois que l'on aura défini ces fonctions de base, on pourra les utiliser pour créer des interactions plus complexes.
- La création d'un agent dans la simulation ne peut pas se faire sans savoir sur quel type de support on le place, la fonction de placement des agents doit donc être prise en charge par le gestionnaire de terrain (le gestionnaire de simulation - ContextCreator - gère normalement la création)
- Un agent doit pouvoir connaître son emplacement. Comme ses coordonnées vont dépendre de la nature du terrain, il faut que ce soit le Ground_Manager qui soit capable de renvoyer les coordonnées d'un objet donné.
- Un agent doit pouvoir se déplacer. Or, là aussi le déplacement va dépendre du support. L'agent va donc demander au Ground_Manager de le déplacer (selon une distance choisie par l'agent en mètre). C'est au Ground_Manager de s'adapter et d'effectuer les tests et les conversions nécessaires pour adapter le déplacement de l'agent dans le terrain.

- Un agent possède une certaine représentation de son environnement, puisque le terme de distance dépend également du terrain, le gestionnaire de terrain doit pouvoir renvoyer à l'agent tous les objets qui lui sont visibles selon un champ de vision donné (spécifié par l'agent en mètres).
- Pour finir, puisqu'un agent peut se déplacer à l'intérieur des terrains, il faut que le gestionnaire de terrain puisse lui envoyer non pas les coordonnées du terrain où veut se rendre l'agent, mais un point à l'intérieur du terrain.

Si on récapitule les quelques informations ci dessus, on peut réaliser l'interface Ground_Manager :

```
Public interface Ground_Manager
{
	/**
	 * @param o an object contained in the projection
	 * @return the coordinate of the object in the projection
	 */
	public Coordinate give_object_coordinate(Object o);
	/**
	 * @param fa a reference to an instance of field agent
	 * @return a point within the area of the field agent
	 */
	public Coordinate get_internal_point(Field_Agent fa);
	/**
	 * @param a : the object to move
	 * @param c : the destination coordinate
	 */
	public void move_Object(Agent a,Coordinate c);
	/**
	 * @param a : the agent which is looking around him
	 * @param radius the field of view
	    * @return an array list of the object which are in the field of
	              view of the agent
	 */
	public ArrayList<Object> find_object(Agent a, double radius);
	/**
	 * @param context :the context where the current projection is
	 * @param nb_agent : the number of agents to add in the projection
	 */

	public void add_agents(Context context,int nb_agent);
}
```

Création et ajout d'un agent

1. Pour créer un agent « basique » il faut le déclarer dans le fichier model.score (clic droit sur le contexte => Create Member => Agent).
2. Puis il faut créer une classe Java correspondante. Le nom du label de l'agent doit correspondre au nom de la classe créée, pour l'instant appelons le « Agent_Mobile ».
3. Une fois que le type Agent_Mobile existe, on va ajouter des instances de cette classe dans la simulation.

La création d'un agent s'effectue simplement par l'appel au constructeur de la classe correspondante, mais pour l'intégrer dans la simulation il faut l'ajouter dans le contexte puis dans une projection. On va donc faire un appel depuis la classe ContextCreator à la méthode « add_agent » du ground_manager. (il est logique d'ajouter les premiers agents de la simulation lors de la définition du contexte, mais on peut aussi bien créer une classe indépendante qui va créer des générations d'agents à des moments donnés).

Dans cet appel on envoie en paramètre le contexte dans lequel on va ajouter l'agent, puis le nombre d'agents à ajouter.

```
public class ContextCreator implements ContextBuilder
{
	//on peut définir le nombre d'agent en variable de classe pour
	//la modifier rapidement.

	public final int number_of_agents = 20;

	@Override
	public Context build(Context context)
	{
	//on crée un ground_manager de type gis
	GIS_Manager gm = new GIS_Manager();
	//on charge le terrain
	gm.create_ground(context,"Geography","data_sig/shapefile.shp");
	//on  ajoute les agents

	gm.add_agents(context, number_of_agents);

	return context
	}
}
```

Il faut ensuite implémenter la méthode « add_agent » dans la classe implémentant l'interface Ground_Manager que l'on a choisi (ici SIG_Manager). Puis il faut configurer un affichage en fonction de la représentation choisie.

- Lors des tests, la configuration de l'affichage peut faire perdre du temps. Mais il est possible de sauvegarder la configuration faite dans l'interface en cliquant sur le bouton « save model ».
- Si vous voulez disposer de plusieurs types d'affichage prêts, vous pouvez récupérer le contenu du fichier qui contient la configuration de l'affichage. Il s'agit d'un fichier XML situé à coté du model. Score. Son nom doit ressembler à « repast.simphony.action.display_1.xml ». Ce fichier est écrasé à chaque fois que vous enregistrez un affichage, mais vous pouvez copier coller le contenu pour conserver différentes configurations d'affichage. Il suffit ensuite de le remplacer.

A.- Méthode pour le SIG_Manager

Dans le cas d'un terrain de type `geography`, on va créer une géométrie de type point pour représenter chaque agent. On place ensuite ces agents de façon aléatoire mais à l'intérieur du shapefile.

```
@override
public void add_agents(Context context, int nb_agent)
	{
	//on créé un générateur de nombre aléatoire
	Random rand =  new Random();
	//on crée une « usine à géométrie »
	GeometryFactory gf = new GeometryFactory();
	    //on récupère la projection concerné par le GIS (celle  déclarée
	    // dans le //fichier « model.score »
	    Geography projection = (Geogra-
	    phy)context.getProjection("Geography");


	//we get the boundaries from the envelope
```

```
int x_min = (int) envelope.getMinX();
int y_min = (int) envelope.getMinY();
int x_max = (int) envelope.getMaxX();
int y_max = (int) envelope.getMaxY();

//finally we add agents
      for (int i = 0 ; i< nb_agent ; i++)
      {
      Mobile_Agent agent = new Mobile_Agent();
      context.add(agent); //we add one agent in the context
      //we take a couple of random coordinate inside the SIG
      int x = rand.nextInt(x_max-x_min) + x_min;
      int y = rand.nextInt(y_max-y_min) + y_min;
      Coordinate c = new Coordinate(x,y);
      //then we build a geometry where we can put the agent
      com.vividsolutions.jts.geom.Point p = gf.createPoint(c);
      projection.move(agent, p);
    }
}
```

☝ Attention la dernière ligne ne déplace pas l'agent mais lui associe une géométrie. Il faut voir le résultat de cette méthode comme une entité composée d'une géométrie et d'un agent. La géométrie est en fait la représentation de l'agent (ici un point) et c'est elle qui possède des coordonnées (sans géométrie, un agent n'a pas de représentation et sans agent une géométrie n'est pas représentée). L'agent existe en tant qu'acteur et possède des attributs, la géométrie n'est qu'une forme possédant des coordonnées.

Pour voir vos agents apparaître, vous devez configurer leur style dans la fenêtre « Displays » du simulateur. Il faut les placer dans une couche au dessus du terrain et leur donner un style de point. Il est également possible de modifier leur taille, leur couleur et la forme (apparence du point) ou de les rendre variables selon la valeur d'un attribut de l'agent.

Attention, il est nécessaire d'effectuer au moins un changement de style par agent, sinon la représentation reste celle par défaut et toute modification sur un agent se répercute sur les autres. Si vous déclarez vos « Agent_Parcelles » de type polygone puis vos agent mobiles de type point, les agent parcelle redeviennent de type point.

Il faut donc utiliser la fonction d'édition pour que le type soit personnalisé pour chaque agent. Il suffit par exemple de changer la couleur de l'agent.

B.- Méthode pour le Raster Manager

Pour ajouter des agents dans le raster_manager, on va ajouter les agents dans la projection de type espace continu selon des coordonnées aléatoires.

On va également ajouter la référence de l'agent dans la liste d'agent_mobile associée à l'agent parcelle qui correspond aux coordonnées où on a ajouté l'agent.

```
@Override
public void add_agents(Context context, int nb_agent)
{
          // Pour ajouter des agents, on crée un objet random qui
          // va générer des coordonnées aléatoires dans la grille
      Random rand= new Random();
      int x;
      int y;
      double[] new_location = new double[2];
```

```
        Dimensions dim = grille.getDimensions();
        int grid_width = (int) dim.getWidth();
        int grid_height = (int) dim.getHeight();

        for (int i = 0 ; i < nb_agent; i++)
        {
              x = rand.nextInt(grid_width);
              y = rand.nextInt(grid_height);
              new_location[0] =x;
              new_location[1] =y;
              Mobile_Agent a = new  Mobile_Agent();
              //on place les agents
              matrice_terrain[x][y].agent_incoming(a);

                     //lors de la création d'un agent, on ajoute celui ci à
                     //la liste des agents contenus dans la grille et on lui
                     //donne des coordonnées de déplacement dans la case de 0.
              context.add(a);
              space.moveTo(a, new_location);
        }
}
```

Rendre les agents dynamiques

A ce stade, les agents apparaissent dans le simulateur, mais ils ne font rien. Pour apporter un peu de vie à cette simulation, nous allons permettre aux agents de se déplacer sur leur terrain.

Puisque les agents ne connaissent pas à l'avance le support sur lequel ils vont être placés, il faut leur envoyer une instance du Ground_Manager qui va se charger de fournir les méthodes qu'ils vont utiliser.

Cette référence étant commune pour tous les agents, on va utiliser une méthode statique pour renseigner les attributs communs à toutes les instances de la classe. Cette méthode sera utilisée par le constructeur du contexte.

L'appel à cette méthode doit se faire avant que les agents commencent à interagir avec le terrain. Dans le cas d'attributs statiques on peut très bien les renseigner avant la création des agents. Par la même occasion on pourra configurer les attributs de classe des agents en fonction des paramètres choisis pour la simulation :

Il faut donc modifier la méthode « `build` » de la classe ContextCreator pour faire appel à une méthode « init » permettant aux agents de connaître les références qu'ils vont utiliser lors de la simulation.

```
        @Override
        public Context build(Context context)
        {
        //on crée un ground_manager de type gis
        GIS_Manager gm = new GIS_Manager();
        //on charge le terrain
        gm.create_ground(context,"Geography","data_sig/shapefile.shp");
        //on va initialiser les agents pour leur donner une référence du
        //Ground_Manager à appeler lors de la simulation
        Field_Agent.init(gm);
        Mobile_Agent.init(gm);
        //on  ajoute les agents
        gm.add_agents(context, number_of_agents);
        return context
        }
```

On va ensuite ajouter cette méthode dans les classes de nos agents. Dans le code ci dessous, on initialise les agents de la classe agent dynamique.

```
public static void init( Ground_Manager m)
	{
	//on récupère les paramètres de la simulation
	Parameters p = RunEnvironment.getInstance().getParameters();
	//puis on récupère les valeurs enregistrées dans les différents champs
	vision = (Double)p.getValue("agent_Vision");

	// on peut ainsi récupérer les attributs d'un agent
	speed = (Double)p.getValue("agentspeed");

	//et les paramètres de l'environnement dont il faut tenir compte
	double mn_in_tick =  (Double)p.getValue("minutes_by_tick");
	//dans cette exemple on adapte le déplacement par tic en fonction
	//de la vitesse et du temps représenté par un tic
	speed = (speed /60)*mn_in_tick;

	System.out.println(speed);
	//enfin, on renseigne la variable « ground_manager » de type
	//Ground_Manager qui sera très utile dans les méthodes en relation
	//avec le terrain et l'environnement de l'agent.
	ground_manager = m;
	}
```

Maintenant que la structure de base permettant aux agents de réaliser différentes actions est définie, nous pouvons commencer à leur donner un comportement global. A chaque pas de temps, les agents vont réaliser certaines actions.

- Dans la classe de vos agents mobiles, rajoutez la méthode suivante avec la ligne qui la précède.

```
@ScheduledMethod( start =1 , interval = 1)
public void step()
{
}
```

  - Cette méthode est appelée par *Repast Simphony* à chaque pas de temps et pour chaque agent possédant cette méthode. (on peut préciser le tick à partir duquel cette méthode est appelée et l'intervalle)

Nous allons y rajouter les appels aux méthodes que l'agent doit effectuer à chaque tour, par exemple se déplacer.

Ajoutez dans la méthode `step` l'appel à la méthode « `move()` » que nous allons définir juste en dessous.

```
public void move()
	{
	//La variable dep est une variable propre à l'agent, de type
	//coordinate. Elle représente le vecteur de déplacement de l'agent
	//en un tic
	//si l'agent ne bouge pas, on va lui donner des valeurs de
	//déplacements aléatoires en fonction de sa vitesse.
		if (dep.x == 0.0  && dep.y == 0.0)
		{
			dep.x = rand.nextDouble()*speed*2-speed;
			dep.y = rand.nextDouble()*speed*2-speed;
		}
		//on demande ensuite au ground _manager de le déplacer.
		ground_manager.move_Object(this,dep);
	}
```

Ce code est réutilisable quel que soit le terrain, mais il faut adapter la méthode « `move_objet` » selon le Ground_Manager.

C.- Implémentation de la méthode "Move_Object" dans le GIS_Manager

```
@Override
public void move_Object(Agent agent,Coordinate dep)
{
//on commence par récupérer le contexte dans lequel on travaille
Context context = ContextUtils.getContext(agent);
//on peut ensuite récupérer la projection que l'on utilise
Geography projection = (Geography) context.getProjection("Geography");
//on récupère la géométrie de l'agent présent dans cette projection.
Geometry geom = projection.getGeometry(agent);
//on peut alors récupérer les coordonnées de la géométrie(donc de l'agent)
Coordinate coord = geom.getCoordinate();

/*puisque l'agent ne connait pas son support, c'est au Ground_Manager d'ef-
fectuer les tests pour vérifier que l'agent ne sorte pas. On ne se contente
pas de modifier son emplacement, on va modifier la référence de déplacement
que l'agent à envoyer pour que celui ci ne reste pas bloqué sur un bord.*/
if ((coord.x < envelope.getMinX() && dep.x < 0) ||(coord.x > enve-
lope.getMaxX() && dep.x >0))
          {
            dep.x = - dep.x;
      }
if ((coord.y < envelope.getMinY() && dep.y < 0) || (coord.y > enve-
lope.getMaxY() && dep.y > 0 ))
      {
            dep.y = - dep.y;
      }
/*une fois que les vérifications ont été faites,on modifie les coordonnées
de la géométrie, puis on doit ré associer l'agent avec cette géométrie mo-
difiée*/
            coord.x += dep.x;
            coord.y += dep.y;
            projection.move(this,geom );
            // TODO Auto-generated method stub
      }
```

D.- Implémentation de la méthode "Move_Objet" dans Raster_Manager

```
@Override
public void move_Object(Agent a, Coordinate c)
          {
            //on récupère les coordonnées de l'agent dans l'espace continu:
      NdPoint p = space.getLocation(a);
            //on récupère la référence sur l'agent parcelle qui correspond à
            // ces coordonnnées
      Field_Agent ap1 = matrice_terrain[(int)p.getX()][(int) p.getY()];

            /*On fait appel à une fonction pour vérifier que le déplacement
            est correct (l'agent ne sort pas de la projection). En cas de be-
            soin, la méthode va modifier la référence de déplacement de
            l'agent pour le renvoyer dans le bon sens: dans tous les cas, la
            fonction renvoi le déplacement adapté au support (et non pas en
            mètre). */
      Coordinate dep =  check_position(p ,c);
      double[] displacement = {dep.x,dep.y};
      //on effectue le déplacement
```

```
        space.moveByDisplacement(a, displacement);
            //après le déplacement, on récupère l'agent parcelle associé à ces
            //nouvelles coordonnées.
        p = space.getLocation(a);
        Field_Agent ap2 = matrice_terrain[(int)p.getX()][(int) p.getY()];
            //si il a changé, il faut déplacer la référence de l'agent de
            //l'ancien agent terrain au nouveau
        if ( ap1 != ap2)
        {
              ap1.agent_leaving(a);
              ap2.agent_incoming(a);
        }
}

public Coordinate check_position(NdPoint p, Coordinate c)
{
        Coordinate dep = new Coordinate();
            //on convertit le déplacement en mètre pour le faire correspondre
            //à un déplacement à l'intérieur du terrain défini par la grille.
        dep.x = (double)c.x / (double)size_of_one_box;
        dep.y = (double)c.y / (double)size_of_one_box;

        //on crée un point pour estimer l'endroit où l'agent va se
            //déplacer
        NdPoint p_s = new NdPoint(p.getX()+dep.x, p.getY()+dep.y );

            //on vérifie que ce point n'est pas en dehors des limites, si
            // c'est le cas, on modifie le déplacement jusqu'à ce que le point
            //d'arrivée soit dans les limites
        if ( p_s.getX() < 0 && c.x < 0 )
        {
              c.x = -c.x;
              while ( p.getX() + dep.x < 0)
              {
                    dep.x += 0.1;
              }
        }
        else if ( p_s.getX()> space.getDimensions().getWidth() && c.x > 0)
        {
              c.x = -c.x;
              while (p.getX()+dep.x>space.getDimensions().getWidth())
              {
                    dep.x -= 0.1;
              }
        }
        if ( p_s.getY() < 0 && c.y < 0 )
        {
              c.y = -c.y;
              while ( p.getY() + dep.y < 0)
              {
                    dep.y += 0.1;
              }
        }
        else if (p_s.getY()>space.getDimensions().getHeight() && c.y > 0)
        {
              c.y = -c.y;
              while(p.getY()+dep.y>space.getDimensions().getHeight())
              {
                    dep.y -= 0.1;
              }
        }
```

```
        return dep;
}
```

Ce déplacement n'est pas très original, mais l'intérêt réside dans le fait que l'agent se déplace selon des valeurs de déplacement qu'il décide lui même,on pourra donc plus tard faire des méthodes pour modifier les valeurs de déplacement pour le faire se diriger vers un point précis.

Donner une perception de leur environnement aux agents

A présent que nos agents sont capables de se déplacer, il peut être utile de leur permettre de savoir où ils sont et sur quel type de terrain ils évoluent. Le comportement d'un agent ne sera pas le même si il arrive dans une zone « forêt » ou une zone « étang ».

Dans la méthode `step`, on va créer une liste d'objet qui va contenir les éléments visibles par l'agent en fonction de son champ de vision.

```
ArrayList<Object>visibles_objects=ground_manager.find_object(this,vision);
```

On va à nouveau utiliser une fonction de l'interface « Ground_Manager », et on va l'implémenter dans la classe SIG_Manager.

E.- Implémentation de la méthode "`find_object`" dans le GIS_Manager

Dans un traitement SIG, pour retrouver les objets dans une certaine zone, on va utiliser une fonction d'intersection. On va ainsi récupérer toutes les géométries (et donc les objets associés à ces géométries) qui sont en intersection avec une géométrie donnée. Pour représenter le champs de vision d'un agent, on va dessiner une géométrie en forme de cercler ayant pour centre la position de l'agent et pour rayon son champs de vision.

```
@Override
public ArrayList<Object> find_object(Agent a, double radius)
{
          //on créé une nouvelle liste d'objet
          ArrayList<Object> list = new ArrayList<Object>();
          Context context = ContextUtils.getContext(a);
         Geography projection=(Geography)context.getProjection("Geography");
         //si le rayon demandé est de 0 , on va utiliser la géométrie de
         l'agent (c'est à dire un point )
          if (radius == 0.0)
      {
      Geometry agent_geom = projection.getGeometry(a);
      for ( Object o : projection.getAllObjects())
            {
            if ( projection.getGeometry(o) != null
      &&projection.getGeometry(o).intersects(agent_geom))
                  {
                  list.add(o);
                  }
            }
      }
          else /*sinon, on va faire appel à une fonction pour dessiner un
          cercle , puis on va récupérer les objets dont la géométrie est en
          intersection avec ce cercle.*/
      {
      Coordinate c = a.getcoord();
      Geometry g = createCircle(c.x,c.y,radius);
```

```
        for ( Object o : projection.getAllObjects())
        {
        if ( projection.getGeometry(o) != null &&
  projection.getGeometry(o).intersects(g))
              {
              list.add(o);
              }
        }
  }
        return list;
  }
```

Création d'une fonction permettant de créer une forme géométrique ronde représentant le champ de vision d'un agent

```
public static Geometry createCircle(double x, double y, final double RA-
DIUS) {
     final int SIDES = 32;
     GeometryFactory factory = new GeometryFactory();
     //on crée un tableau de 32 points
     Coordinate coords[] = new Coordinate[SIDES+1];
     //pour chaque point on va associer des coordonnées du cercle de centre
     //(x,y)
     //on renseigne les 31 premiers points du cercle en suivant un ordra
     // d'angle croissant pour que le tracé soit régulier
     for( int i = 0; i < SIDES; i++)
     {
           double angle = ((double) i / (double) SIDES) * Math.PI * 2.0;
           double dx = Math.cos( angle ) * RADIUS;//décalage x par rapport
                                                  au centre
           double dy = Math.sin( angle ) * RADIUS;//décalage y par rapport
                                                  au centre
      coords[i] = new Coordinate( (double) x + dx, (double) y + dy );
     }
     // le dernier point doit être identique au premier pour compléter le
     polygone
      coords[SIDES] = coords[0];
      //une fois les points crées, on les relie dans un objet LinearRing
      LinearRing ring = factory.createLinearRing( coords );
      //objet que l'ont transforme ensuite en polygone
      Polygon polygon = factory.createPolygon( ring, null );
      //pour que ce polygone soit accépté parmis les agents_parcelles,
      //on le transforme en multi_polygone
      Polygon[] tab_p = {polygon};
      Geometry mp = factory.createMultiPolygon(tab_p);
      //puis on renvoi le cercle crée en tant que géometry
      return mp;
      }
```

> ☝ Attention: Dans ce cas précis, lorsque l'on cherche à récupérer la géométrie en tant que polygone, *Repast Simphony* provoque une erreur, il faut donc rajouter des étapes pour en faire un multipolygone. En effet, il faut adapter le type de la géométrie pour faire en sorte qu'il corresponde au type de géométrie contenu dans le shapefile.

Lors de l'exécution, la méthode step va donc faire bouger l'agent et lui envoyer tous les éléments qu'il peut voir.

☝ Il faut faire attention à la façon dont le logiciel procède, en effet lors d'un parcours les éléments en intersection ne sont pas testés aléatoirement. Ainsi, si on demande à un agent de se déplacer vers le terrain avec lequel il a la plus grande affinité, en cas de conflits (deux terrains à porté avec la même affinité) il choisira le premier testé. Lors d'une simulation, par

exemple, on peut observer que tous les agents ont tendances à se déplacer dans une même direction. Ce problème peut venir ou d'un mauvais choix dans la partie qui donnée des vecteurs de déplacement ou d'un problème de priorité dans les choix de destination.

F.- Implémentation de la méthode "`find_object`" dans le raster_Manager :

```
@Override
public ArrayList<Object> find_object(Agent a, double radius)
{
	ArrayList<Object> liste = new ArrayList<Object>();
	int vision = (int) (radius / size_of_one_box);
	NdPoint emp = space.getLocation(a);
	//on récupère la case qui contient l'agent
	GridPoint gp = new GridPoint((int)emp.getX(),(int)emp.getY());
	//on transforme la vision de l'agent en mètre en vision en case
	    //on effectue ensuite un parcours de toutes les cases visibles
	    //et on récupères les objets contenus
	for (int i = gp.getX()- vision; i <= gp.getX() + vision ; i++)
	{
			for (int j=gp.getY()-vision; j <= gp.getY()+vision; j++)
		{
		if ( i>=0 && j>=0 && i<grille.getDimensions().getWidth()
			&& j < grille.getDimensions().getHeight() )
			{
			liste.add(matrice_terrain[i][j]);
			liste.addAll(matrice_terrain[i][j].getliste_agent());
			}
		}
	}
	return liste;
}
```

Puisque l'on ne sait pas dans quel ordre arrivent les objets vus par l'agent, on considère que tous les objets sont vus simultanément, on va donc effectuer des parcours sur la liste pour trouver les éléments qui nous intéressent et faire des choix aléatoires.

On poursuit donc la méthode step:

```
for ( Object o : visibles_objects )
{//si l'objet récupéré est un agent parcelle et qu'il n'est pas celui sur
lequel   l'agent se trouve actuellement
	    if (o instanceof Field_Agent && (Field_Agent)o != agp )
	{
	    //si on n'a pas encore prévue de destination ou si on en trouve
	    une avec une plus grande affinité
	if(dest==null||((((Field_Agent)o).getaffinite()> dest.getaffinite()))
	{
	    //on crée une liste qui va contenir toutes les parcelles de
	    //la plus haute affinité à proximité
	dest_possible = new ArrayList<Field_Agent>();
	dest_possible.add((Field_Agent)o);
	dest = (Field_Agent)o;
	}
	else if ((((Field_Agent)o).getaffinite() == dest.getaffinite()))
	{
	    //si plusieurs parcelles sont à égalité, on les ajoute dans
	    //la liste
```

```
dest_possible.add((Field_Agent)o);
}
      }
}
if (!dest_possible.isEmpty())
      {
//si la liste n'est pas vide, on choisi une destination au hasard
    //parmi celles possibles
dest=dest_possible.get(rand.nextInt(dest_possible.size()));
      Coordinate p = dest.getcoord();
      if (objectif == null)
            {
            objectif = p;
            }
```

A chaque step, on peut alors appeler une méthode qui modifie le vecteur de déplacement de l'agent pour que celui ci se dirige vers son objectif

```
public void move_to_aim()
      {
      Coordinate coord = this.getcoord();
      //if the agent have a goal
      if (objectif != null)
      {
          //we compare his coordinate with his goal to make him go to
          //his objective
            if ( coord.x - objectif.x > 0)
                  {
                  dep.x = - speed ;
                  }
                  else
                  {
                  dep.x = speed;
                  }
            if ( coord.y - objectif.y > 0)
                  {
                  dep.y = - speed ;
                  }
                  else
                  {
                  dep.y = speed;
                  }

                   //we watch if the agent is or not near to his goal
                              if  (Math.abs(coord.x  -  objectif.x)  <=
                              speed && Math.abs(coord.y - objectif.y) <=
                              speed)
                  {
                              //if is travel is achieved, he loose his goal
                              and he can get a new destination
                        objectif = null;
                                    dep.x  =  rand.nextInt(11)/10*speed  -
                                    0.5*speed;
                                    dep.y  =  rand.nextInt(11)/10*speed  -
                                    0.5*speed;
                  }
            }
      }
```

Annexes

Les outils utilisés

A.- Outils de traitements SIG

Qgis est un logiciel de gestion de SIG, il permet d'importer des fichiers de différents formats, (en particulier les shapefiles). Ses fonctions de base permettent ensuite de visualiser les données en superposant les couches et en les coloriant selon la valeur d'un attribut particulier. La gestion des SIG inclut également un certain nombre d'actions possibles pour la manipulation des données (opération sur les tables d'attributs) et permet de modifier le modèle (récupération de zones grâce au des opérations géométriques), et de changer le système de référencement, ou de modifier les coordonnées.

ARCGIS: c'est un logiciel semblable à Qgis à ceci près que Arcgis est une version professionnelle et propose davantage de fonctions

GOOGLE EARTH: bien qu'il soit souvent utilisé pour la cartographie, Google Earth possède les fonctions permettant de lire et de créer des SIG simples.

GRASS : C'est un logiciel de traitement de SIG de conception modulaire, il propose un grand nombre de traitement et l'appel aux différents modules peut se faire en console, ce qui permet de créer des scripts permettant d'automatiser certains traitements répétitifs.
Bien que Grass soit une application indépendante, elle peut être intégré à Qgis en tant que Plugin et dispose alors d'une interface un peu plus accessible. Cependant tous les traitements ne sont pas toujours possibles depuis l'interface graphique et il est parfois nécessaire d'utiliser la console.

POSTGRESQL | POSTGIS : PostgreSQL est un SGBD (système de gestion de base de données), qui permet de créer des tables et d'effectuer des requêtes SQL. A l'aide d'un utilitaire, on peut convertir les données d'un fichier shapefile en une table de données contenant tous les éléments géométriques du shapefile avec les attributs associés. PostGis quant à lui est un greffon à ajouter sur des tables de données Postgres, il permet de faire appel à l'intérieur des requêtes SQL à des fonctions de requêtes spatiales. PostGis permet donc d'effectuer rapidement des modifications sur un fichier shapefile.
Bien que la combinaison de fonctions géométriques avec le langage SQL soit déroutante, il s'avère être un outil très puissant pour la gestion SIG.
Pour pouvoir utiliser ces outils, il faut toutefois installer un serveur Postgres, créer au moins une base de donnée puis configurer les droits des utilisateurs. Il faut ensuite ajouter les fonctions des PostGis à cette base. Pour finir les utilisateurs doivent pouvoir se connecter, (il faut connaître l'hôte du serveur, les identifiants, les mots de passes et les ports...
Une fois la connexion avec la base configurée, Qgis permet d'accéder directement aux tables et au vues crées et les interpréter comme des shapefile (à condition qu'il existe une clé primaire explicite, c'est à dire le GID). Pour finir on peut enregistrer les tables importées dans Qgis en tant que shapefile.

THE GIMP : (*GNU Image Manipulation Program*) il s'agit d'un logiciel libre de traitement d'image. Il permet de nombreuses opérations et propose une large gamme d'outils.
Dans le cadre de la chaîne de traitement, on l'utilise pour ses outils de sélection, ses opérations sur les contrastes et sa capacité à fonctionner en utilisant différentes couches de calques – des équivalents tels que CorelPaint peuvent être aussi utilisés.

B.- Outils utilisés pour la partie programmation

ECLIPSE : Il s'agit d'un environnement de programmation pour le langage java. Il fonctionne comme une plateforme acceptant de très nombreux plugin permettant une grande flexibilité dans la conception et dans la programmation. Il dispose d'un outils de mises à jour capable de reconnaître des sites comme sources de mise à jour possibles. Le grand nombre de plugin disponible fait d'éclipse un environnement de programmation très adaptable.

REPAST SIMPHONY: Ce logiciel est libre et le code source est disponible. Repast Simphony est le successeur de Repast (REcursive Porous Agent Simulation Toolkit).

Le projet a été créé à l'université de Chicago, puis à été repris par des organisations comme « Argonne National Laboratory ». Actuellement Repast est développé par une équipe de volontaires : ROAD : Repast Organization for Architecture and Development).

Plutôt qu'un logiciel prêt à utiliser, Repast s'apparente plus à une bibliothèque Java mettant à disposition une structure de classes et une interface permettant de représenter la simulation que le programmeur a définit grâce aux outils de Repast et d'autres bibliothèques.

Pour créer une simulation, il faut définir tous les éléments (contexte, projection, agent, interactions...) à l'intérieur de classes Java, mais la structure générale est très adaptable et permet de programmer des simulations très différentes en adaptant le simulateur au besoin.

En revanche puisqu'il est en développement permanent et que ses fonctions sont très ciblées, il est difficile de trouver de la documentation et il est parfois nécessaire de connaître certaines classes du simulateur pour pouvoir programmer certains éléments (par exemple l'affichage).

GEOTOOLS : Bibliothèques Java permettant la gestions de données géoréférencées, on l'utilise principalement pour lire les shapefile et pour la gestion SIG.

JTS : Java Topology Suite, librairies java développée par Vivid Solution permettant une gestion des formes géométriques. Utilisé pour les traitement sur les objets en mode de représentation GIS dans Repast.

Liens utiles

- Site de *Repast* Simphony : http://repast.sourceforge.net/index.html
  → Forum utilisateurs : http://sourceforge.net/mailarchive/forum.php?forum_name=repast-interest
- Tutoriaux pour Repast Simphony:http://repast.sourceforge.net/docs/tutorial/SIM/
- Site de JTS (manipulation des objets geometry): http://www.vividsolutions.com/jts/jtshome.htm
- Site de Geotools (geometry, sig, shp...) : http://geotools.codehaus.org/
- Site de Google Earth: http://earth.google.fr/
- Site de Qgis : http://www.qgis.org/
- Site de GRASS: http://grass.itc.it/gdp/index.php
- Site de Postgres: http://www.postgresql.org/
- Site de Postgis : http://postgis.refractions.net/ & http://www.postgis.fr/
- Site de Gimp: http://www.gimp.org/

Installation de Qgis

Qu'est ce que Qgis? Le mot Qgis provient de la réduction de Quantum GIS que l'on pourrait traduire par « un petit morceau de système d'informations géographiques ». Qgis est un logiciel libre développé par l'OSgeo (Open Source Geospatial Foundation). Il permet la gestion de cartes et le traitement de données géoréférencées. Ce logiciel apporte un grand nombre de fonctionnalités et dispose d'un gestionnaire d'extensions permettant d'importer de nouveaux plugins ou d'ajouter les siens. De plus il est multi-plateformes puisqu'il est disponible pour Linux, Mac OSX et Windows.

Utilisation : Dans le cadre de la chaîne de traitement, Qgis va nous servir à manipuler des shapefiles et à géoréférencer des images. Qgis permet une représentation de la plupart des fichiers images (vecteur ou raster), à condition que ceux ci possèdent des coordonnées. En revanche les traitements de Qgis concernent essentiellement les fichiers vecteurs. Pour la création de fichier, Qgis permet de créer des projets Qgis ou d'exporter des shapefiles à partir d'opérations sur d'autres couches vecteurs.

Installation:

- Ce logiciel libre est téléchargeable sur le site du projet: http://www.qgis.org/ en cliquant sur « download » puis sur la ligne « software » dans le menu qui apparaît.
- L'installation dépend ensuite de la plateforme choisie, Dans le cas de Ubuntu (ou Kubuntu), il faut mettre à jour les sources de dépôts[2].
- Une fois en super utilisateur, il faut éditer la liste qui se trouve à l'adresse `/etc/apt/sources.list`
- Pour aller dans le bon répertoire utilisez la commande suivante: `cd /etc/apt/`
  - Vous pouvez vérifier que le répertoire contient bien le fichier que vous cherchez grâce à la commande `ls`. Cette commande permet d'afficher le contenu du répertoire courant.
- Pour éditer le fichier vous pouvez utiliser un logiciel de traitement de texte, dans notre cas nous allons

[2] Il est nécessaire de passer en super utilisateur (root) pour modifier la liste des dépôts. Pour passer root entrez la commande suivante dans la console: `sudo su` puis entrez votre mot de passe. Pour sortir du mode super utilisateur il faut utiliser la commande « `exit` », si vous utilisez cette commande sans être root, la console se ferme. Pour savoir si vous êtes en mode super-utilisateur ou en simple usager, il suffit de regarder la ligne de commande dans le terminal qui précède la zone de saisie. Si vous êtes reconnu en tant que super utilisateur, la fin de votre ligne de commande doit être « # » sinon la plupart du temps c'est le symbole « $ » qui doit apparaître.

utiliser un des logiciels de traitement de texte en console qui sont plus rapides et plus adaptés pour de petites modifications. Il en existe plusieurs, les plus célèbres sont *nano* (l'évolution de pico) et *vim* (évolution de vi)

- entrez : `nano source.list`
  - Cette commande provoque l'ouverture du fichier « `source.list` ». Ce fichier contient les adresses où votre ordinateur ira chercher des mises à jours et des nouveaux logiciels à installer.
  - Les lignes commençant par « # » sont ignorées lorsque l'ordinateur lit le fichier, elle servent à renseigner l'usager sur l'origine et l'utilité des dépôts.
  - Il est recommandé de laisser une ligne de commentaires pour indiquer quels sont les dépôts que vous avez rajouté et à quoi ils servent (cela peut vous servir lorsque vous voudrez éliminer des dépôts devenus inutiles).
- Placez vous à la fin du fichier à l'aide des touches directionnelles.
- Il faut ensuite copier l'adresse du dépôt qui contient la version de Qgis adaptée à votre système d'exploitation et selon le cas à sa version .
  - Pour Ubuntu, Qgis est disponible pour les trois dernières versions: Gutsy, Hardy et Intrepid
  - remarque: Si vous ne connaissez pas votre distribution de Ubuntu vous pouvez taper « `cat /etc/lsb-release` » dans la console , vous obtiendrez votre numéro de version et son nom (grâce aux champs DISTRIB_RELEASE et DISTRIB_CODENAME)
- Une fois la ligne correspondant à votre version ajoutée, il faut sauvegarder les modifications en faisant « ctrl + o » vous pouvez ensuite sortir de nano en faisant « ctrl + x »
- Maintenant que le dépôt a été rajouté, on peut lancer l'installation (il faut être root), entrez la commande : `apt-get install qgis`
- Votre ordinateur va alors parcourir les dépôts et installer le logiciel et tous les modules dont celui ci a besoin pour fonctionner.
- Si tout se passe bien vous pouvez maintenant lancer Qgis depuis la console en tapant « Qgis » ou depuis le menu de lancement des applications.

Compléments sur les formats raster et vecteur

Raster et vecteur sont des modes de représentation d'une image, il est possible de convertir une image d'un format à l'autre mais des données peuvent être perdues.

> Pour faire simple un fichier de raster peut être vu comme une grande matrice de données (ces données peuvent représenter n'importe quoi). Et les vecteurs sont composés d'un ensemble d'éléments distincts.

Ils ont chacun leur forces et leur faiblesses, et on utilise l'un ou l'autre selon l'objectif que l'on cherche à atteindre. Dans les SIG cependant, on préfère généralement travailler avec des vecteurs:

A.- Les rasters

Les raster fonctionnent comme des matrices de données. Souvent on s'en sert d'image en divisant l'espace manière régulière (généralement par pixel). Ce mode de représentation est facile à utiliser puisque la structure est simple d'accès (les pixels sont ordonnés dans une matrice, et donc accessibles par des coordonnées).

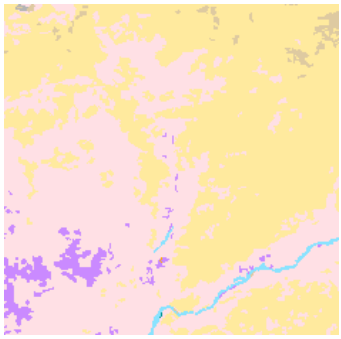


image au format raster

Exemple du codage RGB ou un pixel est représenté par trois composantes: on peut donc utiliser 3 raster de même dimension dont les pixels ont des valeurs entre 0 et 255 pour recomposer une image couleur.

C'est un format assez souple qui correspond bien pour le traitement d'image, puisque toutes les données sont à la même échelle (le pixel) et que les opérations de traitement d'image peuvent être appliquées directement sur la matrice.

Ce format est bien adapté pour le traitement de valeurs continues (comme la température).

- En revanche ce format présente aussi quelques défauts, puisque les objets présents dans le raster ne sont pas différenciés (tout est inclus dans une couche de pixel).
- L'extraction d'élément reste possible en effectuant des tests sur les pixels, mais l'utilisation des vecteurs est souvent plus simple et plus rapide.

B.- Les vecteurs

Ils donnent un mode de représentation géométrique des objets, c'est à dire que tous les éléments sont représentés par des points, des lignes ou des surfaces (polygones).

Ce format présente un grand intérêt pour les SIG puisqu'il permet d'individualiser les objets et donc de définir des attributs. On peut obtenir les coordonnées et les dimensions de chaque élément avec précision. On peut ainsi établir une correspondance entre des objets réels et leur représentation géométriques.

On peut accéder à des informations en effectuant des recherches sur les attributs des différents éléments. Le format enregistre les contours des objets et leurs attributs, il est souvent plus léger (on ne repère pas chaque pixel mais des zones limitées par des points).

Il est possible de comparer les relations entre les coordonnées et les dimensions des différents éléments pour effectuer des requêtes spatiales (intersection, union...).

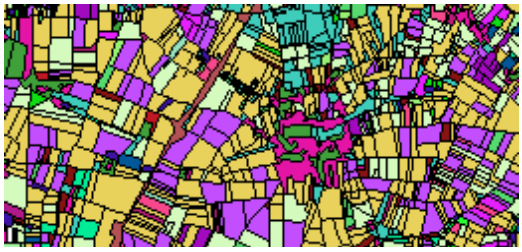


image au format vecteur

- En revanche l'organisation des éléments dépend d'un index, il est donc plus difficile d'effectuer des traitements d'image sur des représentations en vecteur.

Glossaire

A

ABMS : Agent Based Modeling and Simulation

Agent : Dans le cadre de la simulation, un agent est considéré comme une entité autonome définie par ses caractéristiques et ses réactions. Un agent peut disposer d'une perception plus ou moins étendue de son environnement et planifier ses actions pour parvenir à ses buts

Arcview/Arcgis Logiciels de gestion des SIG semblables à Qgis

B

C

CBGP: Centre de Biologie et de Gestion des Populations

Clipping : Le clipping est une méthode permettant d'extraire une partie d'une image.

D

DD (Decimal Degree) : Il s'agit d'une version simplifiée du format DMS. Les coordonnées sont exprimées uniquement en degrés, mais on conserve des décimales. (on travaille donc en base 10)

DMS (Degrés Minutes Secondes) : c'est un format de représentation des coordonnées géographiques, les valeurs sont des fractions d'angle. On exprime ainsi des positionnements par rapport aux parallèles et aux méridiens. C'est un des formats permettant de positionner un point en définissant sa latitude et sa longitude. Une minute d'angle représente 1/60 degrés et une seconde d'angle représente 1/60 minute d'angle.

E

Eclipse : c'est un environnement de programmation permettant de développer en Java. Il accepte de nombreuses extensions qui permettent de faciliter la programmation et d'intégrer des applications comme Repast Simphony.

ESRI : Environmental Systems Research Institute : C'est une firme informatique à l'origine du concept des logiciels SIG. Elle a créé de nombreux logiciels très répandus dont ArcGis. Elle est également à l'origine de certains standards ISO et de formats de données.

F

Feature : c'est une classe contenant les données d'un tuple d'une base de données. On l'utilise pour stoker les polygones des shapefiles avec les données qui leurs sont associées.

G

Geography : Dans Repast Simphony, une géographie est une projection permettant de définir un support virtuel pour des agents évoluant dans un environnement défini par un SIG.

Géomatique: La géomatique regroupe les outils et les méthodes permettant de représenter, et de manipuler des données géographiques.

Geometry : D'après le standard de JTS, une geometry est un enregistrement d'une forme géométrique constituée de un ou plusieurs points.

L'implémentation des classes permettant de gérer et de manipuler les « geometry » a été repris de la bibliothèque JTS. Dans *Repast Simphony* on considère qu'une géométrie est une forme, elle peut

appartenir à un des trois types suivants : Point, ligne ou polygone. Le plus compliqué est le type polygone, qui peut représenter des formes très complexes, et que l'on peut récupérer à partir de fichier de formes (WKT, WKB, shapefile...).

En plus d'une forme, les géométries possèdent des coordonnées. Il est d'ailleurs impossible de déplacer simplement des géométries, il faut changer leurs coordonnées et les recréer.

Géoréférencement: Le géoréférencement est une donnée associée à un élément (agent, carte, image...) permettant de situer cet élément sur le globe terrestre.

Geotools : C'est une librairie open source de Java permettant d'effectuer des traitements sur la géomatique et les SIG.

GML : Geography Markup Language : c'est un langage dérivé du XML permettant de stoker des informations géographiques (par exemple les points définissants un polygone avec les données qui lui sont associées comme son géoréférencement et sa description).

H

I

INRA: Institut National de la Recherche Agronomique, organisme français de recherche en agronomie fondée en 1946

IRD : Institut de Recherche pour le Développement.

J

JAI - Java Advanced Imagery: bibliothèque Java spécialisée dans l'imagerie.

Java : Langage de programmation orientée objet.

Jointure : Technique permettant de fusionner des éléments de deux tables de données en fonction d'attributs ayant des valeurs communes.

JTS : Java Topology Suite : librairie de Java permettant la gestion de données géométriques.

K

KML :Keyhole Markup Language : c'est un langage destiné à l'affichage de données géoréférencées, il est compatible avec la plupart des logiciels SIG et avec les outils Google maps et Google Earth.

L

latitude/longitude : la latitude et la longitude représentent des valeurs angulaires permettant de représenter un point sur la surface de la terre. La latitude varie de 0 degrés a l'équateur jusqu'à 90 degrés aux pôles, elle permet de tracer les parallèles (lignes horizontales parallèles à l'équateur utilisées pour les cartes) . La longitude quant à elle permet de donner une coordonnée en fonction des méridiens (lignes verticales sur une carte joignant les deux pôles) par convention on compte 360 méridiens qui font le tour de la terre et dont l'origine se trouve être le méridien de Greenwich .

M

N

O

OGC : Open Geospatial Consortium: c'est un groupe international qui définit des standards garantissant l'interopérabilité des logiciels de géomatique.

P

Plugin : logiciel qui complète un logiciel hôte pour lui apporter de nouvelles fonctionnalités. La plupart du temps,

ces programmes ne peuvent fonctionner seuls car ils sont uniquement destinés à apporter une fonctionnalité à un ou plusieurs logiciels ;

Projection : dans Repast c'est un système servant de support et de repère pour situer les agents et leur permettre de se déplacer.

Q

QGIS : logiciel de traitement des SIG. Il a été conçu comme une plateforme permettant de faire des liens avec d'autres applications (comme Grass et PostGis).

R

Raster : c'est une matrice de données. On peut s'en servir notamment pour stocker des pixels ordonnancés constituant une image.

Repast :: Recursive Porous Agent : c'est le prédécesseur de Repast Simphony

Repast-Simphony (ne pas confondre avec symphonie) Plugin de Eclipse permettant la simulation multi-agents

RGB ou RVB : Le format RGB est un codage qui permet de définir des images couleurs. De manière générale un fichier raster est composé d'une matrice de données, par exemple des entiers. On utilise ensuite une table de correspondance pour faire correspondre à une valeur un pixel.

Chaque pixel possède une couleur définie par l'association de trois composantes : les trois couleurs primaires : rouge, vert et bleu (RVB) ou Red Green Blue (RGB).

La quantité d'une composante est définie par un entier compris entre 0 et 255 lorsque l'on parle en décimale ou entre 00 et FF lorsque l'on parle en hexadécimal (qui est plus utilisé pour les codes couleurs).

Exemple :

- la couleur ayant pour code RGB FF0000 sera entièrement rouge (la composante rouge a une grande valeur (255) et les deux autres composantes sont a 0.
- Ainsi la couleur 000000 est noire et la couleur FFFFFF est blanche

Remarque: on peut parfois avoir des image en aRGB , les pixels sont alors définis par une composante supplémentaire appelée alpha. Cette composante n'est pas une vraie couleur, mais défini le degrés de transparence (00 signifie totalement transparent et FF une image complètement opaque).

S

Shapefile: (ou fichier de forme) est un format initialement développé par ESRI pour stoker des données issues des SIG.

Un fichier Shapefile (en français «fichier de formes ») est un format de fichier propre aux Systèmes d'informations géographiques. Il permet de stocker des données représentées sous une forme (géométrie) particulière selon des coordonnées géographiques. Ce format a été développé par ESRI pour permettre de regrouper des éléments d'une cartes associés avec des attributs.

De manière classique un fichier shp est généralement constitué d'un groupe de fichiers ayant le même nom mais des extensions différentes. Les trois fichier les plus fréquents sont:

1. le fichier .shp : il contient les formes proprement dites
2. le fichier .shx : il contient un index des formes enregistrés dans le shp
3. le fichier .dbf : il contient les attributs associés à chaque forme.

Mais d'autres fichiers peuvent également faire partie du shapefile, en particulier le fichier .prj

qui sert à définir la projection dans laquelle le shapefile est représenté.

SIG ou **GIS** : Système d'Informations Géographiques. Ce sont des outils permettant de stoker et de traiter des informations spatialement référencées.

Simulation multi-agents : Une simulation multi-agents a pour but d'étudier le comportement d'une population d'agents dont les représentants agissent de façon autonome. Ils peuvent percevoir et interagir entre eux et avec leur environnement. On peut étudier les schémas d'évolution des populations en répétant les simulations et en mesurant l'impact de différents paramètres.

SQL : Structured Query Language : c'est un langage permettant d'interroger ou de manipuler des bases de données relationnelles.

SRID : IDentifiant du Système de Référence: il s'agit d'un nombre servant à identifier un système de coordonnées de référence particulier.

SVN : Subversion: système de traitement de version.

T

TIFF : Tagged Image File Format : c'est un format d'image non compressé extrêmement flexible. Il est très utilisé pour la gestion des rasters car il supporte beaucoup de modifications. De plus il accepte les méta-données. Ce qui permet d'attribuer par exemple un géoréférencement à l'image.

Tuple : ce terme désigne une collection ordonnée d'objet. Dans le cas d'une base de données un tuple représente une ligne d'une table dans une base de données.

U

United States Departement of Agriculture : l' USDA est le département de l'administration fédérale américaine en charge de la politique en matière d'agriculture et d'alimentation. Il fut créé en 1862

UTM (Universal Transverse Mercator : c'est un format permettant de représenter des coordonnées en mètres à n'importe quel emplacement sur le globe.

V

W

WKB : Well Know Binary : équivalent binaire du WKT. Il est interprétable par certaines fonctions SQL

WKT : Well Know Text :format permettant d'enregistrer des formes géométriques dans un format texte (exemple : « POINT(6 10) » représente un point défini par deux coordonnées entières.)

WLD : World : il s'agit d'un format de fichier, il contient uniquement du texte et permet d'associer des coordonnées à une image. Pour géoréférencer « image1.jpg » on peut lui associer un fichier « image1.wld ».

X

XML : (eXtensible Markup Language). La norme XML est disponible à l'adresse www.w3.org/XML/. XML offre une façon pratique et standard de classer des données, afin de faciliter leur lecture, leur accès et leur manipulation. XML utilise une arborescence et une structure de balises identique à HTML

Y

Z