Site Web interactif pour l'exploration de l'histoire démographique des communes françaises de 1789 à aujourd'hui.

Rapport numéro	L6.2
Titre	Site Web interactif pour l'exploration de l'histoire démographique des communes françaises de 1789 à aujourd'hui.
Rédigé par	Christine Plumejeaud (COGIT/IGN), Eric Grosso (COGIT/IGN), Benjamin Parent (COGIT/IGN)
Etat (en cours / final)	Final
Relu par	Marie-Christine Vouloir (EHESS), Sébastien Mustière (COGIT/IGN)
Date	Septembre 2012

Ce document présente la proposition et les développements relatifs à la diffusion des informations produites dans le cadre du projet GéoPeuple, plus particulièrement la diffusion des données historiques et démographiques sur le Web. Il rappelle les objectifs et les besoins auxquels répondent ce développement, présente les choix techniques qui ont été pris, et fait le point sur l'avancement de ce projet et sur la procédure de déploiement de ce site grâce au TGE Adonis.

Sommaire

1. Introduction

L'histoire administrative et démographique des communes est connue et permet de retracer l'évolution de ces entités depuis la révolution française. Un premier site présentant la dimension historique de cette information a été développé par l'EHESS[1]. La structure et la sémantique de cette information est décrite dans [Motte, Séguy, Théré, 2003].

Cependant la dimension géographique de cette information n'est pas prise en compte, et ce site ne permet pas de visualiser les changements des frontières d'une commune, de mesurer son impact sur son profil démographique, ni de l'associer à l'information topographique (comme le relief, l'équipement routier, industriel, les usages du sol, etc.).

Des premiers besoins pour le nouveau site Web ont été formalisés dans le rapport co-écrit par EHESS et la société Digitech [Rat Patron, 2011], notamment en termes de critères de recherche. Une entité peut donc être recherchée par :

- Son toponyme ou l'une de ses variantes,
- L'époque de l'entité : moyen-âge, époque moderne, époque contemporaine,
- Sa nomenclature : administrative, judiciaire, religieuse, etc.,
- Son niveau dans une nomenclature (commune, canton, arrondissement, département, région, etc.)
- Son statut : chef-lieu de canton, sous-préfecture, préfecture,
- Sa période d'existence : entre 1789 et 1801 par exemple, ou à partir de 1801.
- Son code historique unique, conforme aux spécifications [Motte, Pélissier, 2003].

Cette proposition s'appuie sur l'interrogation d'un nouveau modèle de données de l'histoire des communes françaises intégrant un code historique unique par entité. Ce modèle a été révisé en profondeur puis développé dans le projet GeoPeuple (c.f. le rapport [Plumejeaud, 2012]). Elle suggère aussi de dessiner un graphe historique qui permettrait de visualiser plus facilement l'histoire d'une commune et ses degrés de filiation, comme sur la Figure 2 pour l'exemple de Charmesseaux (Figure 1).

Cependant cette proposition ne répond toujours pas à la nécessité de visualiser les limites territoriales hypothétiques des anciennes communes calculées par Hervé LeBras, sur la base d'hypothèses démographiques, ni d'effectuer des recherches spatiales sur les entités.

A partir de là, le projet GeoPeuple, s'inspirant de propositions innovantes en matière de cartographie interactive [Andrienko 2005], de visualisation des redécoupages administratifs [Plumejeaud 2011] et de visualisation des évolutions démographiques [Pison 2011] propose une nouvelle interface de recherche et de visualisation, et a développé un prototype de site Web interactif permettant d'explorer l'histoire et l'évolution de la démographie des communes pour un public non spécialisé. Ce site s'inscrit pleinement dans la tâche 6 du projet « Diffusion des résultats ».

[1] École des Hautes Études en Sciences Sociales, http://cassini.ehess.fr/

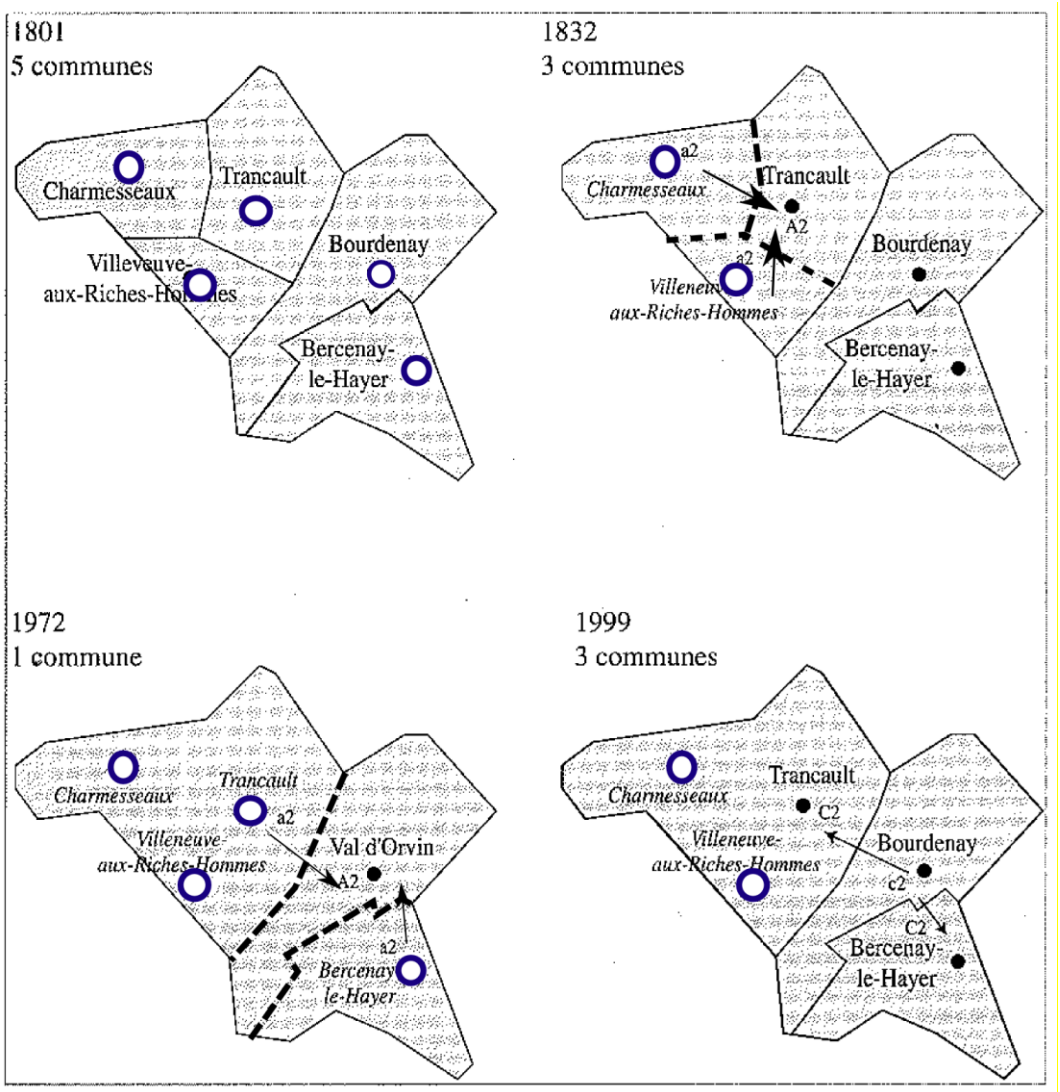

Figure 1. Extrait de [Motte, Ségy, Thére, 2003], p55 : illustration de l'évolution des communes.

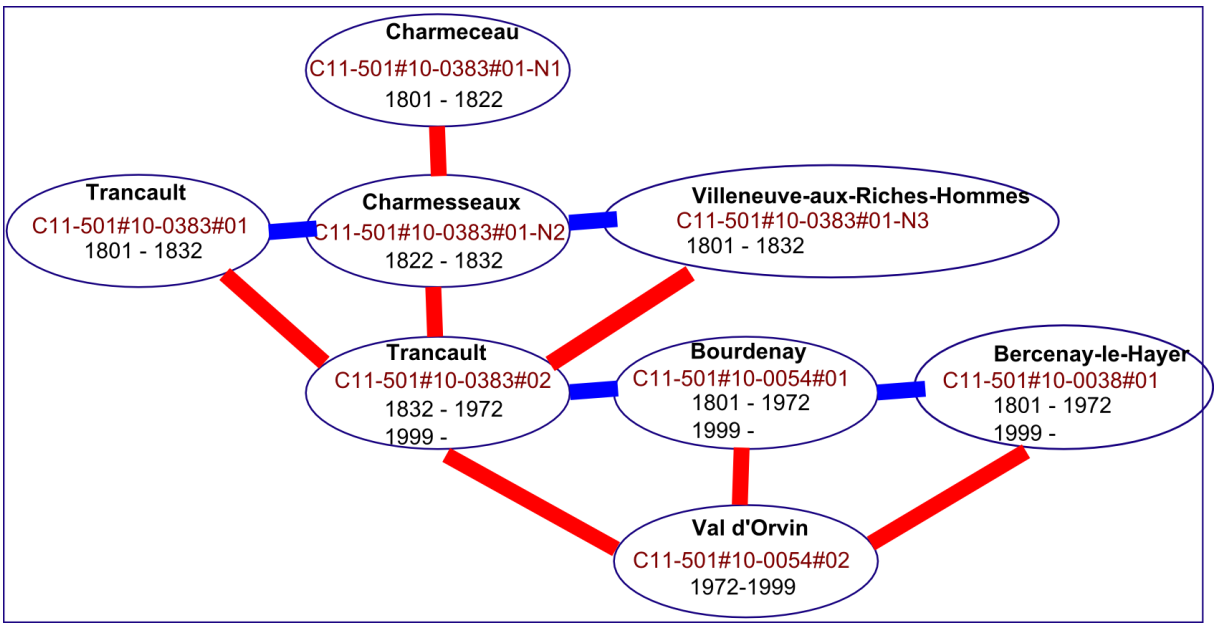

Figure 2. Diagramme systémique correspondant à l'exemple de Charmesseaux.

2. Spécification fonctionnelle

2.1. Besoins exprimés par l'EHESS

Les utilisateurs souhaitent a priori :

1. accéder aux descriptions des entités territoriales (nom, code, chef-lieu, situation spatiale, valeur de population) pour n'importe quelle date, n'importe quelle nomenclature,
2. accéder à la généalogie des communes, cantons, arrondissements, et régions de la sous-division "Departements (1801)", la seule à être presque complètement renseignée,
3. accéder aux données initiales, c'est-à-dire aux images TIFF géoréférencés qui représentent les zones étudiées aux diverses dates.

Le site actuel de Cassini permet par une recherche textuelle (critère toponymique) de retrouver une commune (Figure 3).

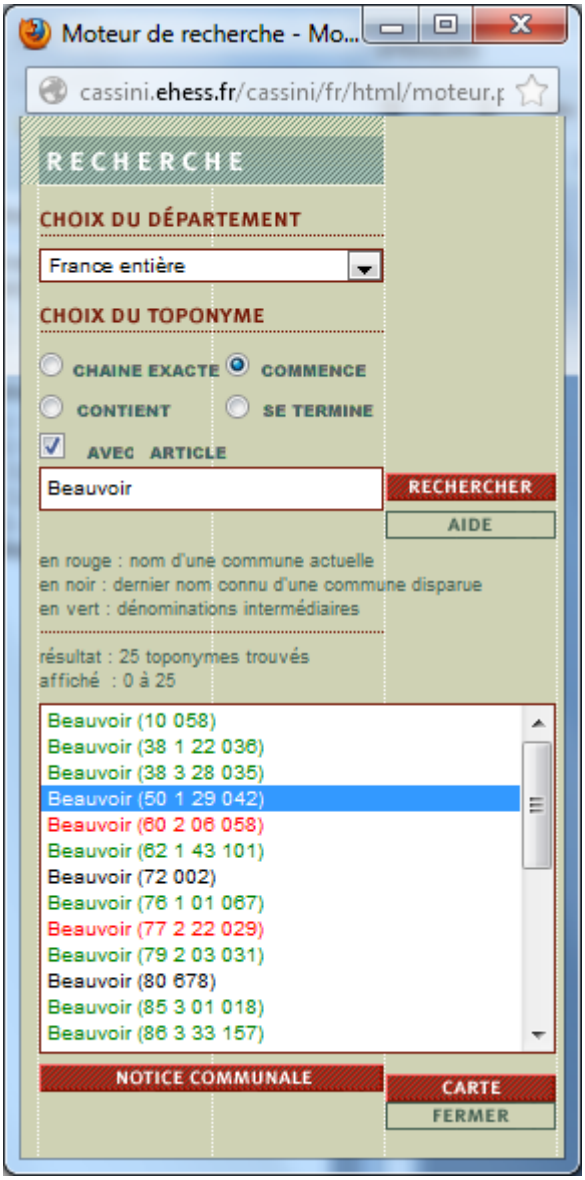

Figure 3. Recherche de "Beauvoir".

Chaque commune est décrite par notice qui donne toutes les informations relatives à une commune (Figure 4), y compris les données démographiques, mais avec un graphique statique sous forme d'image (Figure 5).

Figure 4. Notice de Beauvoir.

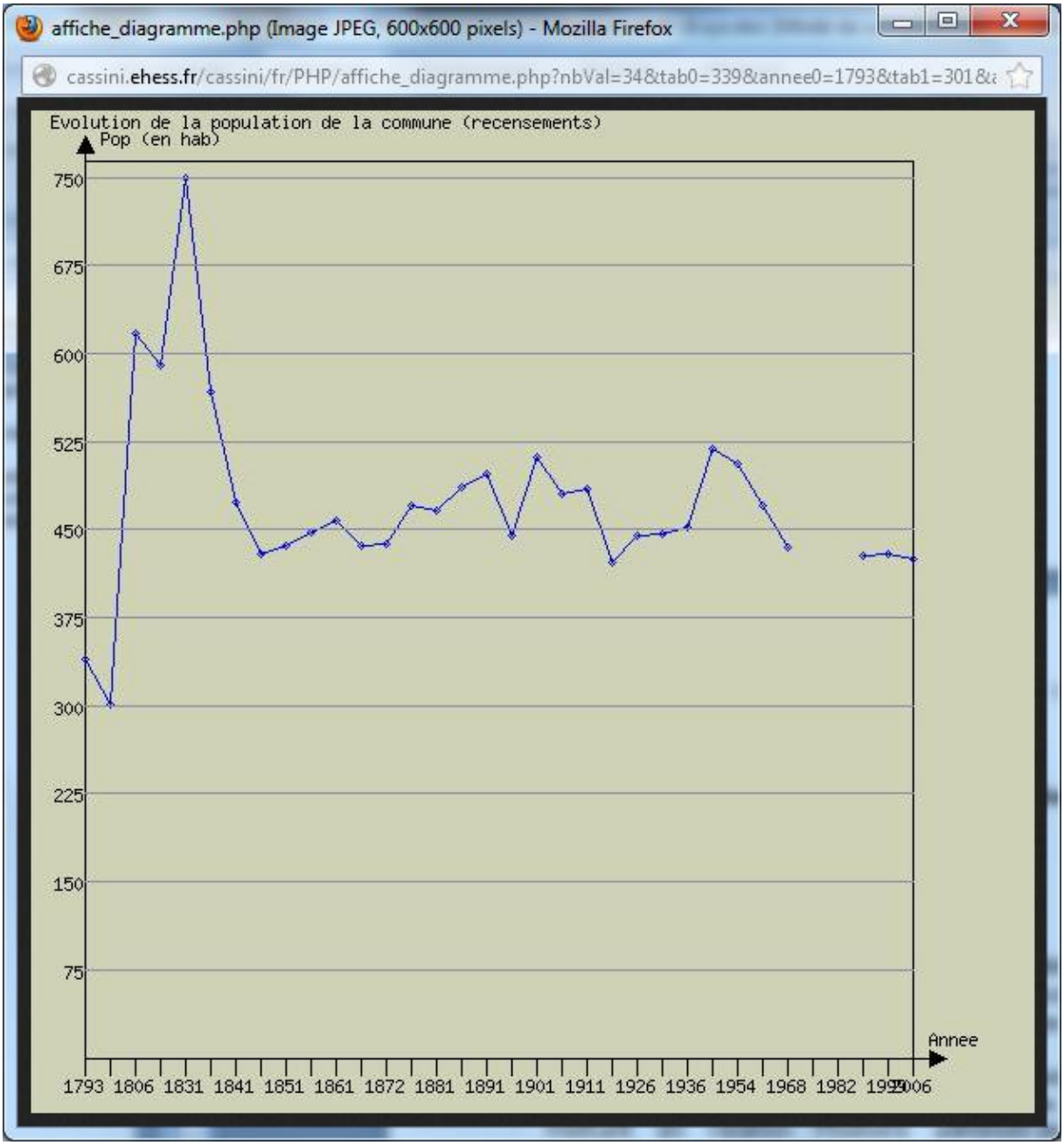

Figure 5. Diagramme d'évolution démographique de Beauvoir.

2.2. Proposition

Christine Plumejeaud a émis une première proposition montrant comment coupler la visualisation des données démographiques et des modifications territoriales (Figure 6).

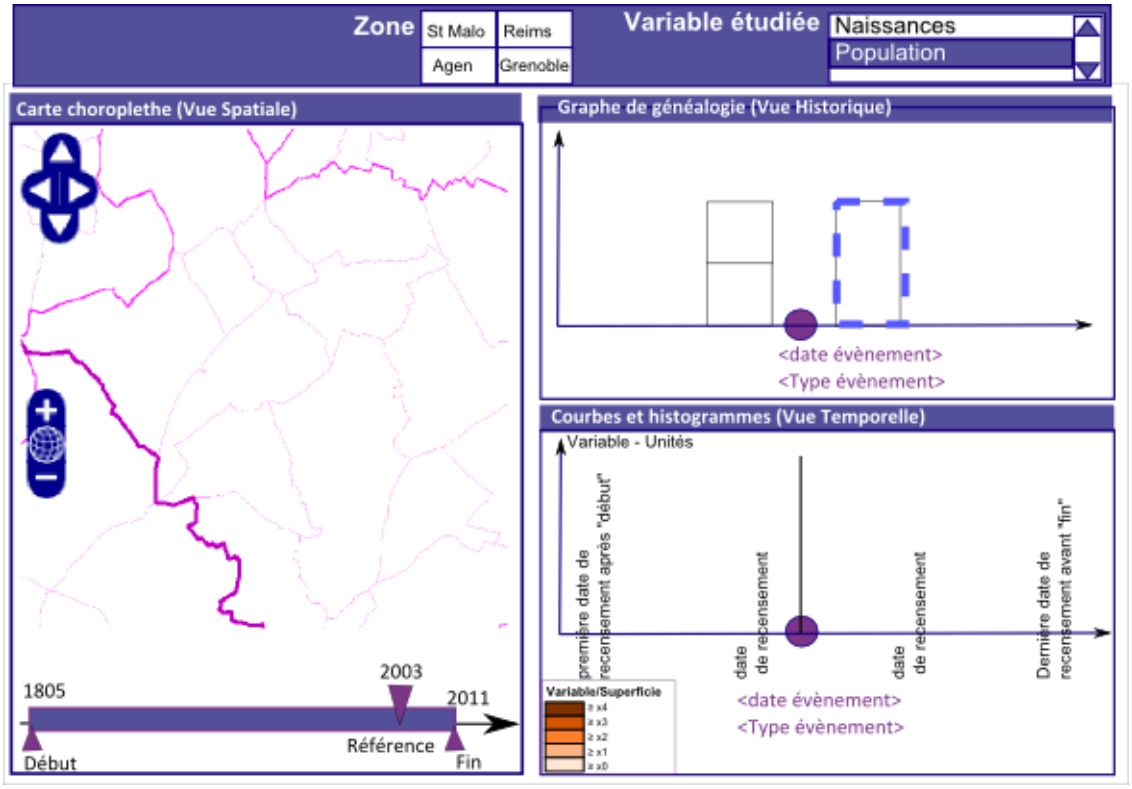

Figure 6. Première maquette.

L'idée principale débattue autour de la première maquette est de synchroniser une vue spatiale (à gauche) montrant les limites des communes et leur évolution (avec un curseur temporel) avec une visualisation sous forme de graphe de l'histoire des communes (en haut à droite), et une visualisation de l'évolution de la démographie et de la densité au cours du temps (en bas à droite), comme dans l'exemple danois de la NUTS entre 1990 et 2003 [Plumejeaud, 2011] (Figure 7).

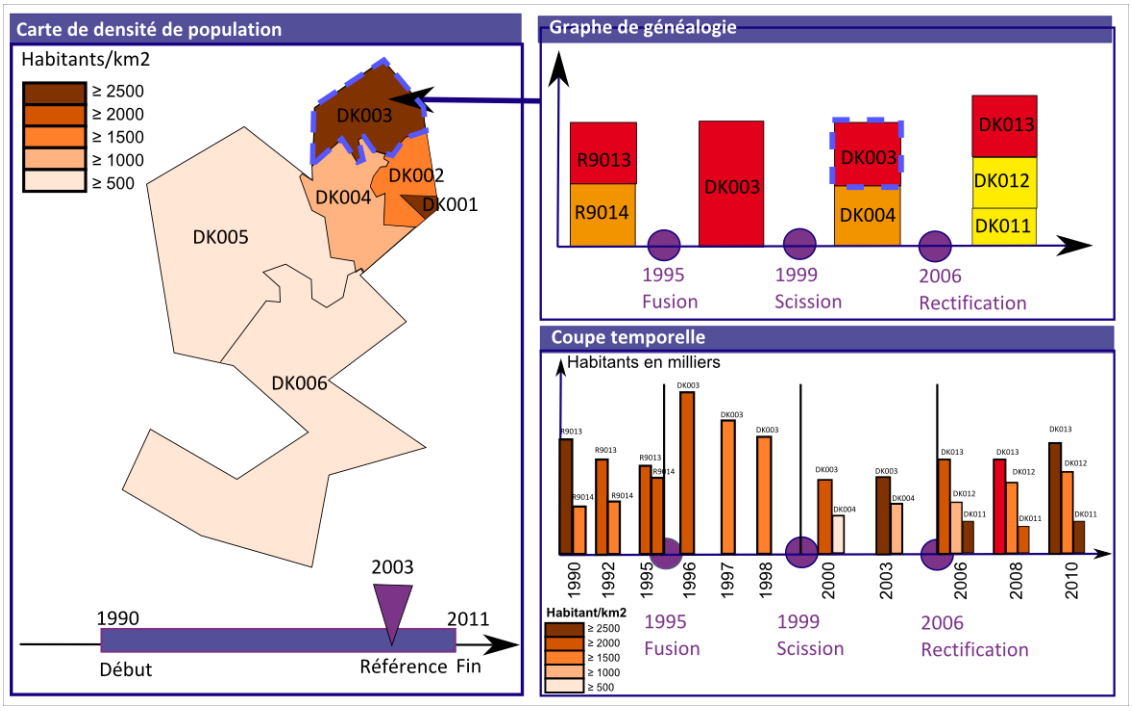

Figure 7. Exemple de modifications sur le Danemark. Source : [Plumejeaud, 2011].

Une réflexion autour de cette première maquette a montré les limites de l'affichage de trois informations simultanément sur le même espace graphique (un écran d'ordinateur standard, 14'') : affichage trop petit et charge d'information trop importante sur le plan cognitif. Par ailleurs, la nécessité d'afficher encore les courbes d'évolution démographiques, mais avec toutes les fonctionnalités qu'offre l'interactivité (zoom, infobulles, etc.) [Antoni, Klein et Moisy, 2004], a ressurgi.

Sur cette base, une seconde proposition de site Web a ensuite été faite par Christine Plumejeaud et Eric Grosso, où il a été décidé de plutôt proposer 4 onglets de lecture d'une même information, chacun apportant un éclairage différent sur la situation administrative et démographique des communes dans l'histoire (Figure 8).

- Le premier onglet est dédié à la visualisation spatiale des limites de la commune (anciennes ou actuelles) et à l'information topographique ;
- Le second onglet est dédié à la compréhension schématique des processus de redécoupages dans lesquels une commune a été impliquée, sous la forme du graphe historique (Figure 10), inspiré de la proposition précédente de EHESS et Digitech (Figure 2) ;
- Le troisième onglet serait une visualisation dynamique de l'évolution démographique des communes considérées (celle de la commune recherchée, et celles en lien par les processus de redécoupage), améliorant ainsi le graphique initial (Figure 5) ;
- Le quatrième onglet permettrait de lier évolution territoriale et démographique via un diagramme en barre de l'évolution de la densité de population.

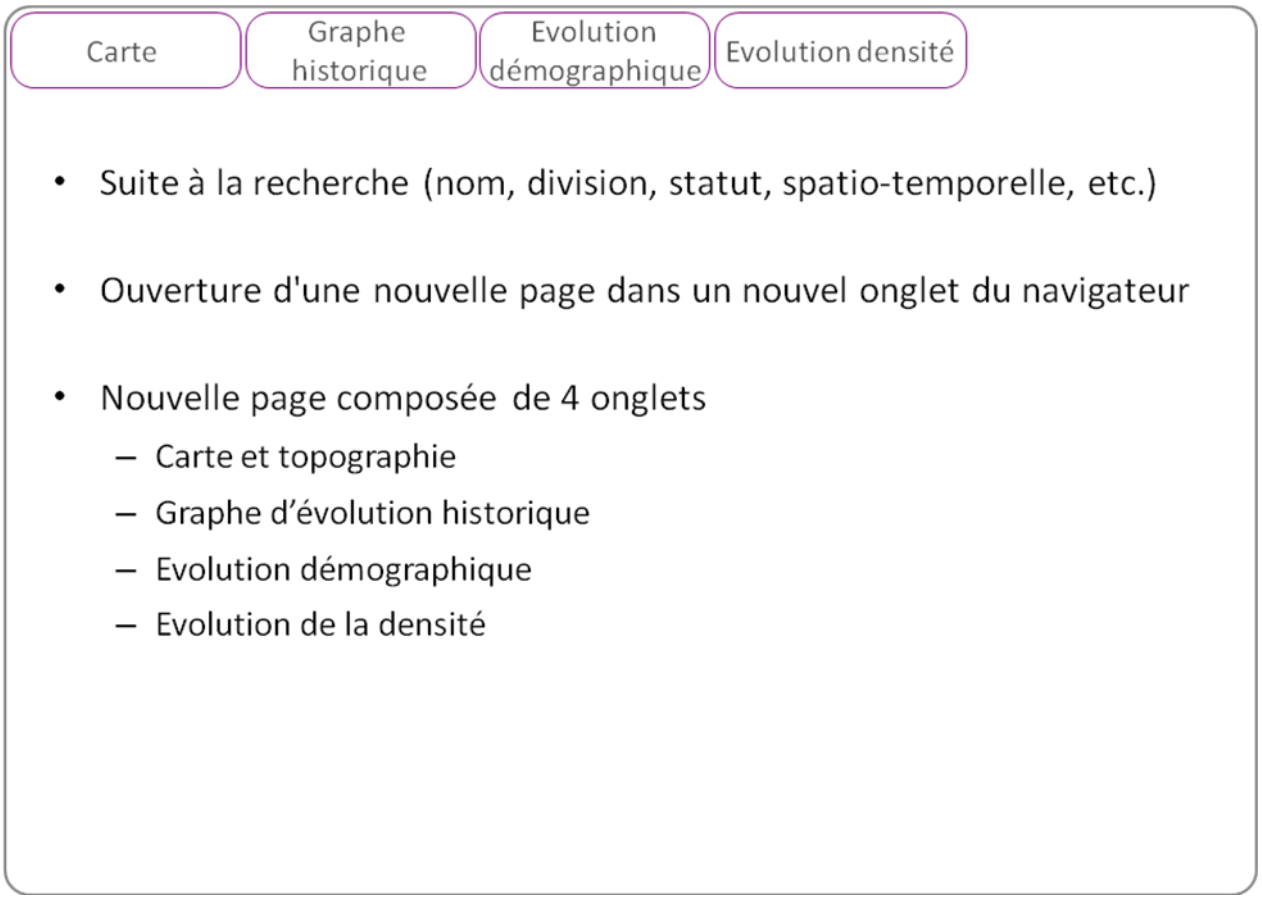

Figure 8. Projet d'onglets synchronisés présentant l'évolution d'une commune et ses affiliées.

Ces onglets feraient suite à une recherche sur une commune sur les critères précédemment cités (Figure 9).

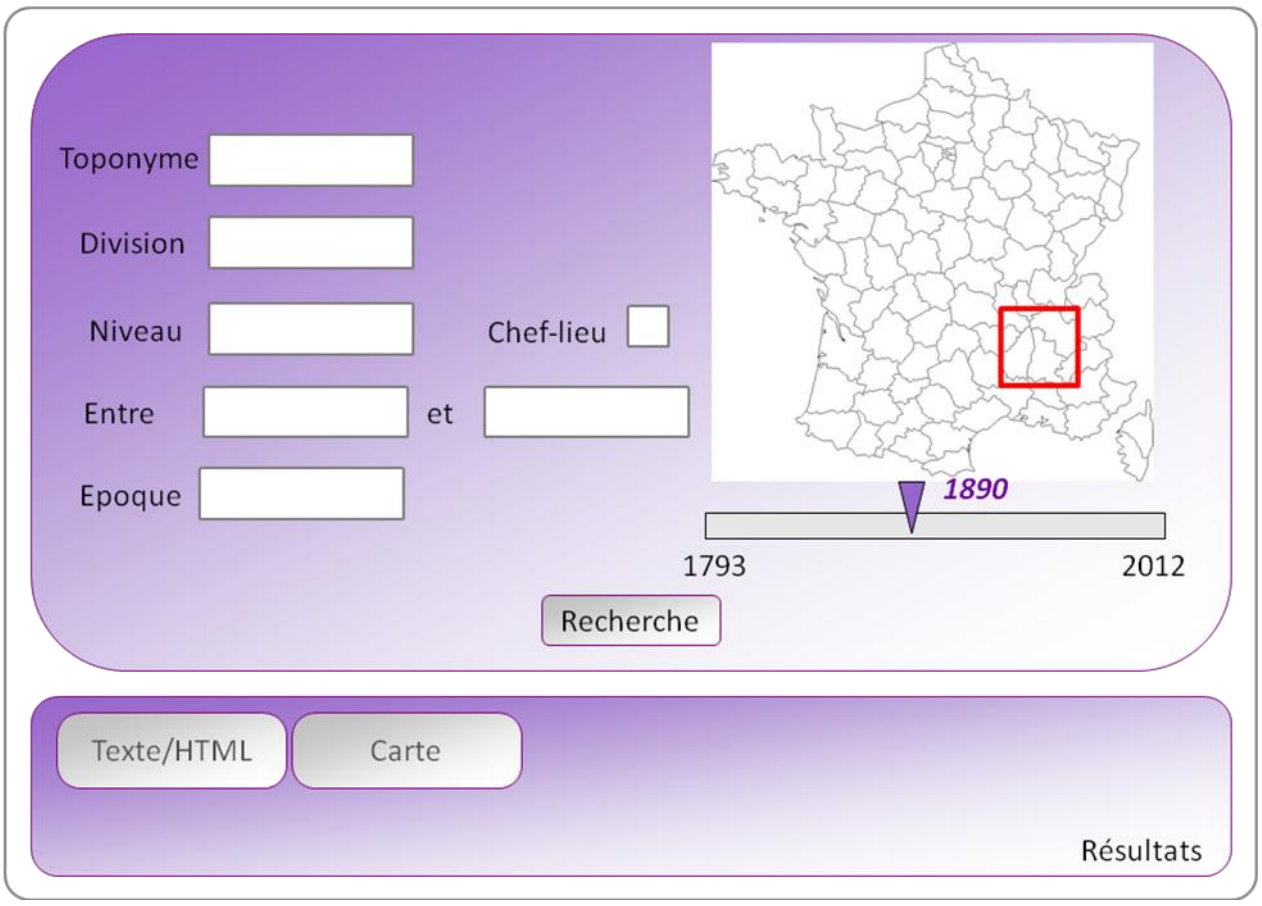

Figure 9. Projet de page de recherche de la commune.

Par cette recherche, toutes les informations liées à la commune, mais aussi aux entités qui sont en lien sont renvoyées. En fait, c'est le ***graphe d'accointance*** qui sera construit (toutes les entités en lien direct par un événement territorial avec l'entité recherchée, mais aussi toutes les entités en lien avec ces entités en lien à quelque degré de filiation que ce soit).

L'idée du graphe historique a été conservée mais revisitée : les entités rétablies doivent réapparaitre sur le graphe, et un axe temporel vertical doit permettre de situer les événements dans le temps. Les bulles doivent contenir l'information nécessaire pour identifier une entité : code historique et toponyme. L'axe temporel vertical permettra de repérer les dates des événements marquants.

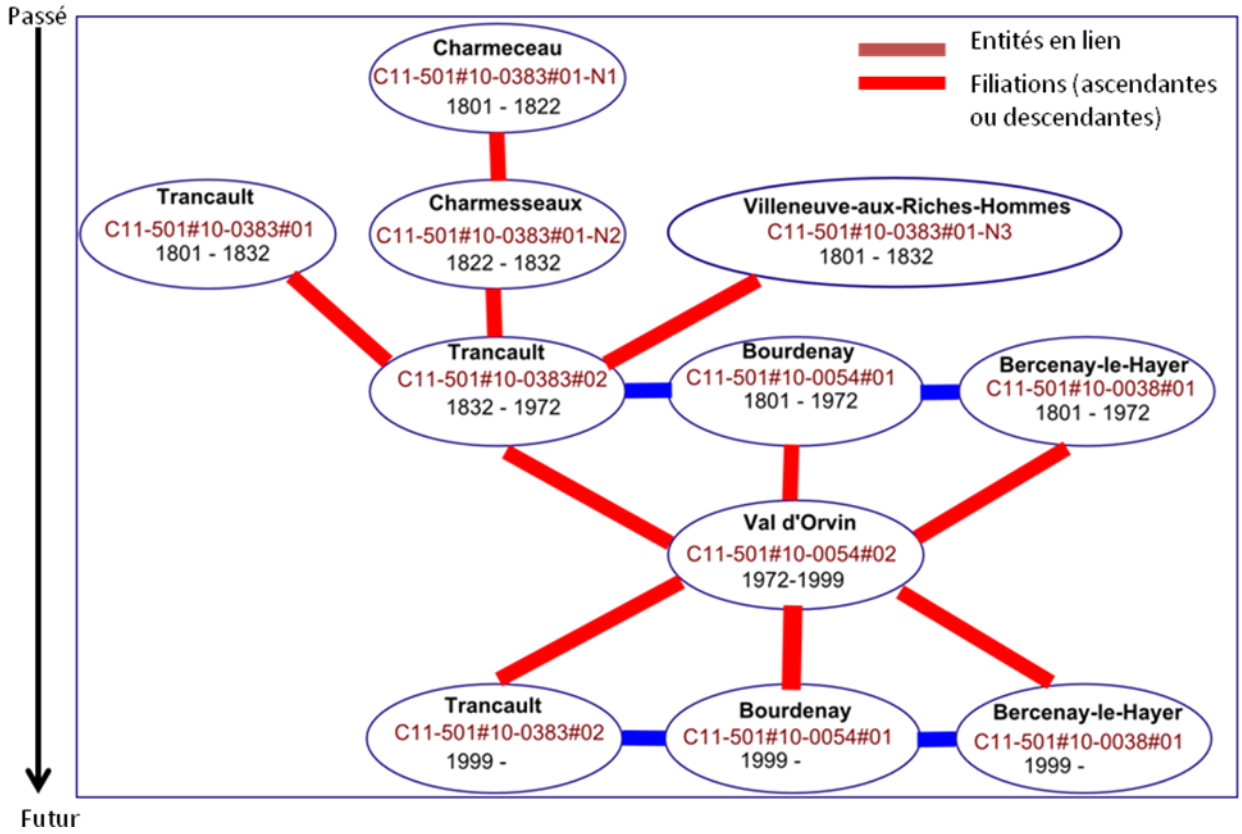

Figure 10. Révision du graphe historique.

Cette seconde proposition a été validée lors d'une réunion avec l'EHESS (Janvier 2012). Lors de cette dernière, d'autres idées ont été évoquées par ailleurs, comme l'introduction d'un graphe ensembliste sur l'onglet du graphe historique (Figure 11).

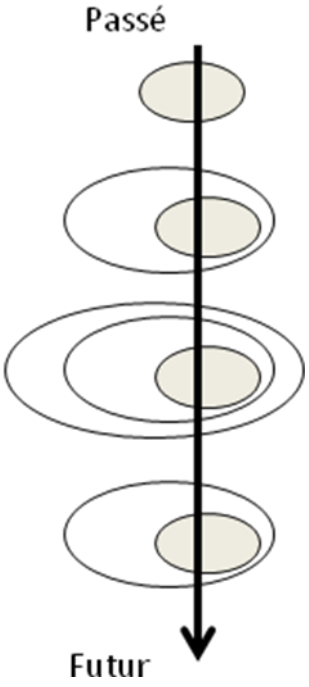

Figure 11. Introduction d'un graphe ensembliste.

Enfin, concernant la spatialisation de l'information, si bien entendu la visualisation des frontières des communes est prioritaire, il est question aussi de visualiser l'information topographique vectorisée par le projet GeoPeuple sur les cartes, en fond, comme des thèmes qui pourraient être choisis par l'utilisateur au besoin (Figure 12).

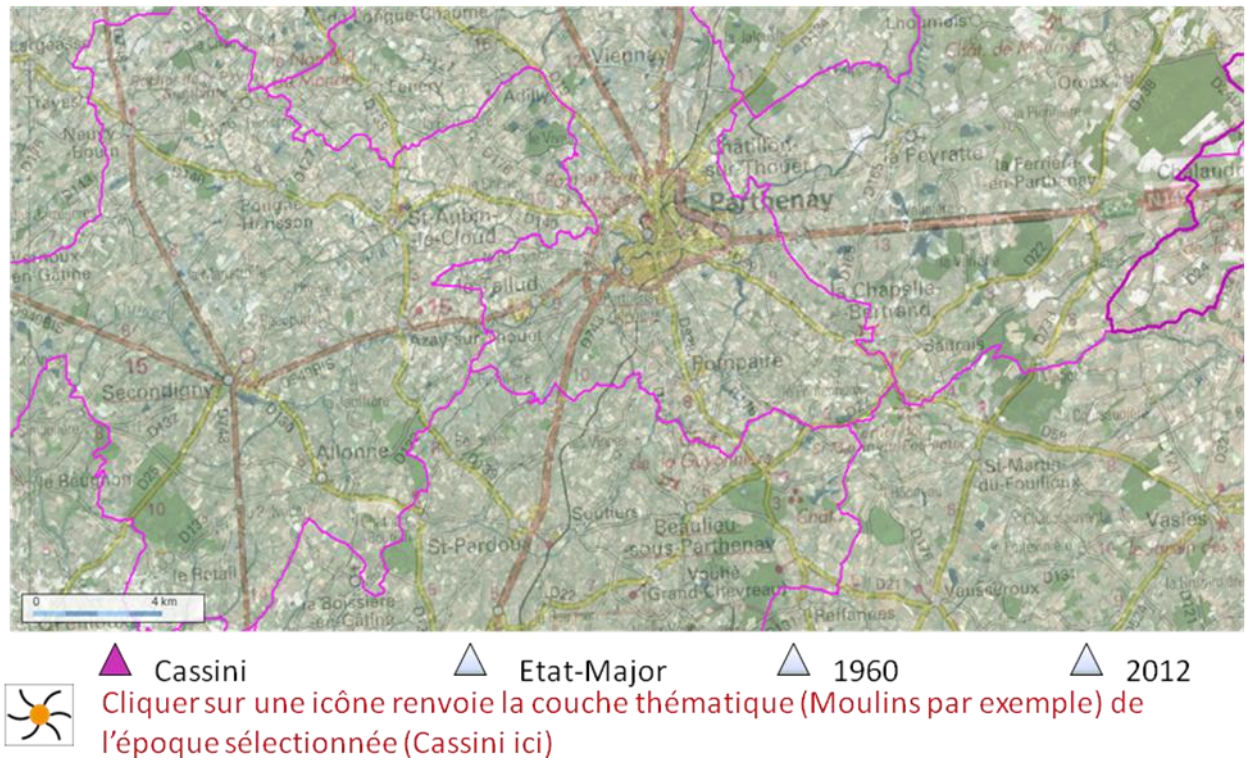

Figure 12. Visualiser des données topographiques sur les communes.

3. Architecture

Nous présentons ici les choix de conception technique qui ont présidé à la réalisation du prototype de site Web de GeoPeuple. Eric Grosso est le principal investigateur concernant le développement du client Web interactif, tandis que Christine Plumejeaud a pris en charge la conception et le développement d'un format d'échange de données optimisé entre la base de données et le client Web.

3.1. Une architecture ouverte pour des données libres

Il a été décidé de fournir une solution modulaire en développant une API Web ***RESTful*** qui permet aux utilisateurs d'interroger facilement la base de données, pour servir plusieurs formats directement sur le Web, et de construire le site GéoPeuple en surcouche. Les formats proposés sont HTML (par défaut), XML, JSON et GeoJSON pour faciliter la manipulation des données mais également KML, la majorité d'entre eux étant soit des standards W3C soit OGC[2]. Cette architecture globale est illustrée Figure 12. Cette méthode a l'avantage de rendre les utilisateurs potentiels de ces données plus libres du format d'affichage des données. En effet, si nous proposons une forme de visualisation évoquée dans les paragraphes précédents sous la forme de 4 onglets dynamiques, il reste possible que les utilisateurs souhaitent construire d'autres modes d'exploration et d'analyse de ces données, ou de connecter ce réservoir de données à d'autres sites Web.

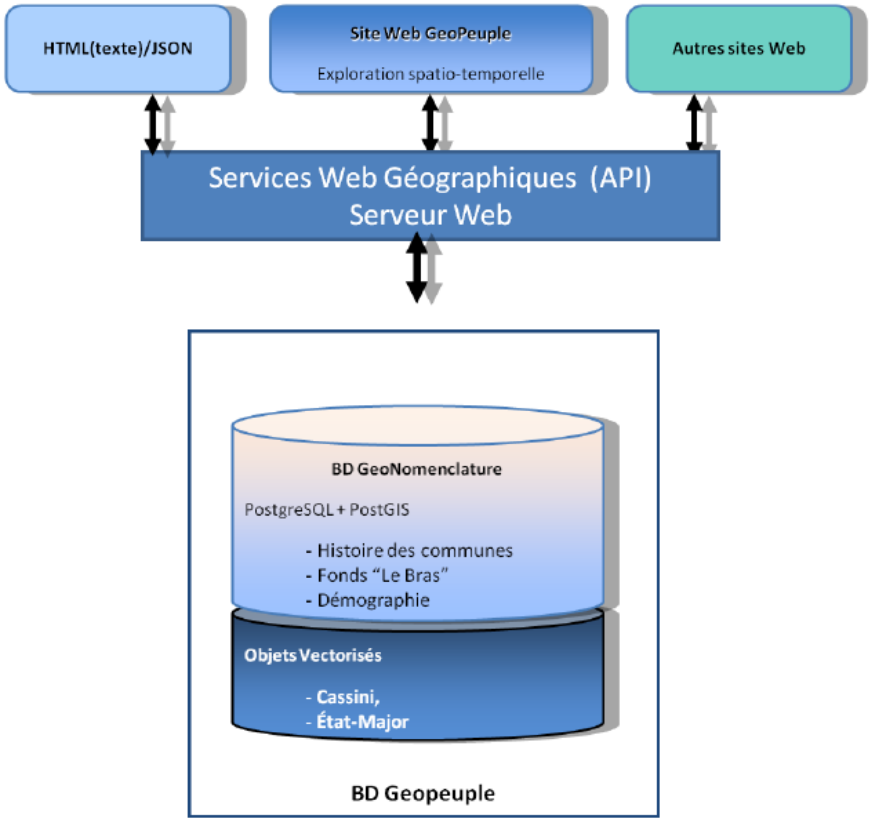

Figure 13. Architecture pour la diffusion libre des données du projet GeoPeuple.

[2] Open Geospatial Consortium, http://www.opengeospatial.org/

Afin d'aider les utilisateurs à requêter l'API Web ou à intégrer facilement l'API dans des pages Web, des extraits de code HTML et de code JavaScript (AJAX) sont fournis. En outre, le code de cette API Web sera diffusé au moyen d'une licence libre à la fin du projet permettant à d'autres projets de réutiliser cette architecture.

Afin de délivrer le code sous licence open source, l'architecture complète, y compris l'API Web RESTful et le site sont basés uniquement sur des composants open source. Côté serveur***, PostgreSQL et PostGIS*** ont été choisis pour stocker et gérer la base de données géo-historique, ***Java et le framework Spring*** pour développer l'API Web RESTful, ***Hibernate*** pour cartographier le modèle de domaine orienté objet de la base de données, ***Tomcat*** comme serveur d'applications Web et ***GeoServer*** (serveur de diffusions de données géographiques et cartographiques) pour partager des données spatiales en ligne. Côté client, le langage ***JavaScript*** est utilisé pour développer l'interface Web, incluant des bibliothèques existants.

3.2. Choix technologiques

3.2.1. SVG et Raphael

Pour représenter le diagramme systémique sur une page Web à partir des données en entrée contenues dans le fichier JSON, nous utilisons le langage JavaScript côté client dans le but de générer un dessin vectoriel au format **SVG**[3]. Cette méthode permet de rendre dynamique l'affichage du diagramme et de le faire évoluer en temps réel et de le rendre réactif aux stimuli de l'utilisateur. Pour rendre ceci facilement concevable, on fait appel à une bibliothèque JavaScript dédiée au dessin vectoriel SVG, libre et performante, nommée **Raphaël**[4]. Cette bibliothèque a été choisie parmi beaucoup d'autres pour ses fonctionnalités étendues et sa prise en main aisée. Le seul bémol réside dans la faiblesse de sa documentation.

3.2.2. HighCharts

Pour pouvoir dessiner des graphiques paramétrés sur un navigateur Web, nous avons choisi d'utiliser la bibliothèque JavaScript nommée **Highcharts**[5]. Cette bibliothèque n'est pas open source mais reste gratuite à l'usage non commercial dans le cadre des produits de la recherche et des universités ce qui rentre dans le cadre de notre projet. Elle permet de définir des plages de données et de nombreuses représentations visuelles pour former des graphiques dynamiques à l'écran. Elle intègre différentes formes de graphiques telles que des histogrammes, des courbes, des aires, des nuages de points, des secteurs, etc. Plusieurs fonctionnalités comme un zoom adaptatif et dynamique gérable facilement par l'utilisateur grâce au curseur, l'affichage de données sur les courbes, la modification en temps réel par l'utilisateur du contenu du graphique et des options d'export avec différents formats standards tels que PNG, JPEG, PDF et SVG nous ont incités à choisir cette bibliothèque. De plus, sa prise en main est très simple et ne demande que peu de temps d'adaptation.

[3] Scalable Vector Graphics, format standard du W3C, http://www.w3.org/TR/SVG/
[4] Raphaël, http://raphaeljs.com
[5] Highcharts, http://www.highcharts.com

3.2.3. JQuery

Nous avons choisi d'utiliser la bibliothèque *JavaScript jQuery 1.7.2*, version la plus récente ne produisant aucun problème de compatibilité, la version 1.8.0 posant des problèmes de compatibilité avec la bibliothèque *Highcharts*. *JavaScript jQuery 1.7.2* présente plusieurs avantages :

- Elle permet de programmer beaucoup plus rapidement une fonctionnalité que de l'écrire à partir de rien. Elle permet donc de rehausser la valeur des projets de conception Web en diminuant le temps consacré ou bien en permettant d'implémenter des fonctionnalités plus poussées,
- Ensuite, elle est conçue pour fonctionner dans tous les navigateurs récents (ainsi qu'Internet Explorer 6 à 8). Chaque navigateur offrant une version différente de JavaScript cela peut rendre le code complexe, illisible et non nécessairement interopérable,
- Aussi, le fait d'utiliser une librairie aussi poussée permet d'éviter d'avoir recours à *Flash*, qui est une technologie propriétaire. *Flash* peut, encore aujourd'hui, être un problème lorsque le site est visionné à partir d'un téléphone.

En sus de ces avantages, *jQuery* sert de fondement à plusieurs autres bibliothèques JavaScript utilisées dans cette application telles que *Apprise*, *Highcharts*, *jQuery UI*. *Apprise*[6] est une petite bibliothèque qui permet d'utiliser des fenêtres de pop-up personnalisables.

3.2.4. API GeoPortail et OpenLayers

Pour représenter des fonds de cartes, des photos, etc. de façon dynamique et interactive à travers un navigateur Web, l'IGN a conçu un service web nommé Géoportail[7].

L'IGN met en œuvre sur l'infrastructure du Géoportail des services s'appuyant sur des protocoles et standards ouverts qui permettent :

- d'afficher les données de référence de l'IGN tuilées avec un haut niveau de performance (services de consultation WMTS),
- d'afficher les données de référence de l'IGN à la demande selon des standards universels (services de consultation WMS),
- d'effectuer une recherche par nom de lieu ou par adresse (service de recherche OpenLS).

Chacun des services du Géoportail se fonde sur ces protocoles et standards pour répondre aux besoins précis des utilisateurs d'information géographique dans différents contextes d'utilisation :

- le service de consultation INSPIRE de l'IGN,
- l'API Géoportail, bibliothèque de programmation JavaScript et Flash spécialement conçue pour le développement de sites internet cartographiques,
- les web services de données IGN pour SIG pour les applications professionnelles et systèmes d'information géographiques,
- les web services de données IGN pour mobiles pour les logiciels destinés au grand public.

L'un des atouts des services du Géoportail est de reposer sur les données géographiques de référence produites par l'IGN :

- les cartes IGN composées des cartes papier de l'IGN scannées,

[6] Apprise, http://thrivingkings.com/read/Apprise-The-attractive-alert-alternative-for-jQuery
[7] Géoportail, http://www.geoportail.gouv.fr/accueil

- les photographies aériennes sont constituées des ortho-photographies (photographies aériennes) de l'IGN et d'images satellites en provenance de partenaires,
- les limites des parcelles cadastrales, propriétés immobilières situées dans chaque commune française.
- Les données issues de la BD TOPO® IGN : limites administratives, routes, hydrographie, bâtiments, aérodromes.

Parmi toutes ces fonctionnalités, nous utilisons le service de consultation, en partie basé sur le standard **WMS** (Web Map Service), qui nous permet d'importer dans notre application les cartes de l'IGN, les bâtiments, le réseau routier, le réseau hydrographique et l'inventaire forestier afin de faire apparaître un fond de carte actuel et de permettre à l'utilisateur de se repérer facilement sur un format de carte auquel il est habitué. Afin de faire apparaître la géographie physique des zones affichées, les orthophotographies sont aussi sélectionnées pour l'application.

En ce qui concerne l'API, nous avons ici retenu la bibliothèque JavaScript plutôt que la version Flash, pour rester dans le même contexte d'utilisation pour tous les onglets et ainsi assurer une communication de données entre ceux-ci, mais également pour garantir une accessibilité au code, Flash étant une solution propriétaire.

L'API Géoportail fournit les fonctionnalités suivantes :

- navigation dans une interface cartographique (zoom, déplacement…),
- affichage des données géographiques (ordonnancement des couches, gestion de l'opacité),
- calcul (mesure de distances, de surfaces…),
- saisie d'objets géométriques (points, polygones…),
- superposition de données métiers aux principaux formats standards (GML, KML, GPX, W*S, GeoJSON, etc.) avec les données IGN.

Cette API est intéressante pour notre application Web car elle permet de manière aisée d'avoir accès à une interface cartographique, à un affichage de données géographiques et de superpositions de couches vectorielles dans de nombreux formats dont le format GeoJSON que nous utilisons ici, et à la saisie d'objets géométriques qui va nous fournir une interactivité entre la carte et l'utilisateur.

L'API Géoportail est basée sur la bibliothèque ***OpenLayers***[8]. Elle surcharge les bibliothèques de cette dernière pour leur apporter de nouvelles fonctionnalités. La bibliothèque *OpenLayers* est principalement une bibliothèque de fonctions *JavaScript* assurant un noyau de fonctionnalités permettant l'écriture d'une application cartographique client légère en *JavaScript*. Une partie de cette bibliothèque permet aussi de gérer l'ergonomie proposée à l'utilisateur, mais ce n'est pas directement son rôle. L'API Géoportail fournit des fonctionnalités supplémentaires à *OpenLayers*, ainsi qu'une interface utilisateur prédéfinie.

Le principe d'*OpenLayers* permet le chargement, la superposition et la gestion de couches (*layers* en anglais) représentant chacune un fond cartographique ou un ensemble de données superposées à un fond cartographique sous forme de dessin vectoriel SVG ou Canvas qui reste plus lent en tant que moteur de rendu.

L'application courante est donc basée sur **l'API Géoportail** étendue (**version 2.0 béta**)[9] et sur *OpenLayers* par la même occasion. Il existe trois versions de l'API Géoportail : minimale, standard et

[8] bibliothèque libre sous licence BSD, http://www.openlayers.org
[9] http://api.ign.fr/article/146/passage-a-l-api-200

étendue. Seule la version étendue contient l'ensemble des fonctionnalités d'*OpenLayers*. Nous avons choisi la version standard, les fonctionnalités proposées par cette dernière étant suffisantes pour répondre aux besoins liés à notre contexte. Ces bibliothèques possèdent d'importantes documentations en ligne ainsi que des communautés actives et expérimentées offrant de nombreux exemples et tutoriaux.

3.2.5. Bibliothèque jQuery UI

La ligne d'évolution temporelle interactive et dynamique de l'onglet « carte » est réalisée avec la bibliothèque JavaScript *jQuery UI*[10]. Celle-ci se base sur la bibliothèque jQuery version 1.3.2 ou plus récente. Elle fournit un ensemble plug-ins interactifs, de composants d'interface graphique, des effets visuels utilisant un style jQuery, offre une interface de programmation événementielle et se focalise sur les standards du Web, l'accessibilité, un style personnalisable, et un design convivial.

Les éléments retenus pour cet onglet sont un curseur défilant (*slider*) qui représente la ligne du temps et un style pour les boutons qui servent à faire défiler le curseur grâce à une animation automatique. L'utilisation de ces composants via *jQuery UI* a l'avantage d'être indépendant de la façon dont les navigateurs implémentent actuellement les standards de HTML 5 qui fournit un composant graphique similaire à un curseur, le type « range » de la balise « input ». En effet Firefox et Internet Explorer n'implémentent pas encore ce composant.

3.3. Flux de données

Le déroulement d'une requête de l'utilisateur du site est le suivant (Figure 14) :

- La page du formulaire de recherche permet à l'utilisateur d'effectuer une recherche sur la base de données. Cette recherche renvoie pour résultat l'identifiant d'une commune.
- A partir de cet identifiant, le script gérant les onglets va demander au serveur un fichier JSON[11] qui contient les données relatives à une commune mais aussi à ses voisines car celles-ci participent avec la première à des événements communs.
- Le gestionnaire des onglets instancie chaque onglet en appelant un script pour chacun et en leur transmettant le contenu du fichier JSON.

Les fichiers JSON sont pré-générés avant d'être stockés sur le serveur de façon à accélérer les temps de chargements. En effet, une autre solution aurait été de calculer un contenu JSON à partir de résultats de la base de données mais cela prendrait un temps de calcul côté serveur non négligeable à chaque requête qu'il vaut mieux éviter pour améliorer les performances.

En effet, sachant que pour chaque commune le serveur renvoie autant de données supplémentaires que la commune possède de voisines, soit 2 (n + 1) requêtes au minimum pour une commune (n étant le nombre de voisines), si les réponses ne sont pas pré calculées, la charge de calcul assignée au serveur risque d'être trop importante. Il faut ajouter à cela le fait que pour toutes les communes qui sont voisines, le résultat de la requête est toujours le même. Cela permet en plus de factoriser les réponses et de réduire le nombre de requête à pré calculer. Étant donné que la base de données est très peu souvent mise à jour, ce pré-calcul des requêtes ne pose pas de problème.

[10] jqueryui.com

[11] JavaScript Object Notation, Notation Objet issue de JavaScript, http://www.json.org/jsonfr.html

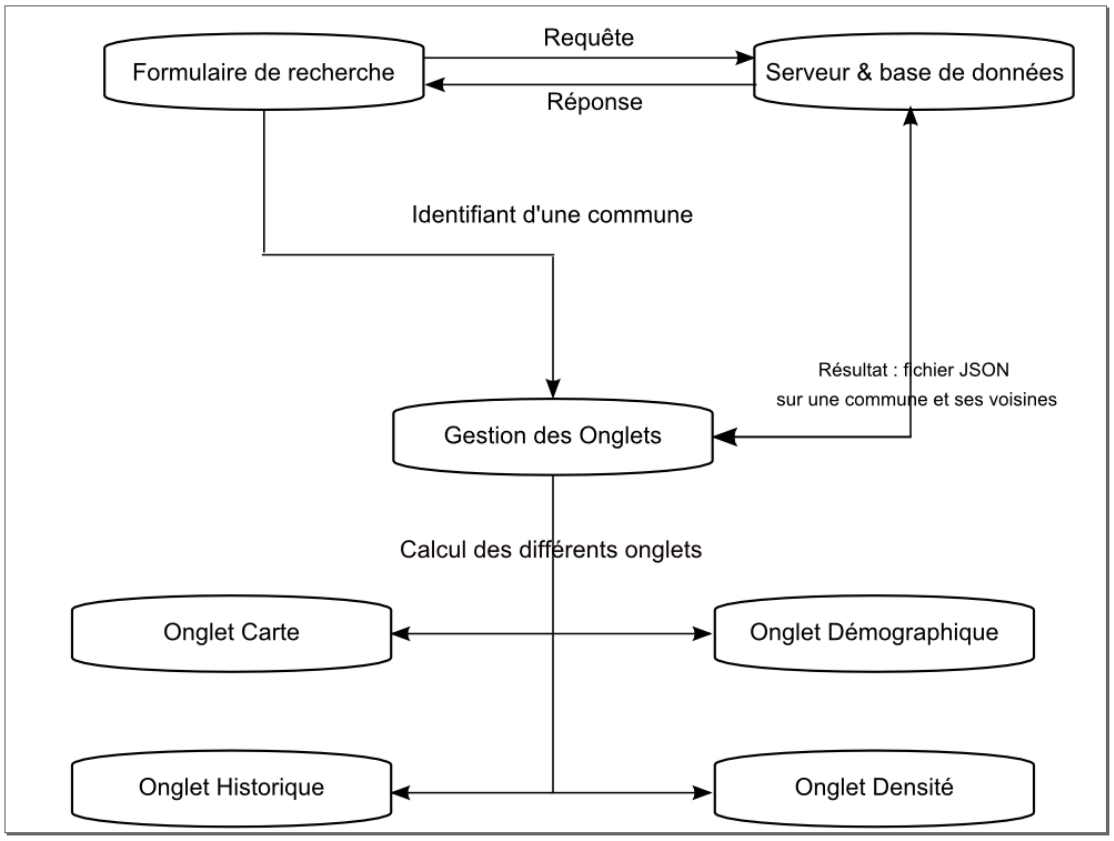

Figure 14. Déroulement d'une requête.

3.3.1. Format des données échangées

La source de données qu'utilise l'algorithme est un fichier au format GeoJSON qui est retourné au client par le serveur suite à la requête sur une commune (Figure 15). Ce fichier décrit l'évolution de la commune choisie en indiquant à chaque date le nouvel état qui est le sien suite à un événement. L'évolution de toutes les autres communes qui participent aux mêmes événements est de même décrite dans ce fichier. C'est donc un ensemble de communes liées entre elles par des événements que ce fichier décrit (le **graphe d'accointance**). Il donne aussi pour chaque commune les chiffres de sa démographie (ou les raisons de l'absence de chiffre). En fin de fichier sont précisées les limites des communes, regroupées par couches, chaque couche calculée correspondant à l'état du territoire à une date, et les couches sont données dans l'ordre chronologiques.

Pour chacune de ces communes, en plus de son évolution, le fichier précise la liste des événements auxquels la commune participe. Chaque événement est caractérisé entre autres éléments, par un *motif* (fusion, transfert, création, suppression, etc.), sa date de production, et la liste des communes qui participent à cet événement ; en indiquant en plus pour chacune de ces communes, si elle existe avant l'événement (*init* vaut vrai) et si elle existe encore après l'événement (*end* vaut vrai), ceci dans le but de faciliter le dessin des entités à l'écran et les liens entre elles.

Une commune possède une série de recensements démographiques qui surviennent à une *date* (précisée par l'année seulement) et qui possèdent un *ordre*. Cet ordre correspond à un numéro de recensement effectué sur la commune, il s'incrémente à chaque nouveau recensement sur une même commune. Il sert à identifier un recensement car la date n'est pas suffisante.

Liste des communes :

-commune 1

Évolution de commune 1 :

-évolution 1 : date, motif (= état de la commune), toponyme

-évolution 2 : date, motif, toponyme

…

Événements de commune 1 :

-événement 1 :

date, motif,

liste des communes participant à l'événement:

-commune 1 : motif, init, end

-commune 2 : motif, init, end

…

-événement 2 :

date, motif,
liste des communes participant à l'événement:

-commune 1 : motif, init, end

…

Démographie de commune 1 :

- recensement 1 : date, population, ordre, estLacunaire, codeLacune, surface

- recensement 2 : date, population, ordre, estLacunaire, codeLacune, surface

…

-commune 2

Évolution de commune 2 :

…

Événements de commune 2 :

…

Démographie de commune 2 :

…

-commune 3

…

Bounding box :

-coordonnées

Liste des couches :

-couche1 : idCouche1, date1,

-composants :

-composant1 :

-propriétés : idCommune1, nom, code insee, canton, arrondissement, département, région, population, idZone

-géométrie : type de géométrie,

-coordonnées :

-coord1 : x1, y1

-coord2 : x2, y2

…

-composant2 : propriétés : idCommune2, …

…

…

-couche2 : idCouche2, date2,

-composants :

-composant1 : propriétés : idCommune1, …

-composant2 : propriétés : idCommune2, …

…

…

…

Figure 15. Structure d'un fichier JSON.

En effet, dans la base de données, on retrouve parfois pour une même commune, deux recensements qui diffèrent par leur date mais pas par leur ordre. Il s'agit par exemple des départements de Nice et de la Savoie, ou de l'Alsace, qui lors de périodes d'annexions à l'étranger n'ont pas été recensés aux mêmes dates que les départements français. Cependant, pour la reconstitution de séries chronologiques complètes, les historiens démographes alignent ces recensements sur les dates les plus proches des recensements français. C'est pourquoi nous considérons que deux recensements possédant le même ordre sont identiques.

Un recensement peut être lacunaire lorsque l'attribut *estLacunaire* est égal à *true*. Dans ce cas la valeur de la population est égale à zéro. Un recensement lacunaire peut avoir plusieurs raisons qui sont indiquées par l'attribut *codeLacune* dont les différents motifs et significations sont répertoriés en annexe.

La *surface* est une donnée numérique exprimée en km², évaluée à partir des géométries (elle dépend donc du niveau de généralisation des géométries). Le niveau de précision choisi nous rapproche de la surface légale des communes (qui comprend les lacs et étangs) qui n'est connue que pour les limites actuelles des communes. A certaines dates, les géométries des communes ne sont pas connues. Dans ce cas, la surface prend la valeur -1. Pour obtenir la densité d'une commune à une date, il suffit de diviser la valeur de l'attribut population représentant le nombre d'habitants par la valeur de l'attribut surface, ce qui donnera une valeur de la densité en habitants par km².

D'après ces données on peut calculer une densité pour chaque commune à chaque date de recensement si tant est que la surface est connue et que le recensement n'est pas lacunaire. Si l'attribut *estLacunaire* vaut *true* alors l'attribut population vaut 0 mais cela ne signifie pas que la densité est égale à 0. Dans ce cas la densité n'est pas calculable.

3.3.2. Gestion des onglets

La gestion des onglets repose sur deux fichiers. Le premier est la page HTML nommée ***onglet.html*** qui contient le visualisateur des onglets, le second est le fichier JavaScript nommé ***scripts_onglets.js*** dont le but est de construire le visualisateur des onglets et d'instancier chaque onglet en communicant à chacun d'entre eux le contenu du fichier JSON reçu du serveur. La page HTML inclut tous les fichiers de styles et de scripts nécessaires au fonctionnement de l'utilisation, puis appelle la fonction ***scriptOnglets*** fournie par *scripts_onglets.js* en lui passant en paramètre l'identifiant de la commune retournée par le formulaire de recherche.

Dans cette application, les onglets sont gérés en parallèle, c'est-à-dire qu'ils sont tous calculés une fois puis les résultats sont enregistrés dans la mémoire du client. Une fois le contenu des onglets chargé, plus aucun rechargement dynamique n'est effectué.

Le fichier *scripts_onglets.js* contient une fonction nommée *scriptOnglets* qui prend en paramètre l'identifiant d'une commune. Cet identifiant est une variable globale accessible depuis tous les scripts de la page notamment les scripts des onglets. Cette fonction est le point d'entrée de l'application. C'est elle qui doit instancier les onglets et exécuter chaque script qui lui est rattaché.

En effet chacun de ces scripts est une fonction dont le principe est le suivant : la fonction *scriptOnglets* appelle la fonction d'un onglet en lui passant le contenu des données JSON en paramètre, celle-ci analyse ces données et en déduit un contenu HTML inséré dans un balise HTML <DIV>, puis un pointeur vers ce contenu est retourné à la fonction *scriptOnglets*. Ce contenu est ensuite inséré dans la page *onglets.html* dans l'un des onglets.

Ce principe est décrit par l'algorithme suivant :

```
Fonction scriptOnglet (identifiant_commune)
        Récupération du contenu JSON
        Fonction un_onglet (json)
                Création du conteneur DIV
                Analyse des données JSON
                Calcul d'un contenu XHTML (HTML / SVG)
                Insertion du contenu dans le conteneur
                Retourne le conteneur
        Fin Fonction
        Insertion du conteneur dans un onglet de la page
Fin Fonction
```

Ainsi ce résultat mis en cache peut représenter un volume de données relativement important en mémoire vive du côté du client. Toutefois cela permet de ne pas surcharger le serveur au détriment des ressources du client qui sont sollicitées pour cette application et de limiter au maximum les échanges sur le réseau afin de prévenir de potentielles fortes latences.

3.3.3. Algorithme de mise à jour des onglets

3.3.3.1. Récupération des données JSON

Dans un premier temps, la fonction doit récupérer le fichier JSON enregistré sur le serveur qui contient les données relatives à la commune requêtée par l'utilisateur (voir format en annexe). Ce fichier a pour nom l'identifiant de la commune que la fonction *scriptOnglets* a reçu en paramètre.

A partir de l'identifiant de la commune, l'algorithme effectue une requête *HTTP* sur le serveur, ici utilisée non pas en vue d'analyser des données au format XML et de manière asynchrone comme avec AJAX mais au format JSON et de manière synchrone. En effet, le reste du script dépend des données transmises et ne peut donc pas s'exécuter sans ces données. Si jamais le fichier précisé dans la requête n'est pas trouvé sur le serveur, un message d'erreur sous forme de pop-up Apprise est affiché à l'écran de l'utilisateur et la fonction arrête son exécution ici (Figure 16).

Figure 16. Message d'erreur affiché à l'écran à la suite d'un fichier non trouvé.

Si le fichier existe, son contenu textuel est évalué avec la fonction JavaScript native *eval* et l'objet JavaScript qui en résulte est stocké en mémoire dans une variable globale à l'application.

3.3.3.2. Construction des onglets

L'étape suivante est la mise en place des différents onglets.

D'un point de vue de l'aspect graphique, chaque onglet n'est qu'un conteneur HTML contenant le titre de l'axe d'étude sur un fond coloré qui lui est propre (Figure 17). Quatre couleurs permettent de distinguer ces quatre axes d'étude, ce sont le rouge, le vert, l'orange et le bleu. Ces couleurs sont stockées dans un tableau global à tous les scripts et font partie de la charte graphique de ces pages. Chaque titre d'onglet est surmonté d'une ligne de la même couleur que l'onglet qui n'est affichée qu'au dessus de l'onglet couramment en visualisation. Sous les titres se trouve un conteneur qui sert à afficher les résultats. Lorsque l'utilisateur clique sur un onglet le contenu change pour répondre à la demande.

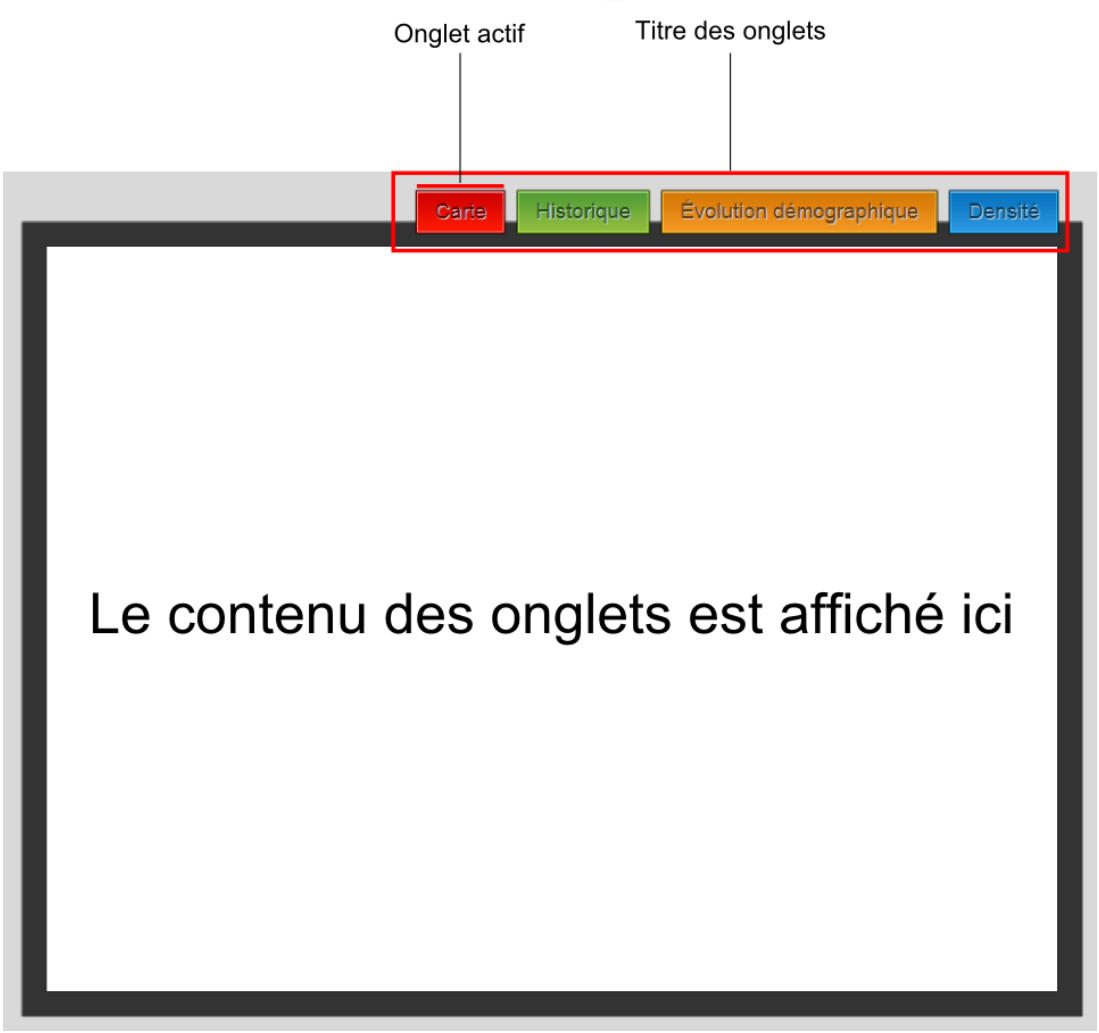

Figure 17. Disposition de la fenêtre des onglets.

Pour insérer le contenu de chaque onglet dans la page, un objet JavaScript est créé. Il possède 4 attributs chacun étant le titre d'un onglet. La valeur de chaque attribut est la fonction qui est appelée pour générer le contenu de l'onglet correspondant. L'algorithme parcourt cet objet qui est utilisé

comme un tableau associatif grâce à la fonction jQuery *each*, et qui insère les titres des onglets avec les noms des attributs mémorisés et leur attribue une couleur dans l'ordre dans lequel ils sont traités grâce au tableau global de couleurs.

C'est à cette occasion qu'est attaché à chaque onglet le contenu qui lui correspond grâce à la méthode jQuery *data* (Figure 18). Cette méthode s'applique à chaque élément de la page contenant le titre d'un onglet et prend en paramètres une chaîne de caractère qui sert de clé à la donnée attachée, et une donnée qui est ici la valeur de retour de chaque fonction d'un onglet, autrement dit le contenu de chaque onglet renvoyé par leur fonction respective qui est exécutée dans la foulée.

```
            //Objet contenant les titres des onglets

            //et les fonctions JavaScript associées

            onglets = {

             'Carte' : carte,

            //la valeur de la variable 'carte' est la fonction qui

            //génère le contenu de l'onglet Carte

            'Historique' : historique,

            'Évolution démographique' : demographie,

            'Densité' : densite

    };

    //Récupère la réponse JSON du serveur

    json = eval('(' + httpRequest.responseText + ')');

    //Couleurs disponibles pour les onglets
    coloris = ['red','green','orange','blue'];

    var numeroOnglet = 0;

    //Crée séquentiellement les onglets et leur assigne une couleur du  tableau

            $.each(onglets, function(nomOnglet, fonction){

    //Création de l'onglet et assignement de sa couleur

            var li = $('<li><a href="#" class="tab ' +    coloris[numeroOnglet++ % coloris.length] + '">' +
            nomOnglet + ' <span class="left" /><span class="right" /></a></li>');

            //Ajoute l'onglet à la page

            $('ul.tabContainer').append(li);

            //Insère le contenu de l'onglet dans le conteneur en appelant la fonction correspondante

            li.find('a').data('contenu', fonction(json));

});
```

Figure 18. Extrait du code de gestion des onglets.

3.3.3.3. Changement du contenu de chaque onglet

L'algorithme doit définir le changement du contenu au clic sur un onglet. Pour ce faire, il utilise la fonction jQuery *click* qui permet de définir un comportement sur l'ensemble des onglets lorsque l'un de ceux-ci est cliqué. Si l'onglet cliqué est déjà l'onglet actif, la fonction arrête son exécution. Sinon la ligne affichée au dessus de l'ancien onglet actif est effacée et redessinée au dessus du nouvel onglet actif de la couleur de ce dernier. Ensuite le conteneur est vidé de tous ses nœuds enfants et le contenu attaché à l'onglet cliqué est récupéré par le nom de sa clé et y est inséré. Finalement une bordure autour du contenu est redessinée de la même couleur que l'onglet.

Lorsque l'on passe d'un onglet à un autre, leur état ne se réinitialise pas car ils ne sont chargés qu'une seule fois au démarrage de la page. Leur contenu évolue seulement avec les interactions de l'utilisateur ce qui permet de conserver l'apparence « en cours » de chaque onglet comme par exemple le zoom sur un graphique ou sur la carte et d'éviter ainsi des temps de chargement.

Si l'utilisateur veut effectuer une nouvelle recherche, il peut cliquer sur le lien « Nouvelle recherche » en haut à gauche de la fenêtre des onglets. Ceci lui ouvre une nouvelle page de recherche dans un nouvel onglet de son navigateur.
Dans le cas où l'utilisateur désire faire plusieurs recherches différentes, chacune de ces recherches génère une nouvelle page de résultat distincte des autres composée de 4 nouveaux onglets.

4. Développement et validation des onglets

Les développements suivants sont le fruit du travail de Benjamin Parent, stagiaire de licence III en informatique de Paris 1, encadré par Christine Plumejeaud et Eric Grosso, entre le 14 mai et le 14 aout 2012.

Les tests et validation ont été effectués sur les dernières versions des navigateurs les plus utilisés en France[12] qui respectent le plus les standards du Web, en particulier HTML 5 : Chrome 20 et 21, Firefox 13 et 14, Internet Explorer 9, Safari 5 et Opera 12. L'objectif de l'application n'est pas d'assurer une compatibilité avec les versions d'Internet Explorer antérieures du fait de leur non respect du standard SVG au profit de l'obsolète VML et de leurs performances dépassées. Les utilisateurs de Windows XP n'ayant pas accès à la version 9 doivent se rabattre sur l'un des autres navigateurs mentionnés précédemment.

4.1. Onglet historique

L'onglet Historique a pour but de représenter l'évolution d'une commune choisie au cours du temps en montrant son interaction avec les autres communes proches avec lesquelles elle est en relation lors d'événements. Ces événements sont constitués de remembrements inter communaux tels que des fusions de communes, des transferts de parcelles, des changements de nom ou d'appartenance (appartenance à des entités comme les régions, les départements, les arrondissements, les cantons, …), des transferts de chef-lieu, des créations de nouvelles communes, des suppressions ou des rétablissements de communes disparues.

Chaque évènement est associé à un identifiant unique et à sa date : les entités impliquées dans un même événement sont indexés par l'identifiant de l'évènement. Une commune est une entité, représentée par un identifiant unique au cours du temps. Chaque entité impliquée dans un évènement est associée à un motif, dont le code (compatible avec la nomenclature COG INSEE[13]) représente ce qui est arrivé à l'entité.

Par exemple, comme le montre la figure 5, lorsqu'une entité e1 absorbe une entité e2 pour former une entité e1 plus grande, e1 est associée au code 32, et e2 au code 31. Lorsqu'une entité e3 cède des parcelles pour la création d'une nouvelle entité e4, e4 est associée au motif 20, et e3 au motif 22. Et si e5 donne des parcelles à e6, le motif de e5 est 60, celui de e6 est 62. La liste des codes exhaustive est fournie en Annexe.

Les entités existant avant l'événement sont appelées « Réactifs », celles existant après l'événement sont appelées « Produits ». Une entité peut être à la fois « réactif » et « produit » sur le même événement.

[12] http://gs.statcounter.com
[13] http://www.insee.fr/fr/methodes/nomenclatures/cog/historique.asp

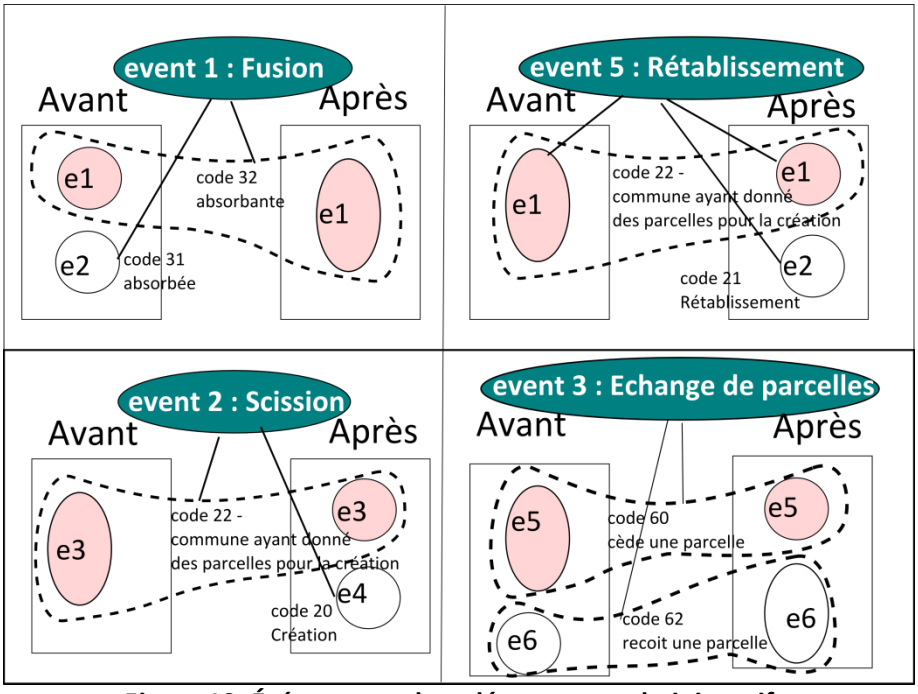

Figure 19. Événements de redécoupages administratifs.

4.1.1. Algorithme de positionnement des éléments du graphe

Dans un premier temps, la zone de dessin est créée et positionnée dans le conteneur, sa hauteur et sa largeur sont égales à la hauteur et à la largeur du conteneur. L'algorithme principal comprend ensuite 4 étapes :

4.1.1.1. Récupération des données

La première étape de l'algorithme consiste à récupérer la liste des entités fournie par le fichier JSON affin de pouvoir ensuite manipuler le contenu du fichier. Une fois ceci fait, on parcourt la liste des communes qui sont traitées une à une afin d'analyser les informations qu'elles apportent.

Le premier traitement est la récupération de la liste des événements auxquels la commune participe. La liste des événements de chaque commune est donc parcourue à son tour et ses événements sont stockés dans un tableau. Cependant au long du parcours des communes les mêmes événements reviennent plusieurs fois, car en effet, les communes enregistrées dans le fichier JSON sont liées entre elles par les mêmes événements. Pour éliminer les doublons sur les événements, lorsqu'un événement a déjà été enregistré dans le tableau, l'algorithme vérifie que la commune qui est en train d'être analysée correspond ou non à la commune sur laquelle porte la requête. Si c'est le cas, c'est l'événement de la commune de la requête qui est gardé et l'autre rejeté, sinon l'événement déjà enregistré reste inchangé.

Lorsque tous les événements distincts ont été récupérés, l'algorithme va récupérer l'évolution de chaque commune décrite dans le fichier. En parcourant le tableau de l'évolution d'une commune, l'algorithme a accès à chaque état de la commune à une date où elle subit un changement, c'est à dire où elle participe à un événement. Le but ici est donc de dessiner une première fois toutes les versions d'entités en prenant soin pour chacune d'elles de conserver dans un tableau qui leur est propre tous les motifs successifs qu'elles ont eu au cours de leur évolution. Ainsi toutes les versions

d'entités affichées à l'écran possèdent tous les motifs successifs de l'entité unique qu'elles représentent.

Ces entités territoriales dessinées n'ont pas encore de position déterminée sur le dessin car les différentes dates qui composent la ligne temporelle n'ont pas encore été récupérées. C'est pourquoi elles sont placées à l'origine du repère aux coordonnées (0,0). On les stocke dans un tableau afin de pouvoir les traiter par la suite.

Chaque état d'une commune est attribué à une date dans son évolution ce qui correspond à un événement qu'elle subit, c'est pourquoi l'algorithme se sert du tableau d'évolution pour récupérer dans le même temps les dates de chaque événement qui apparaîtront sur la ligne temporelle. Ces dates sont elles aussi stockées dans un tableau, sous la forme d'un objet JavaScript (une liste d'attributs) qui comprend l'année, la date d'affichage ainsi que la liste de toutes les entités qui existent à cette date. Cependant les mêmes dates sont fournies plusieurs fois à cause des recoupements des événements entre communes mais l'algorithme ne garde qu'une seule fois chaque date distincte.
Une fois toutes les communes parcourues, on a donc récupéré toutes les dates des événements, les événements eux-mêmes, ainsi que dessiné les versions de chaque entité territoriale. On peut alors traiter ces différentes informations pour gérer la disposition de tous les éléments à afficher et former les liens entre entités qui correspondent aux événements.

4.1.1.2. Positionnement des éléments affichés

Tout d'abord les dates sont triées en ordre chronologique croissant, de même que les événements. Ensuite la ligne temporelle est créée grâce aux dates désormais ordonnées en instanciant la classe *GraphiqueLigneTemporelle*. Si jamais la ligne du temps sort du cadre alors le cadre est automatiquement agrandi. À ce stade de l'exécution de l'algorithme, le dessin comprend une ligne temporelle ainsi que des entités territoriales toutes superposées au niveau de l'origine de la zone de dessin (angle supérieur gauche), celles-ci ne sont donc pas encore correctement placées en face de leur date. La deuxième étape consiste donc à les positionner les unes par rapport aux autres et par rapport à la zone de dessin toute entière.

La classe *GraphiqueLigneTemporelle* possède une méthode ***getDates*** qui retourne un tableau contenant les dates disposées sur la ligne du temps. On peut donc à partir de l'ordonnée de chacune de ces dates, placer en face les entités qui ont un événement. L'algorithme parcourt ces dates et pour chacune d'elle parcourt la liste des versions des communes et vérifie si la date de la commune est égale à la date courante. Si c'est le cas, la position de la commune est alors rectifiée pour s'aligner en face de la date. Comme il y a plusieurs communes à placer en face de chaque date, un espace de taille arbitraire (50 pixels) est inséré entre chaque commune sur le même alignement. Dans ce même temps on calcule la largeur totale en pixels de chaque ligne de communes puis quand on a fini de placer une ligne on utilise la fonction aligner pour centrer horizontalement la ligne courante par rapport à la largeur de la ligne précédente.

Ensuite l'ensemble des entités territoriales dessinées est de nouveau centré mais par rapport à la zone de dessin cette fois. En fait elles sont décalées sur la droite de l'axe du temps afin qu'elles n'apparaissent pas confondues avec celui-ci. De plus la zone de dessin est automatiquement élargie en fonction de ce décalage si certaines entités sortent du cadre, de façon à ce que toutes les entités restent visibles, comme le montre la Figure 20.

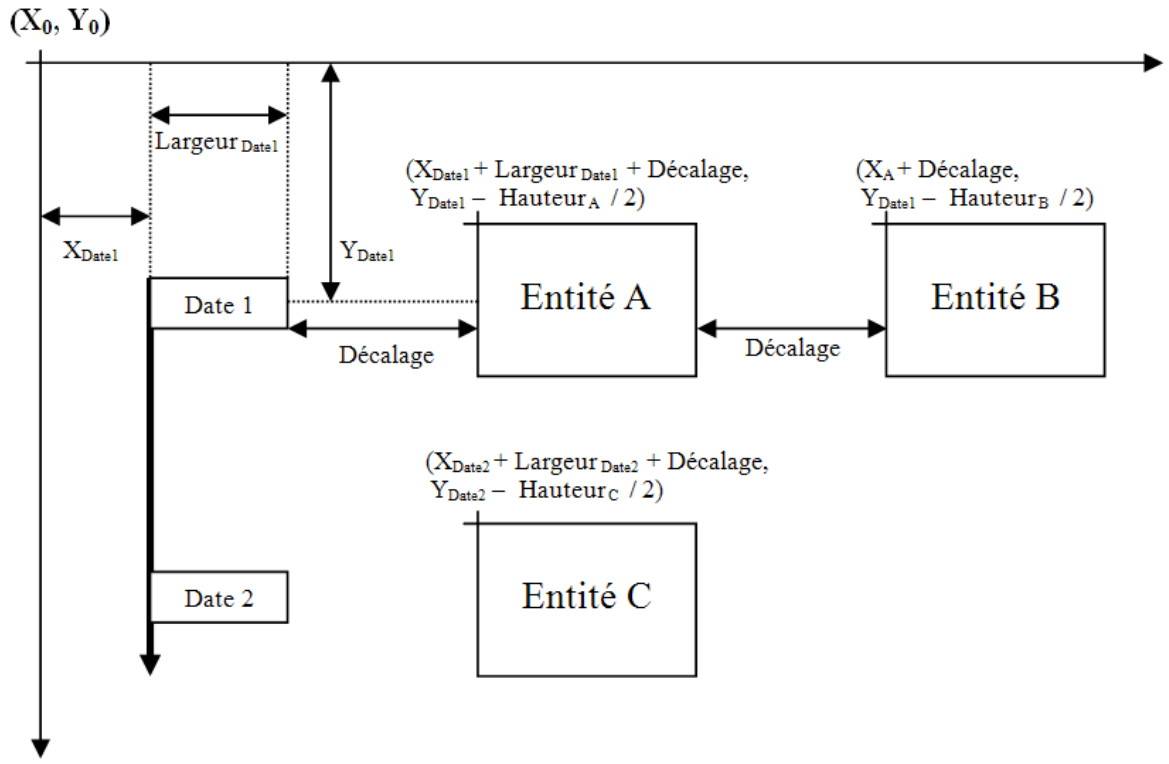

Figure 20. Positionnement des éléments.

Il est important de positionner les communes avant d'effectuer des liens entre elles car on a besoin des coordonnées définitives des communes pour pouvoir tracer les liens aux bonnes coordonnées. Cependant cela a pour résultat de faire passer au premier plan les liens. Si un lien passe sur une commune, le lien masque partiellement la commune. Ainsi, après avoir créé les liens, les communes sont effacées et redessinées pour les refaire passer au premier plan devant les liens.

4.1.1.3. Gestion des événements et des liens

L'algorithme parcourt la liste des événements précédemment établie (Étape 1) afin d'en déduire pour chacun une représentation graphique sous forme de liens.

Pour chaque événement l'algorithme possède la liste des communes qui y participent ; il parcourt donc ces communes afin de déterminer parmi elles quels sont les « Réactifs » et quels sont les « Produits ». Ceci est facilement réalisable car pour chaque commune d'un événement, le fichier JSON fournit cette information : l'attribut *init* d'une commune est égal à *true* si la commune est un réactif, l'attribut *end* est égal à *true* si la commune est un produit.

Toutefois il faut aussi savoir à quelles entités graphiques correspondent ces communes. Les fonctions *getEntiteProduit* et *getEntiteReactif* sont alors appelées : elles renvoient les entités dessinées correspondantes qui sont stockées dans 2 tableaux respectifs afin d'être finalement passées en paramètres de la fonction *relier* qui va alors se charger de dessiner les liens entre réactifs et produits.

4.1.1.4. Redessin des entités

Il est nécessaire de redessiner les entités pour les refaire passer au premier plan. Pour ce faire, l'algorithme efface chaque entité dessinée mais redessine seulement les entités qui possèdent des

liens avec d'autres entités. En effet certaines d'entre elles ne possèdent pas de lien car elles sont mentionnées à une date de leur évolution comme étant absorbées par exemple, de telles communes sont donc supprimées du diagramme.

4.1.2. Validation

4.1.2.1. Test positif

Voici une capture d'écran du diagramme systémique produit pour la commune Aissaines Labesse (Figure 21):

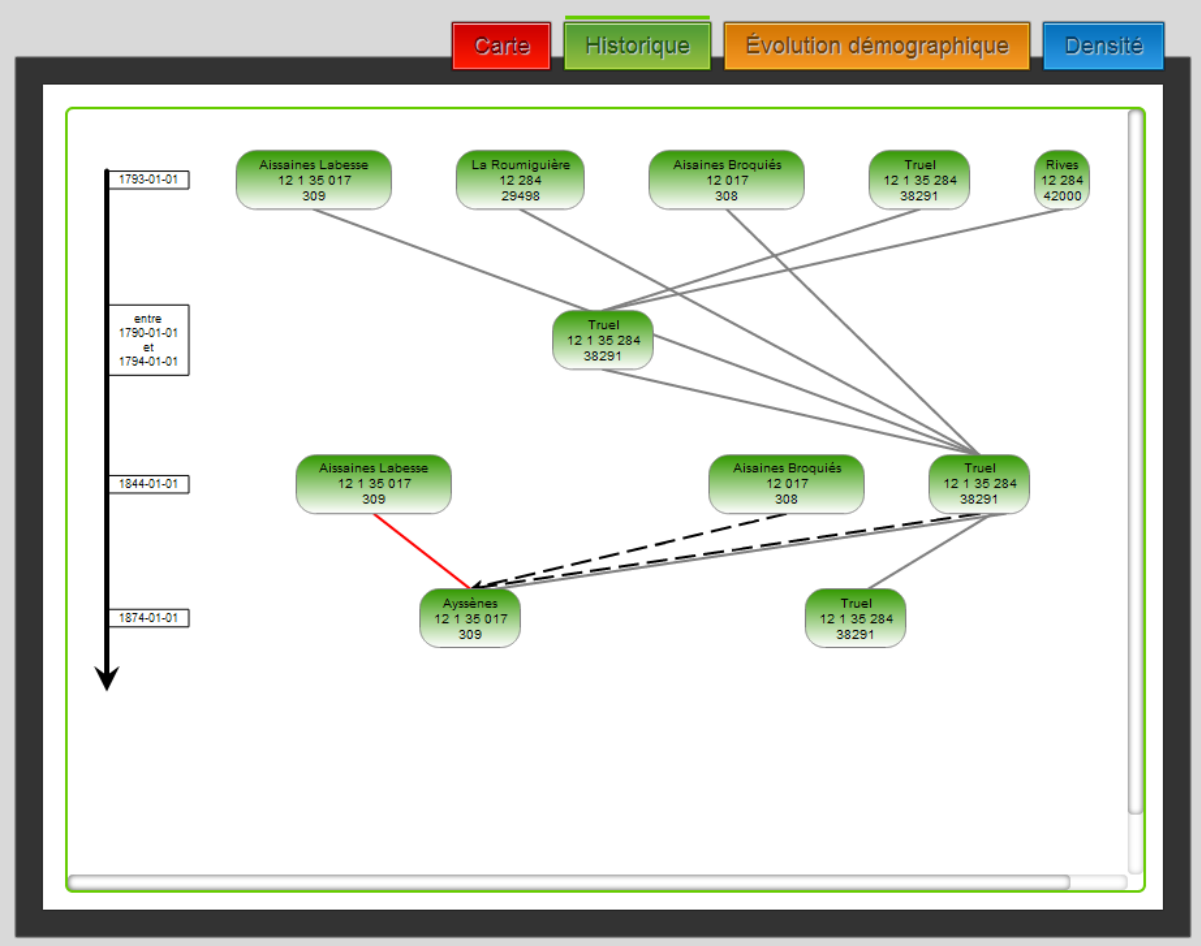

Figure 21. Résultat sur Aissaines Labesse.

Cette capture montre un cas composé de trois événements. Entre 1790 et 1794, la commune de Truel absorbe la commune de Rives qui disparaît. Puis en 1844, Truel absorbe de nouveaux trois autres communes, Aissaines Labesse, La Roumiguière et Aissaines Broquiés. Finalement en 1874, la commune d'Aissaines Labesse est rétablie et se détache de Truel en emportant avec elle une parcelle d'Aissaines Broquiés contenue dans Truel. Dans le même temps, elle change son nom en Ayssaines. On retrouve les informations affichées sur l'entité : le toponyme, le code INSEE et l'identifiant de la commune en base de données. Lorsque le curseur passe sur un lien (Figure 22), celui-ci passe en surbrillance, et si ce lien est cliqué, le motif de l'événement apparaît dessus. Ces motifs peuvent être cachés en recliquant sur le lien ou sur le motif.

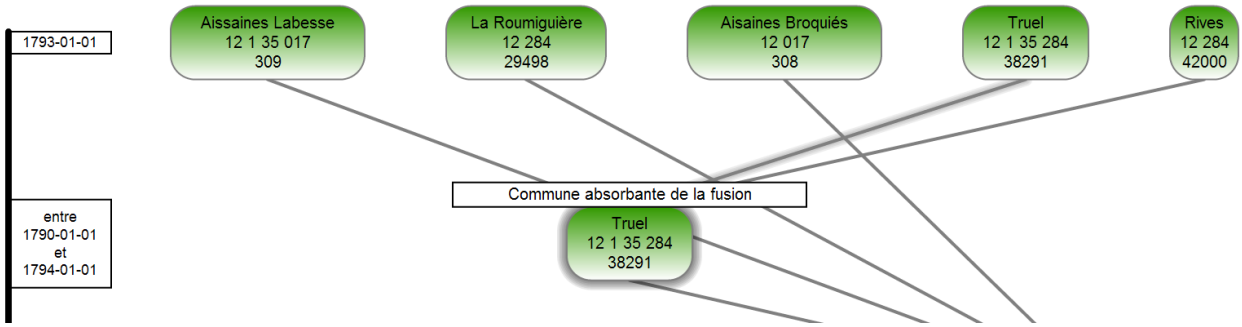

Figure 22. Zoom sur une bulle d'information au clic sur un lien.

4.1.2.1. Tests aux limites

Voici deux cas qui testent les conditions aux limites :

- Requête sur une commune qui n'est impliquée dans aucun événement

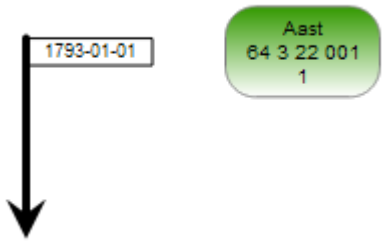

Figure 23. Cas sans événement.

La commune d'Aast apparaît dans la base en 1793, mais elle ne subit jamais d'évolution et n'interagit jamais avec aucune commune, elle apparaît donc seule sur le diagramme, à sa date d'apparition. Ceci signifie que la commune existe encore de nos jours sous ce nom et sous cette forme.

- Requêtes impliquant beaucoup d'événements et de communes

La commune de Paris est au contraire impliquée dans plusieurs événements avec un grand nombre d'autres communes. Pour regarder tout le diagramme, figure 15, l'utilisateur a besoin de faire défiler l'écran sur la droite sur un peu plus de trois écrans. En ce qui concerne la hauteur, le diagramme ne dépasse jamais le bas de l'écran, il suffit donc de défiler l'écran horizontalement. Cependant la lisibilité est fortement réduite dans ce cas à cause du grand nombre de liens qui se rejoignent et se coupent.

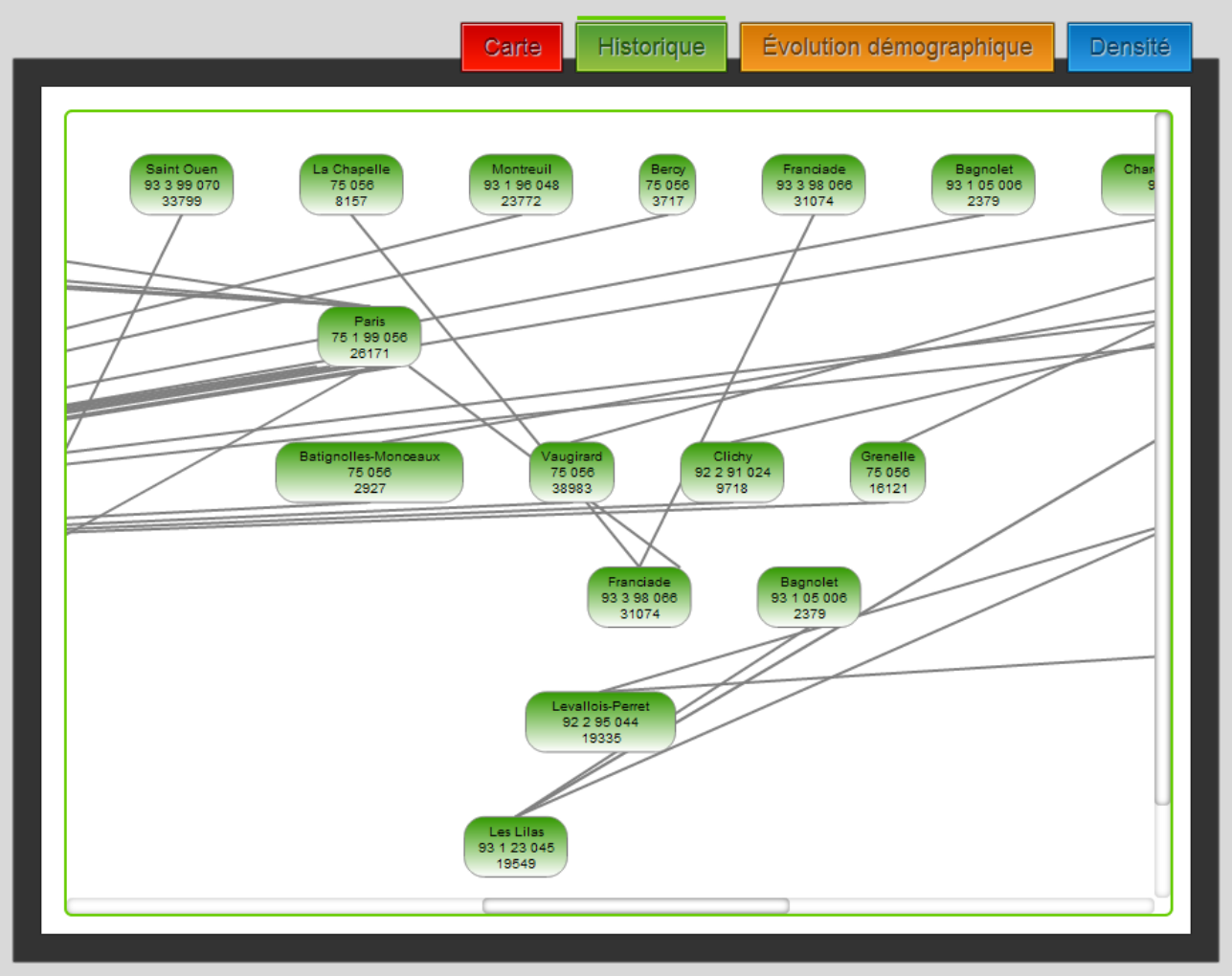

Figure 24. Diagramme systémique de Paris.

4.2. Onglet démographique

L'onglet Démographique doit permettre de visualiser l'évolution de la population d'une commune choisie au cours du temps. Cette évolution est représentée à l'aide d'un graphique comportant une courbe démographique qui décrit le nombre d'habitants qui habitent la commune en fonction du temps.

4.2.1. Algorithme

Les données du fichier JSON qui sont utilisées pour la génération de cet onglet sont enregistrées dans la partie démographie de chaque commune. Si un recensement est lacunaire, il doit être ignoré par le graphique, ce qui a pour effet de produire des interruptions des courbes démographiques.

Dans un premier temps l'algorithme doit récupérer les données nécessaires au calcul du graphique depuis le fichier JSON. Il parcourt toutes les communes contenues dans ce fichier afin d'analyser leur démographie. Pour ce faire, l'algorithme parcourt la liste des recensements de chaque commune. Pour chacun de ces recensements, il vérifie qu'il n'est pas lacunaire.

Dans le cas où ces tests donnent un résultat positif, le recensement, c'est-à-dire sa date et sa population, est enregistré dans un tableau en vue d'être traité durant la deuxième étape. Ensuite le

nom officiel de la commune à cette date est recherché dans l'évolution de la commune et stocké lui aussi dans un tableau.

Dans le cas contraire, si le recensement est lacunaire, il est ignoré et les données qu'il contient ne seront pas prises en compte dans le graphique.

A chaque fois que tous les recensements valides d'une commune ont été stockés, l'algorithme constitue un objet JavaScript qui représente une série de données permettant de tracer la courbe d'une commune sur le graphique.

Cet objet est constitué de deux attributs :

- un tableau d'objets JavaScript contenant les noms officiels de la commune à chaque date,
- un tableau d'objets JavaScript contenant toujours 2 éléments : une date de recensement et le nombre d'habitants à cette date.

Chacun de ces objets représentant une courbe est insérée dans un tableau qui constitue donc l'ensemble des courbes du graphique.

Si une commune est totalement lacunaire sur la période, elle n'est pas représentée par une courbe sur le graphique mais elle apparaît tout de même dans la légende. Une courbe peut être représentée en plusieurs morceaux séparés si la commune est lacunaire sur plusieurs périodes disjointes.

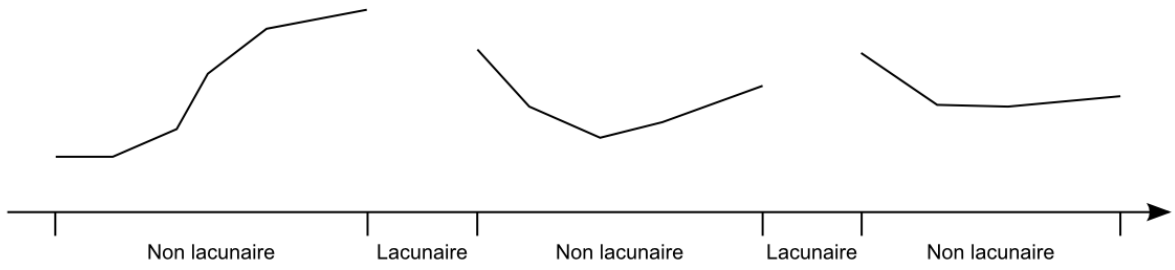

Figure 25. Illustration d'une courbe lacunaire sur certaines périodes.

4.2.1.1. Mise en place du graphique

La bibliothèque *Highcharts* permet de créer automatiquement un graphique en précisant simplement d'une part, tous ses attributs de style et de format et d'autre part, les séries de données que le graphique doit représenter. *Highcharts* propose certaines fonctionnalités, sans programmation supplémentaire. Ainsi, en ce qui concerne le style, les points d'une même série sont reliés entre eux par des tracés courbes, ce qui permet de représenter une tendance sans être nécessairement une interpolation fiable. *Highcharts* permet également de zoomer sur une partie du graphique.

Le zoom fonctionne à la fois horizontalement et verticalement et est activable simplement en cliquant sur une zone du graphique puis en glissant le curseur pour délimiter la zone rectangulaire du zoom. Cette fonctionnalité est personnalisable dans *Highcharts* qui permet trois sortes de zooms : horizontal, vertical et combiné. Ici, le zoom combiné a été activé car il se prête bien à l'analyse des courbes (Figure 26).

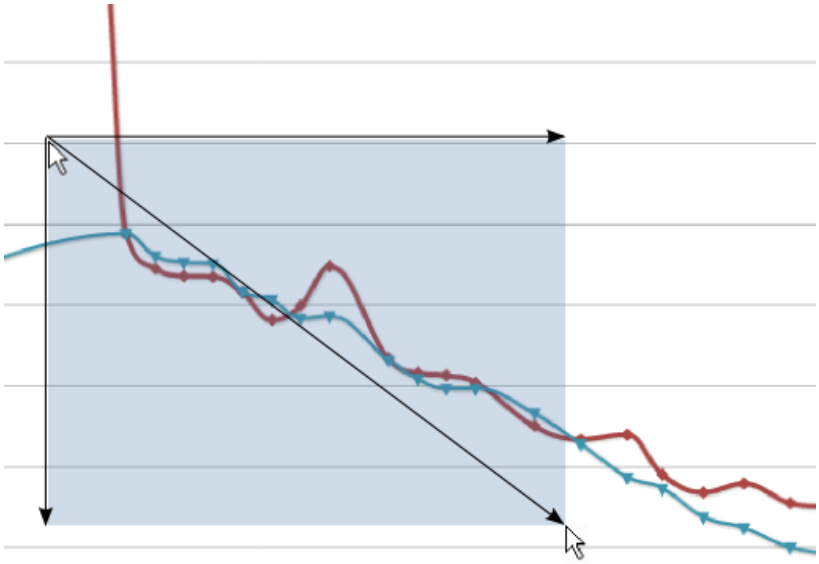

Figure 26. Sélection d'une zone du graphique pour zoomer.

Les points sont représentés par des formes géométriques et des couleurs de façon à ce que chaque série ait son propre motif différenciable des autres, avec de possibles redondances cependant si le graphique représente plus de 45 courbes. En effet il y a en tout 5 types de marqueurs pour les points et 9 couleurs différentes, ce qui donne 9 × 5 = 45 types de courbes différentes. Dans la base de données GéoPeuple, le nombre maximal d'entités en lien, donc de courbes, est de 27 (cas de Paris). Le problème de redondance est donc évité.

Les séries de données, rassemblées à l'étape 1, sont insérées directement dans le graphique. L'important étant que les données (les populations) soient des tableaux de 2 éléments, le premier étant la valeur en abscisse (ici, les dates), le second, la valeur en ordonnées (le nombre d'habitants). Ces mêmes séries comprennent aussi les noms des communes accompagnées de leurs dates.

En sus des fonctionnalités intégrées de *Highcharts*, nous avons développé des fonctions pour l'affichage de métadonnées liées à chaque valeur sur chaque courbe. La définition du graphique permet de donner un format à chaque élément affiché, y compris les textes. Ainsi, un texte est associé à chaque point du graphique, la définition d'un format pour ce texte permet d'utiliser la fonction *getByAttribute* pour récupérer le bon nom de la commune à la date du point en recherchant dans les dates et les noms des communes sauvegardés précédemment. L'algorithme va aussi chercher pour chaque point, si le point qui le suit est lacunaire afin d'afficher en plus la période sur laquelle les lacunes se suivent et la raison de ces lacunes. Pour le nom des communes dans la légende, c'est le nom officiel le plus récent qui est affiché.

4.2.1. Validation

4.2.1.1. Test positif

La Figure 27 est la représentation de la démographie de la commune d'Ayssènes et des communes qui interagissent avec elles. L'axe du temps prend ses valeurs depuis 1793, date des premiers recensements dans la base de données et continue jusqu'à la date la plus récente que propose les séries affichées. Il décompte les années tous les 25 ans.

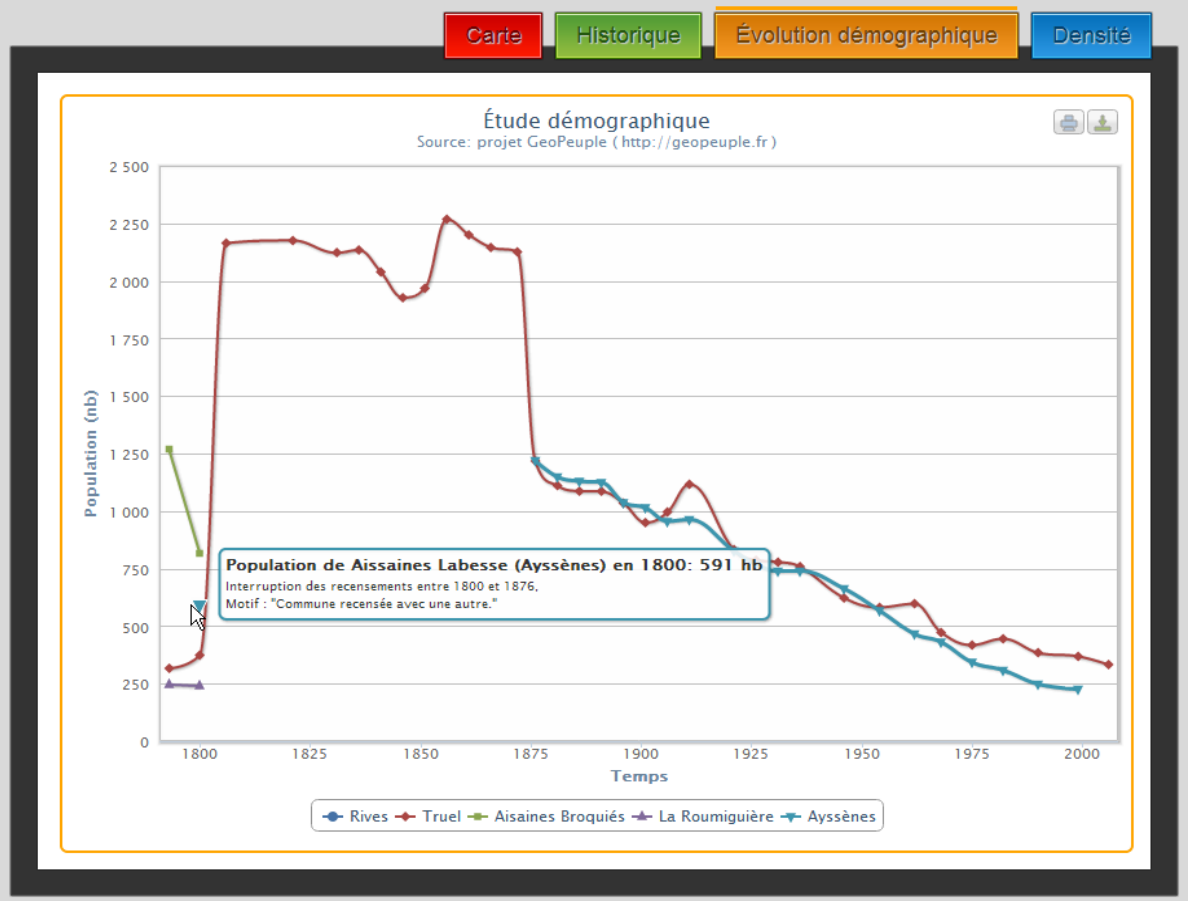

Figure 27. Résultat sur la commune d'Ayssènes.

La commune d'Ayssènes ne possède que des recensements lacunaires entre 1800 et 1876. On ne connaît donc pas en réalité quelle fut l'évolution de la population d'Ayssènes entre ces deux dates, l'apparence de la courbe ne permettant pas de rendre compte de la réalité puisqu'elle est interrompue sur cette période.

Lorsque le curseur survole un point, un cartouche avec une bordure de la couleur de la courbe apparaît et indique la valeur de la population à cette date pour cette commune. Si une interruption de la courbe suit immédiatement ce point, le cartouche affiche en plus sur quelle période les recensements suivants sont lacunaires ainsi que le motif de ces lacunes. Si à la date du point survolé la commune possédait un autre nom que son nom le plus récent, c'est le nom à cette date qui est affiché. Cependant pour ne pas rendre le graphique difficile à comprendre, le nom le plus récent de la commune (qui correspond à celui de la légende) est aussi affiché entre parenthèses dans le cartouche (Figure 28).

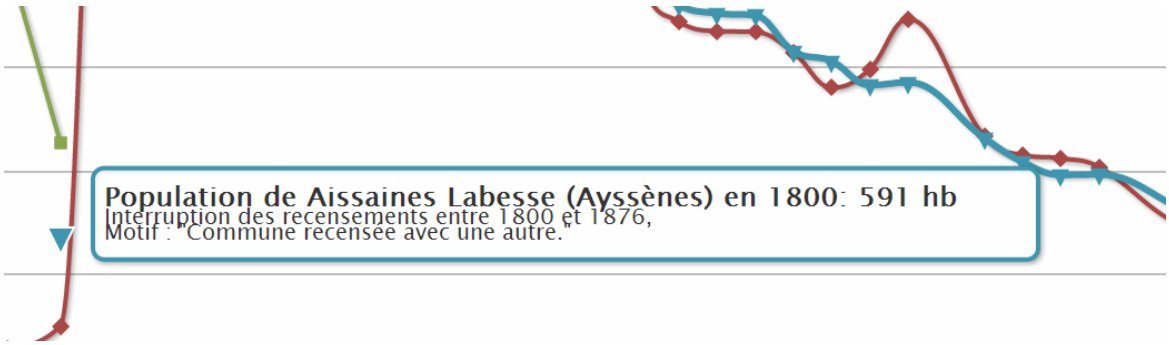

Figure 28. Contenu d'un cartouche au passage de la souris sur un point.

La commune de Rives est indiquée dans la légende mais il n'y a aucune courbe qui représente sa démographie à l'écran car elle est toujours lacunaire à chacun de ses recensements sur la période. Les communes d'Aissaines Broquiés et de La Roumiguière ne possèdent que deux entrées non lacunaires en 1793 et 1800. Après ces dates, tous leurs recensements sont lacunaires, c'est pourquoi leurs courbes s'arrêtent en 1800.

Issue d'une fonctionnalité de *Highcharts*, l'utilisateur peut, en cliquant sur le nom d'une commune dans la légende, faire apparaître ou disparaître selon le cas une commune, et l'échelle du graphique est redimensionnée automatiquement.

4.2.1.1. Test aux limites

- Test avec une grande différence de valeur de population

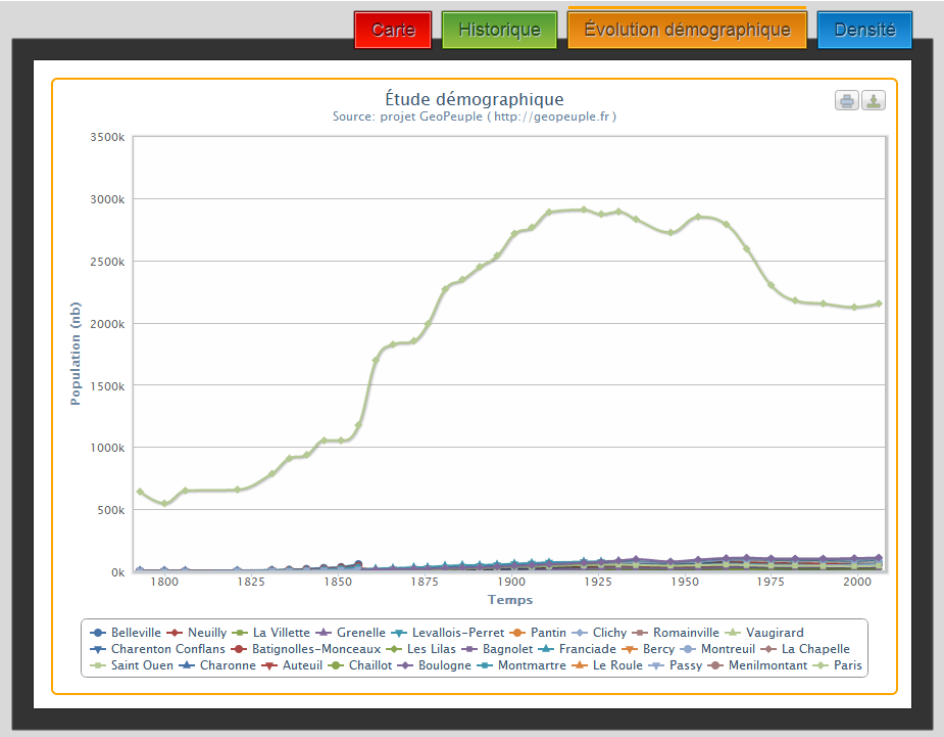

Figure 29. Cas de Paris.

Dans le cas de Paris, Figure 29, la différence entre le nombre d'habitants de la capitale et des communes voisines est telle que ces dernières voient leurs courbes démographiques amassées et confondues au bas du graphique. Il n'est pas possible d'avoir une bonne lecture du graphique dans ce cas. Cependant il suffit de cliquer sur la commune de Paris dans la légende pour faire disparaître sa courbe et ainsi réadapter l'échelle au nouveau contenu illustré par la Figure 30.

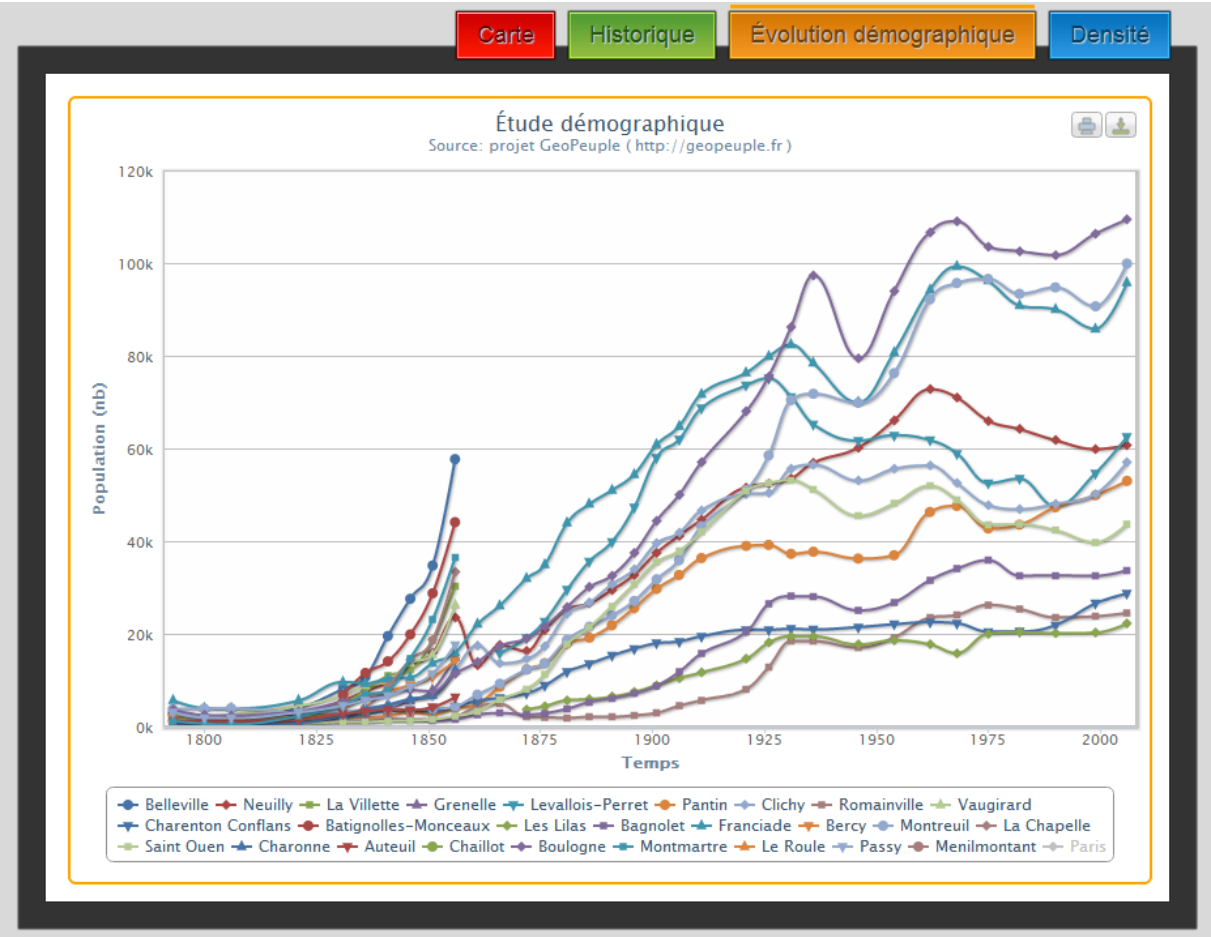

Figure 30. Cas de Paris sans la courbe de Paris.

On discerne mieux dans ce cas les différentes courbes bien qu'elles soient toujours nombreuses et confondues entre 1793 et 1850. Le zoom permet de dépasser cette difficulté (Figure 31 et Figure 32).

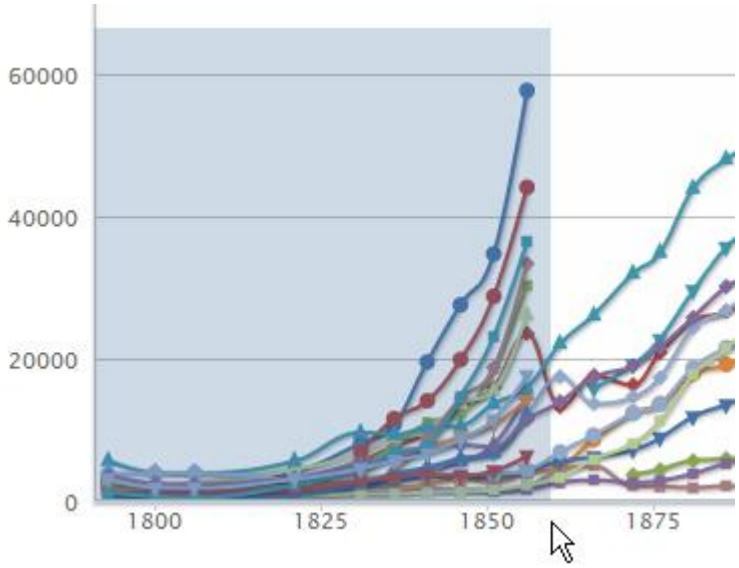

Figure 31. Sélection de la zone de zoom entre 1793 et 1850.

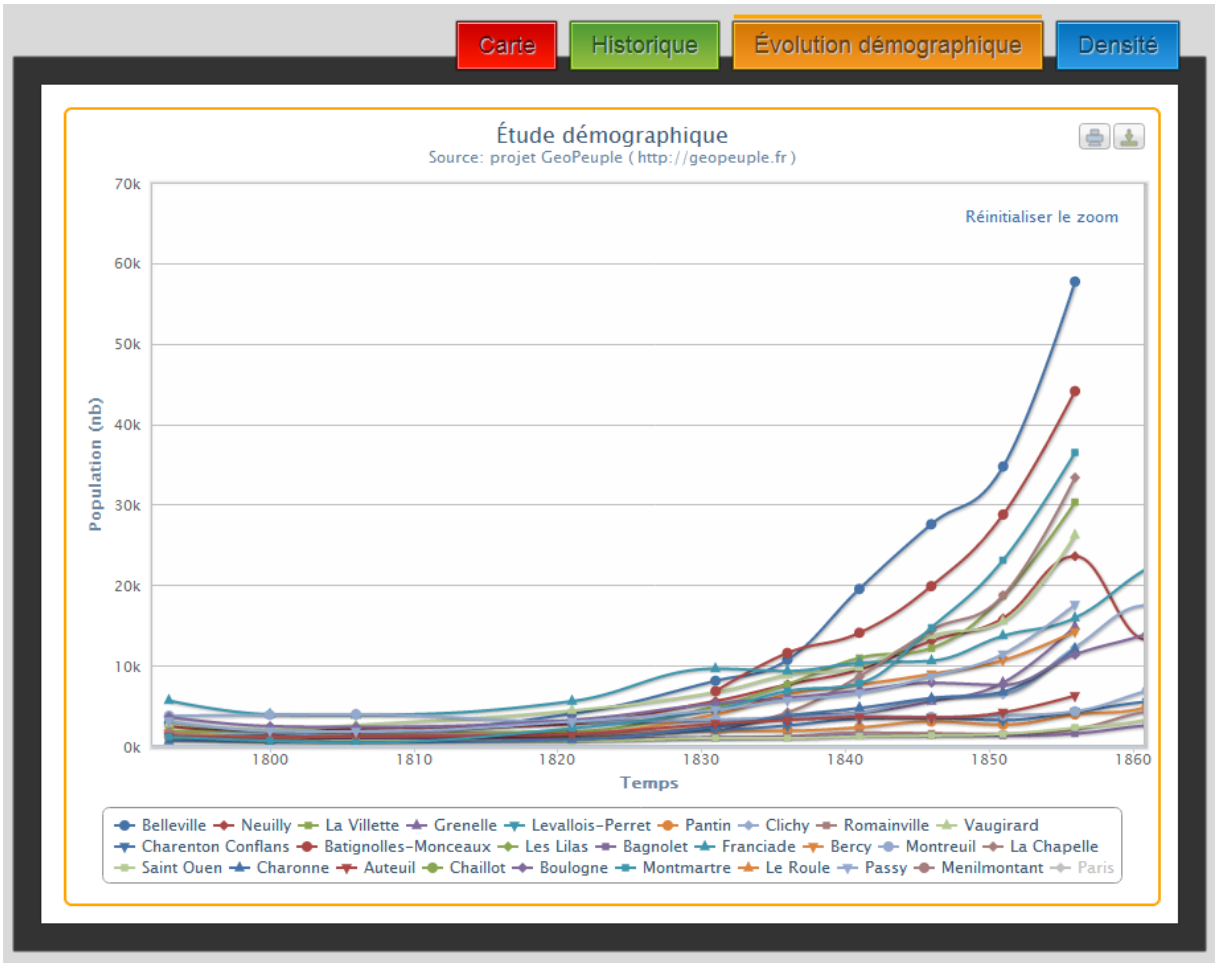

Figure 32. Zoom entre 1793 et 1850.

Highcharts autorise des niveaux de zoom très élevés tout en permettant de cacher à volonté les courbes que l'utilisateur ne veut pas voir. Ceci permet de bien visualiser toutes les données même quand le graphique contient beaucoup de courbes confondues.

4.3. Onglet carte

L'onglet carte a pour objectif de représenter cartographiquement l'évolution territoriale d'une commune ainsi que de celles qui lui sont adjacentes au cours du temps. Comme pour les onglets précédents, l'utilisateur effectue une requête sur une commune en particulier, et les résultats lui sont présentés pour cette commune ainsi que pour les autres communes qui participent aux mêmes événements.

L'évolution territoriale permet de confronter les onglets Historique et Démographique et de fournir soit une illustration des événements décrits dans l'historique de la commune, soit une explication aux variations démographiques présentées dans le graphique de l'évolution de la population.

Le contenu de cet onglet est composé de deux parties. La première est un visualisateur de carte qui permet l'interaction avec l'utilisateur. La deuxième est une ligne temporelle avec un curseur, placée sous le visualisateur de carte. Cette ligne temporelle permet de dater les états des communes affichées à l'écran d'après la position du curseur. Le mouvement du curseur fait évoluer les frontières des communes en fonction de la date sur laquelle il pointe.

4.3.1. Algorithme

4.3.1.1. Chargement de Géoportail

L'**API Géoportail** fournit une fonction appelée *Geoportal.load* qui a pour rôle de charger une instance du visualisateur de carte avec interface utilisateur tout en permettant à l'utilisateur de paramétrer le contenu Géoportail qui sera chargé au démarrage.

On doit préciser dans cette fonction les paramètres suivants :

- les coordonnées du centre de la carte. La latitude et la longitude qui composent ces coordonnées sont récupérées à partir de la *bounding box* définie dans la source de données. En effet, cette dernière décrit un rectangle dans lequel la commune est inscrite. En calculant le centre du rectangle on obtient ainsi le point sur lequel la carte est centrée. Ces coordonnées sont écrites dans la projection Lambert 93 utilisée en France par l'IGN dans le fichier JSON et doivent être converties dans la projection WGS 84 utilisée mondialement pour la localisation par GPS notamment, afin de les utiliser sur la carte,
- le niveau de zoom que l'on fixe ici à la valeur *null* pour être redéfini plus tard dans l'algorithme,
- la classe du visualisateur qui est pour ce cas la classe *Geoportal.Viewer.Default* qui fournit une interface graphique apportant des boîtes à outils pour que l'utilisateur puisse gérer les couches affichées et leur opacité, le niveau de zoom, et consulter les coordonnées du curseur sur la carte ainsi que l'échelle de l'affichage courant,
- les différentes couches à charger, à savoir les cartes de l'IGN, les photos aériennes, les bâtiments, le réseau routier, le réseau hydrographique et l'inventaire forestier,
- les options des couches. Pour toutes les couches définies précédemment, la résolution minimale est de 0.5 m soit une échelle de 1 : 2000 et la résolution maximale est de 2048 m soit une échelle de 1 : 8 192 000. Pour les cartes forestières la résolution minimale est de 2 m soit une échelle de 1 : 8000. Chacune de ces couches n'est visible qu'entre leur résolution minimale et leur résolution maximale et ne sont donc pas visibles pour tous les niveaux de zoom,
- la fonction qui sera appelée lorsque la carte avec tout son contenu est chargée. Cette fonction s'appelle *initMap* et a pour objectif de charger par dessus les cartes, toute la partie vectorielle issue du fichier GeoJSON en entrée.

Geoportal.load retourne une instance du visualisateur utilisée dans la suite de l'algorithme.

4.3.1.2. La fonction initMap

Cette fonction est appelée au déclenchement de l'événement *onView* du visualisateur, c'est-à-dire lorsque le visualisateur a été créé et que les ressources distantes en provenance du Géoportail ont commencé à être téléchargées.

La première étape de cette fonction est de signaler au script principal qui gère les onglets que la carte est chargée ce qui a pour effet, de simuler un clic sur l'onglet carte, d'afficher son contenu et de permettre à l'utilisateur de commencer la navigation à travers les différents onglets. En effet, l'onglet carte est le premier onglet de l'application, qui doit attendre que celui-ci ait terminé son chargement pour pouvoir commencer à utiliser toutes les fonctionnalités.

4.3.1.3. Principe des couches

Le fichier JSON fournit un ensemble de géométries pour chaque commune à plusieurs dates. Toutes les géométries des communes à la même date forment une couche vectorielle superposable à la carte et valable pour cette date. Le but est de montrer l'évolution de ces géométries au fil du temps. Dans un souci de clarté, seulement 2 couches consécutives dans le temps sont affichées à la fois à l'écran. Les transitions entre couches successives se font en superposant les couches et en respectant un style visuel qui indique quelle couche disparaît et quelle couche lui succède (Figure 33).

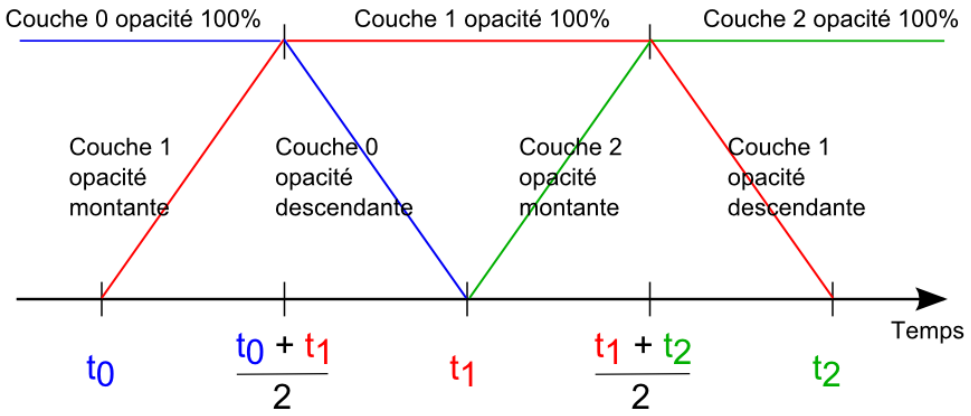

Figure 33. Évolution de l'opacité des couches en fonction de la date.

Au démarrage, les 2 premières couches sont affichées, le pointeur est positionné en t_0, l'opacité de la couche 1 est donc nulle, la couche 1 est totalement transparente. En déplaçant le curseur vers la droite, l'opacité de la couche 1 augmente jusqu'à atteindre 100% à mi chemin entre t_0 et t_1. En continuant jusqu'à t_1, l'opacité de la couche 0 diminue pour atteindre la transparence totale, alors que l'opacité de la couche 1 reste à 100%. En dépassant t_1, la couche 0 est effacée et la couche 2 est chargée. En allant vers t_2, l'opacité de la couche 2 augmente et atteint 100% entre t_1 et t_2. Au-delà, la couche 1 diminue son opacité, et après t_2 une autre couche sera chargée et ainsi de suite.

Pour visualiser correctement ces superpositions de couches, l'algorithme définit toujours une couche inférieure et une couche supérieure à chaque instant t. La couche supérieure est apposée sur la couche inférieure, c'est elle qui voit son opacité évoluer et elle est toujours représentée en pointillés pour pouvoir distinguer les 2 couches en même temps dans les cas où les géométries sont presque identiques d'une date à l'autre. La couche inférieure a toujours une opacité de 100% et est dessinée en traits pleins. A chaque nouvel intervalle de temps l'algorithme doit donc savoir quelle couche est en traits pleins et quelle couche est en pointillés, ce principe est illustré sur la Figure 34.

	De t_0 à $(t_0 + t_1) / 2$	De $(t_0 + t_1) / 2$ à t_1	De t_1 à $(t_1 + t_2) / 2$	De $(t_1 + t_2) / 2$ à t_2
Couche supérieure Pointillés	C1	C0	C2	C1
Couche inférieure Traits pleins	C0	C1	C1	C2

Figure 34. Positionnement et affichage des couches en fonction de la date.

En plus de ce codage du style on ajoute des couleurs aux couches afin de bien les distinguer. Deux couches successives auront toujours deux couleurs différentes. Ces couleurs sont au nombre de 4 à savoir rouge, vert, orange et bleu qui sont également les couleurs des 4 onglets, ceci dans le but de rester dans la même charte graphique que le reste de l'application.

Ces mécanismes permettent de clarifier la superposition et les transitions d'une couche à une autre tout en changeant l'affichage entre deux dates de manière progressive.

4.3.1.4. Chargement des couches vectorielles

La seconde étape est de charger les couches vectorielles encodées dans le fichier GeoJSON.

Pour ce faire, les dates des couches sont extraites du GeoJSON et stockées dans un tableau. Si l'année 2006 n'est pas présente dans les données extraites parce que les événements décrits ne vont pas jusqu'à cette date alors elle est ajoutée au tableau de façon à ce que la date la plus récente soit toujours l'année 2006.

L'algorithme instancie un objet de la classe *RibbonArray* contenant les 4 couleurs citées précédemment. Puis il définit un style par défaut pour les couches et un style pour une commune sélectionnée au clic de la souris. Le style par défaut correspond au style d'une couche inférieure en traits pleins. Le style de sélection permet de mettre en évidence une commune sélectionnée par l'utilisateur en épaississant son trait et en le redessinant en plein s'il est en pointillés.

La classe *Format.GeoJSON* de la bibliothèque *OpenLayers* permet de créer un objet qui lit un objet GeoJSON et retourne un objet *OpenLayers* représentant une collection de géométries accompagnées d'attributs. L'algorithme parcourt les données GeoJSON en utilisant cette classe afin d'instancier des couches vectorielles qui sont stockées dans un tableau. La première couche est définie avec le style d'une couche inférieure en traits pleins en la coloriant de la première couleur du tableau de couleurs. La seconde couche avec le style d'une couche supérieure en pointillés, colorée de la deuxième couleur du tableau et avec une opacité nulle. Les couches suivantes n'ont aucun style défini par défaut si ce n'est qu'elles sont invisibles.

Une fois ces couches constituées elles sont ajoutées à la carte mais ne sont pas visibles dans le gestionnaire de couches du visualisateur afin que l'utilisateur ne puisse pas modifier lui-même leur opacité, leur ordre ou leur visibilité.
Si les données fournies par le GeoJSON représentent moins de 2 couches, aucune évolution n'est démontrable, c'est pourquoi aucune ligne temporelle n'est affichée mais un message indique à la place que les géométries des communes ne sont pas connues avant 2006.

4.3.1.5. La ligne temporelle

Pour construire la ligne temporelle située sous le visualisateur de carte, une flèche grise est d'abord dessinée via la bibliothèque *Raphaël* de gauche à droite afin de représenter le sens du temps. A sa première extrémité, la date de la première couche est affichée, à son autre extrémité c'est l'année 2006 représentant la dernière date connue pour les géométries qui est affichée.

Figure 35. Représentation de la ligne temporelle et des mouvements du curseur.

Il s'agit ensuite de dessiner pour chaque date, un repère vertical sur la ligne temporelle. Chaque repère représente une date à laquelle il existe un état connu différent pour les géométries. Ils sont colorés de la même couleur que la couche affichée sur a carte à cette date afin de permettre à l'utilisateur de repérer facilement quelle couche il manipule. Ainsi les repères reprennent eux aussi comme les couches les couleurs des 4 onglets. Lorsque toutes les couleurs ont été utilisées, la première couleur est de nouveau appliquée au repère suivant et ainsi de suite (Figure 35).

Sous chaque repère est dessinée la date pour laquelle les géométries sont affichées sur la carte. Pour éviter que deux dates ne se chevauchent et ne réduisent la lisibilité de la ligne temporelle dans le cas où deux dates seraient trop proches l'une de l'autre, l'algorithme ne dessine que les dates qui ne se chevauchent pas en calculant la position de chaque date à l'écran et en la comparant avec la largeur d'une date. Ceci a pour conséquence que certaines dates ne sont pas affichées. Cela ne gêne pourtant pas la compréhension de la ligne temporelle puisque les dates effacées sont proches d'autres dates qui permettent de donner une approximation à l'utilisateur.

Si l'utilisateur désire tout de même connaître la date d'une couche bien qu'elle ne soit pas affichée par manque de place, il peut cliquer sur le repère sans date qui l'intéresse pour faire apparaître la date dans un petit rectangle au dessus du repère (Figure 36). Lorsque l'algorithme rencontre un repère pour lequel il n'y a pas de place pour afficher la date, il mémorise cette date de façon à pouvoir l'afficher au clic de la souris. Celui-ci augmente la largeur d'un repère au passage de la souris pour montrer une interaction possible avec. Si l'utilisateur veut cacher cette date, il peut recliquer sur le repère ou sur la date. Pour ce faire, une instance de la classe *RectDate* est créée en passant en paramètres du constructeur le repère correspondant, et la date à laquelle il correspond.

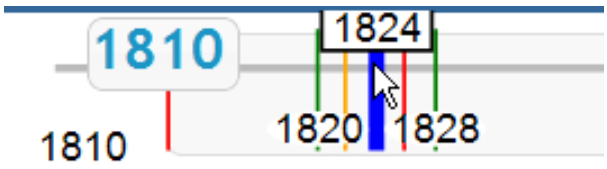

Figure 36. Clic sur un repère pour faire apparaître une date.

Dans le cas où l'année 2006 a été ajoutée de manière artificielle au tableau des dates par l'algorithme, aucun repère n'est dessiné en 2006 car cela signifie que les données du GeoJSON ne fournissent des géométries que jusqu'à une date antérieure. Ainsi, dans ce cas, lorsque le curseur se déplace entre la dernière date et 2006, les couches vectorielles ne varient plus et l'affichage reste celui de l'avant-dernière date, inchangé, alors que sur tout autre intervalle, les frontières changent. En revanche si la dernière date connue est 2006 d'après les données du GeoJSON, alors un repère est dessiné en 2006 et le comportement des géométries reste normal.

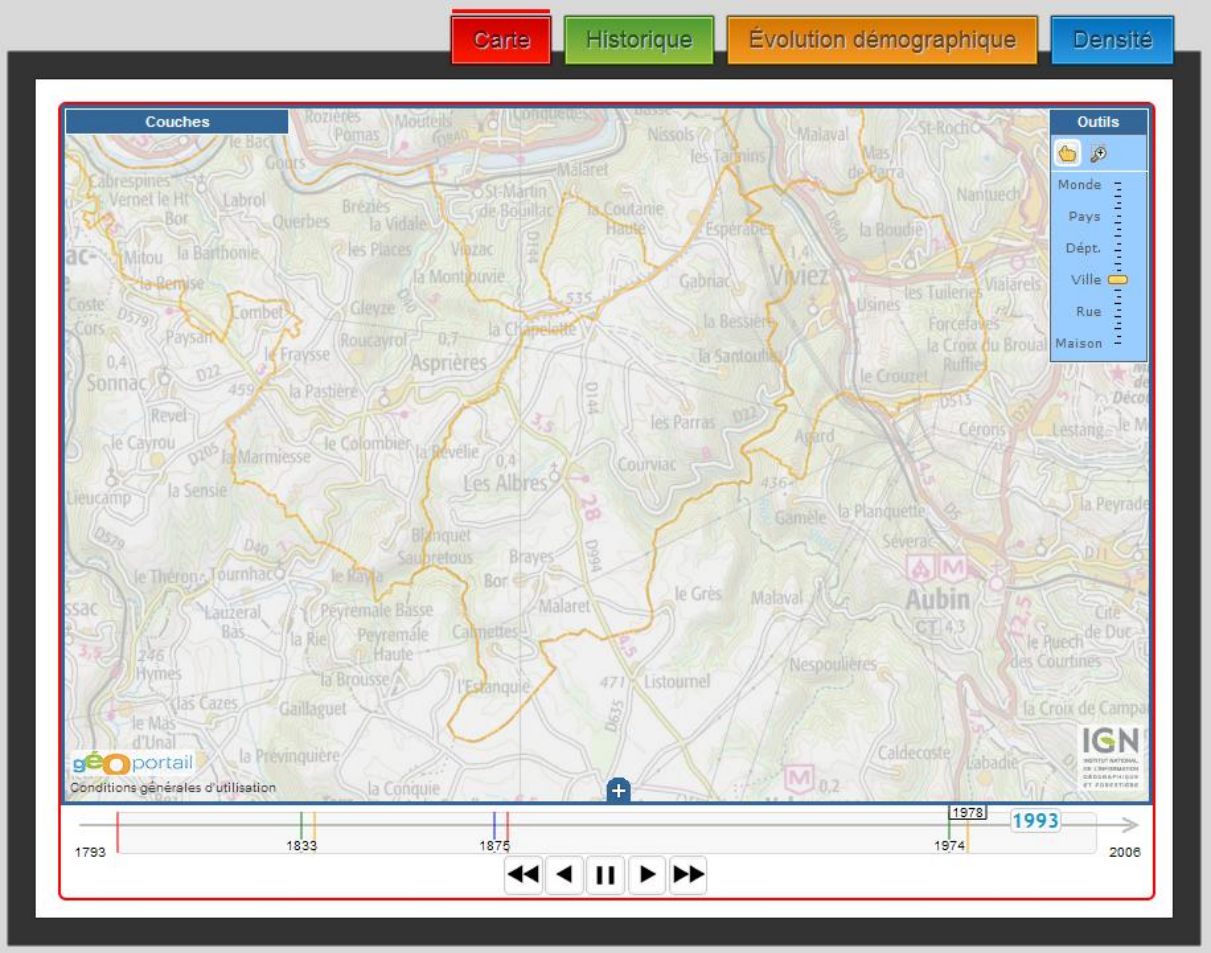

Figure 37. État des frontières entre la dernière date et 2006.

Sur la Figure 37, le dernier repère dessiné représente l'année 1978 car il s'agit du dernier état des frontières des communes connu jusqu'à ce jour. Au-delà de cette date, jusqu'à 2006, l'état des frontières ne changent plus.

Vient ensuite l'affichage du curseur via la fonction *slider* de la bibliothèque jQuery UI. Cette fonction s'applique sur un élément HTML qui devient le conteneur de la barre et de son curseur. Cette fonction prend en paramètre un objet JavaScript qui définit des options pour le curseur. Ces options sont les suivantes :

- ***min*** : la valeur minimale que peut prendre le curseur. Nous la fixons avec la valeur de la première date récupérée depuis les données GeoJSON, donc la date la plus ancienne,
- ***max*** : la valeur maximale que peut prendre le curseur. Nous la fixons avec la valeur de la dernière date, dans tous les cas, 2006,
- ***value*** : la valeur que va prendre le curseur au moment de l'instanciation. Nous lui donnons la même valeur que la première date, ainsi le curseur commence à la date la plus ancienne,
- ***slide*** : une fonction qui est appelée chaque fois que le curseur glisse le long de la barre. Nous lui donnons la valeur d'une fonction déclarée par la suite, la fonction *sliding*,
- ***change*** : une fonction qui est appelée chaque fois que le curseur glisse puis s'arrête ou que la valeur du curseur est changée de manière programmatique, ce qui nous intéresse ici. Nous lui donnons la valeur d'une fonction qui appelle la fonction *sliding* uniquement si l'animation de la ligne temporelle est lancée. En effet l'événement *change* est déclenché même quand l'utilisateur déplace manuellement le curseur mais ici nous ne voulons garder que le cas où le curseur a été déplacé par l'algorithme et non par l'utilisateur.

Une fois le curseur instancié, l'année courante est affichée dessus. Il s'agit donc au départ de la première date enregistrée.

4.3.1.6. Animation automatique du curseur

Le curseur peut être actionné manuellement par l'utilisateur grâce à la souris ou aux boutons directionnels du clavier. Il fait ainsi évoluer les couches vectorielles affichées dans l'ordre chronologique en faisant glisser le curseur vers la droite ou dans l'ordre chronologique inverse en faisant glisser le curseur vers la gauche.

Cependant l'application met aussi à sa disposition des boutons de contrôle du curseur qui permettent de lancer une animation automatique en cliquant dessus. Cinq boutons sont présents et présentent un style *jQuery UI* semblable à celui du curseur pour garder une harmonie du style dans l'interface graphique tel qu'illustré sur la Figure 38. Lorsqu'ils sont survolés par le pointeur de la souris, leurs contours « s'illuminent » en orange, lorsqu'ils sont cliqués leurs contours sont dessinés en pointillés pour montrer quelle animation est en train d'être jouée jusqu'à ce qu'un autre bouton soit pressé ou que l'animation se termine. L'animation se termine lorsque l'animation est stoppée par l'utilisateur ou lorsqu'une extrémité de la ligne temporelle est atteinte par le curseur.

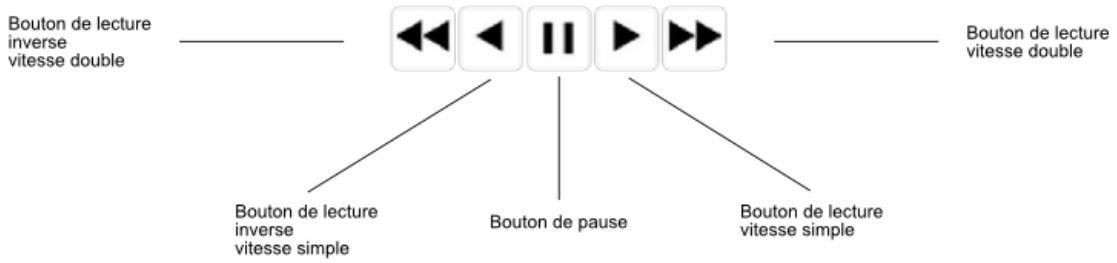

Figure 38. Les boutons de contrôle d'animation.

Deux boutons permettent de faire défiler le curseur dans l'ordre chronologique. Quelle que soit la position courante du curseur, l'animation prend son départ depuis cette valeur mais ne revient pas à la première date pour débuter. La différence entre ces deux boutons est la vitesse de défilement du curseur. Deux autres boutons permettent de faire défiler le curseur dans l'ordre chronologique inverse depuis n'importe quelle position de départ du curseur, avec de même deux vitesses de lecture. Un dernier bouton permet de stopper l'animation. Lorsqu'il est appuyé le curseur s'arrête sur sa valeur courante. Il s'agit du seul bouton dont les contours ne se changent pas en pointillés lorsqu'il est cliqué.

Lorsque l'un des quatre boutons de lecture est appuyé, le curseur se déplace de lui-même et les couches vectorielles se succèdent automatiquement dans le même temps en fonction de l'évolution de la date pointée par le curseur. Lorsque l'animation est mise en pause, les couches vectorielles cessent de se succéder et affichent l'état courant des géométries. Les boutons simulent le déplacement du curseur par l'utilisateur, mais ceci à vitesse constante, de façon à faire une animation contrôlable des transitions des couches.

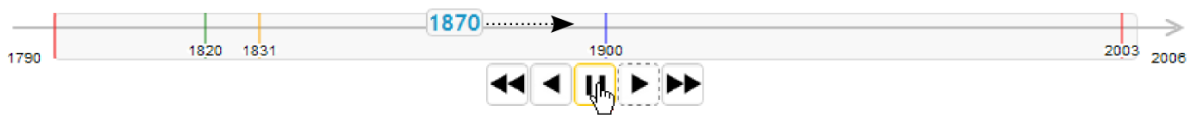

Figure 39. Animation de la ligne temporelle et style des boutons.

Le déclenchement de l'animation par un bouton repose sur l'appel de la fonction nommée *slideAnimation*, qui a pour but de lancer une nouvelle animation quel que soit l'état courant du curseur (s'il a déjà entamé une animation ou non). Cette fonction est appelée au clic d'un bouton de lancement d'animation avec des paramètres différents selon le bouton. Ces paramètres sont le pas et la vitesse de l'animation.

Le pas correspond au nombre d'années parcourues dans un seul mouvement du curseur. C'est ce paramètre qui donne la direction du curseur lors de l'animation. Si le pas est positif, le curseur se déplace vers la droite sinon il se déplace vers la gauche. Dans ce contexte, la valeur absolue du pas est toujours égale à 1 quel que soit le bouton car une valeur supérieure ne permet pas une bonne précision dans les transitions des couches, certaines d'entre elles risquant de ne jamais être affichées si elles sont « sautées » à cause d'un trop grand pas. Les valeurs du pas sont donc de +1 (sens chronologique) et de -1 (sens inverse).

La vitesse correspond à l'intervalle de temps qui s'écoule entre chaque mouvement du curseur exprimé en millisecondes. Les animations à progression rapide ont un intervalle de temps de 75 ms et les animations à progression plus lente ont un intervalle de 150 ms.

La fonction *slideAnimation* commence par arrêter l'animation en cours s'il y en a une, grâce à une variable booléenne qui indique si la fonction *slideAnimation* a déjà été appelée auparavant. Puis cette variable est fixée à *true* pour indiquer qu'une nouvelle animation commence. Pour lancer l'animation, un chronomètre est instancié avec la fonction JavaScript native *setInterval* qui prend en paramètre une fonction à exécuter en boucle à intervalle de temps régulier. Cet intervalle est la vitesse passée en paramètre précédemment. La fonction répétée en boucle incrémente la valeur du curseur avec le pas passé en paramètre et vérifie que le curseur est arrivé à la première ou à la dernière date auquel cas le chronomètre est arrêté et l'animation se stoppe. Le fait de changer la valeur du curseur déclenche l'événement *change* du curseur qui appelle alors la fonction *sliding* seulement si une animation est en train de se jouer ce qui est le cas ici.

4.3.1.7. La fonction sliding

La fonction *sliding* a pour objectifs de gérer la transition des couches vectorielles à l'écran et d'actualiser leur style en fonction de l'année sur laquelle pointe le curseur. Elle prend en paramètres un objet JavaScript qui représente l'événement qui a appelé la fonction ainsi que le composant graphique sur lequel l'événement a été déclenché, ici le curseur. Ce second paramètre permet de récupérer la valeur courante du curseur, c'est-à-dire l'année à prendre en compte pour actualiser l'affichage des couches.

Cette fonction est appelée chaque fois que le curseur change de valeur, que ce changement soit dû au déclenchement d'une animation ou à l'action directe de l'utilisateur.

La première étape de son algorithme est d'actualiser la date affichée sur le curseur avec la valeur courante du curseur.

Dans un deuxième temps il va rechercher quelle date appartenant à une couche est la plus proche de la valeur courante du curseur. Pour ce faire, il parcourt les dates des couches et compare la distance en années entre la valeur courante du curseur et la date courante avec la distance entre la valeur courante du curseur et la date la plus proche mémorisée. Si la première distance est la plus petite, la date courante devient la date la plus proche. L'indice de cette date dans le tableau des dates est aussi mémorisé.

En troisième temps l'algorithme va gérer l'opacité des couches. Si le curseur est situé à droite de la date la plus proche, la couche qui suit la couche de la date la plus proche dans l'ordre chronologique augmente son opacité. Si le curseur est situé à gauche de la date la plus proche, c'est la couche précédente qui diminue sont opacité.

En dernier temps l'algorithme doit gérer le changement des couches vectorielles. Ce changement n'a lieu que si le curseur a changé de position par rapport à la date la plus proche, c'est-à-dire s'il est passé à droite de cette date alors qu'il était à gauche précédemment ou inversement, s'il est passé à gauche de cette date alors qu'il était à droite. Pour changer les couches, les couches visibles dans l'état courant sont effacées puis la nouvelle couche inférieure et la nouvelle couche supérieure doivent être déterminées.

La couche inférieure est la couche correspondant à la date la plus proche qu'on obtient facilement grâce à l'indice de la date la plus proche mémorisé précédemment. La couche supérieure est la couche qui suit la couche inférieure dans l'ordre chronologique si le curseur est placé à droite de la date la plus proche ou bien la couche qui précède la couche inférieure dans l'ordre chronologique si le curseur est placé à gauche de la date la plus proche.

La couleur de la couche supérieure est obtenue en utilisant la méthode *getByIndex* de l'instance de la classe *RibbonArray* contenant les couleurs, en lui passant en paramètre l'indice de la date la plus proche. La couleur de la couche inférieure est obtenue en utilisant la méthode *getByIndex* avec en paramètre l'indice qui suit l'indice de la date la plus proche si le curseur est placé à droite de la date la plus proche, sinon avec l'indice qui précède la date la plus proche.

Ensuite la fonction *afficherCouches* est appelée. Elle prend en paramètres deux couches, la première est la couche inférieure, la seconde la couche supérieure. Elle change l'indice de la couche supérieure dans le visualisateur pour la mettre au plus haut niveau c'est-à-dire qu'elle est posée par dessus de toutes les autres. Puis elle actualise le style par défaut et le style de sélection pour les deux couches et les redessine sur la carte.

4.3.1.8. Sélection d'une commune

L'algorithme définit un style pour les communes sélectionnées par l'utilisateur. Ce style est appliqué sur une commune lorsqu'elle est considérée comme étant sélectionnée. Une commune sélectionnée est mise en évidence en épaississant son contour en traits pleins même s'il est en traits pointillés.

La bibliothèque *OpenLayers* fournit la classe *Control.SelectFeature* qui permet de définir le comportement d'une caractéristique lorsqu'elle est sélectionnée et lorsqu'elle est désélectionnée. En instanciant cette classe, l'algorithme assigne à toutes les géométries des communes le même comportement. Lorsque l'on clique sur une commune un petit pop-up apparaît dans lequel est écrit le nom de la commune et le nombre d'habitants à l'époque qu'indique le curseur.

4.3.1.9. Adaptation du zoom

Pour finaliser la vue sur la carte, l'algorithme calcul le niveau zoom le mieux adapté à la *bounding box* de la commune et le fixe sur la carte de façon à ce que l'utilisateur puisse voir la commune et ses voisines.

4.3.2. Validation

4.3.2.1. Test positif

Sur la carte présentée Figure 40, la commune de Viviez a été sélectionnée au moyen d'un clic de la souris. Ses contours apparaissent en vert épais car elle fait partie de la couche vert pointillée apparaissant sur la couche rouge en traits pleins. Son nom et le nombre d'habitants à l'époque indiquée par le curseur (1808) sont affichés dans un pop-up blanc sous les contours de la commune. Ici, on voit que le nom de la commune a changé puisqu'il s'écrivait alors Viviers.

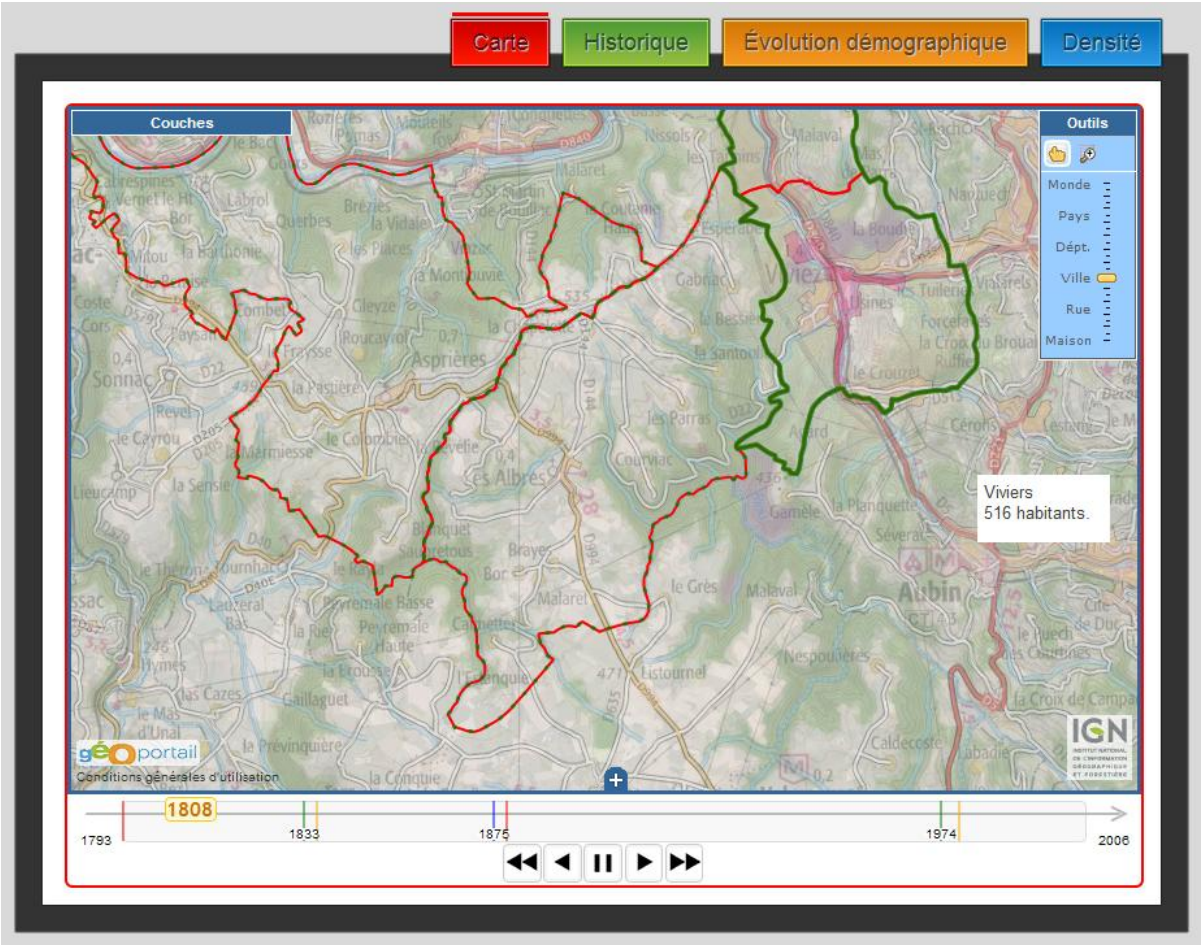

Figure 40. Sélection sur Viviez.

4.3.2.2. Tests aux limites

- Tests avec de grands mouvements du curseur

Si l'utilisateur déplace le curseur avec de trop amples et trop rapides mouvements, au moyen de la souris, il se peut que le curseur passe d'une année à une autre de manière de manière non progressive, c'est-à-dire qu'il ne passe pas par toutes les valeurs présentes entre le moment où il commence à se déplacer et le moment où il s'arrête. Il peut par exemple passer directement de l'année 1800 à l'année 1900 sans passer par les années intermédiaires. L'application est pensée de façon à ce que le curseur passe exactement par toutes les valeurs intermédiaires pour passer d'un

état des géométries à un autre. Le fait de sauter ces valeurs peut avoir des conséquences sur le style des géométries qui peuvent alors présenter des anomalies.

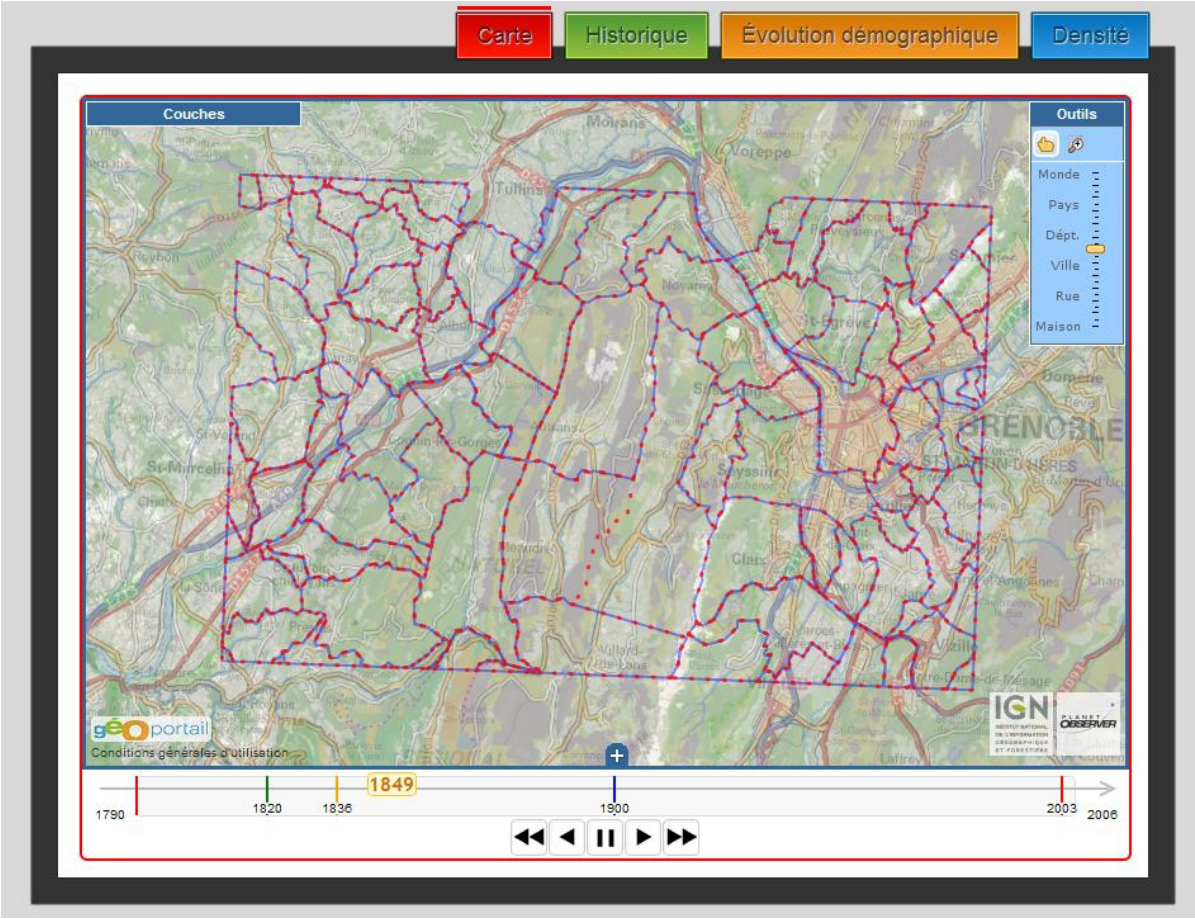

Figure 41. Anomalie sur l'apparence des couches.

Sur la Figure 41 ci-dessus, le curseur est placé entre une date à géométrie orange (1836) et une date à géométrie bleue (1900), plus proche de la couche orange. La couche inférieure en traits pleins devrait être orange et opaque mais elle est bleue et n'est pas complètement opaque alors que les couches inférieures ne doivent jamais voir leur opacité diminuer. La couche supérieure en pointillés présente elle aussi des anomalies puisqu'elle devrait être bleue mais elle est rouge, de plus le niveau d'opacité est trop élevé, pas assez transparent. Le contour des géométries n'est pas non plus exact à cette date.

Le problème posé ici est que les géométries ne s'actualisent pas comme elles le devraient. En effet l'actualisation des couches se fait relativement à la date précédente. Or la logique temporelle n'est plus respectée si l'on passe d'une année à une autre qui ne lui est pas contigüe. Le curseur étant placé sur l'année 1849 dans ce cas, l'état des géométries devrait être calculé par rapport à l'année 1848 ou l'année 1850 selon le sens de déplacement. Cependant il est calculé par rapport à une date non contigüe ce qui laisse apparaître un état différent de ce qu'il devrait être à cette date.

Cependant l'aspect des géométries revient à la normale dès lors que le curseur se déplace de nouveau en passant exactement par toutes les années et que les couches sont rechargées une fois au passage à une autre date. Pour cette raison il est préférable d'utiliser les flèches directionnelles du clavier ou bien les boutons d'animation automatique qui font avancer le curseur valeur par valeur.

- Test avec des dates consécutives très proches

Lorsque deux dates de géométries consécutives sont trop peu espacées (moins de 3 années d'écart), tel qu'illustré Figure 42, une anomalie se produit lorsque le curseur passe entre les deux dates (Figure 43).

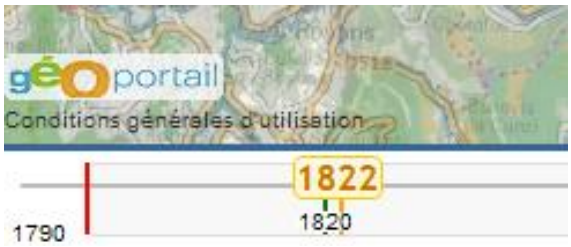

Figure 42. Curseur entre 1820 et 1822, deux dates avec un faible écart.

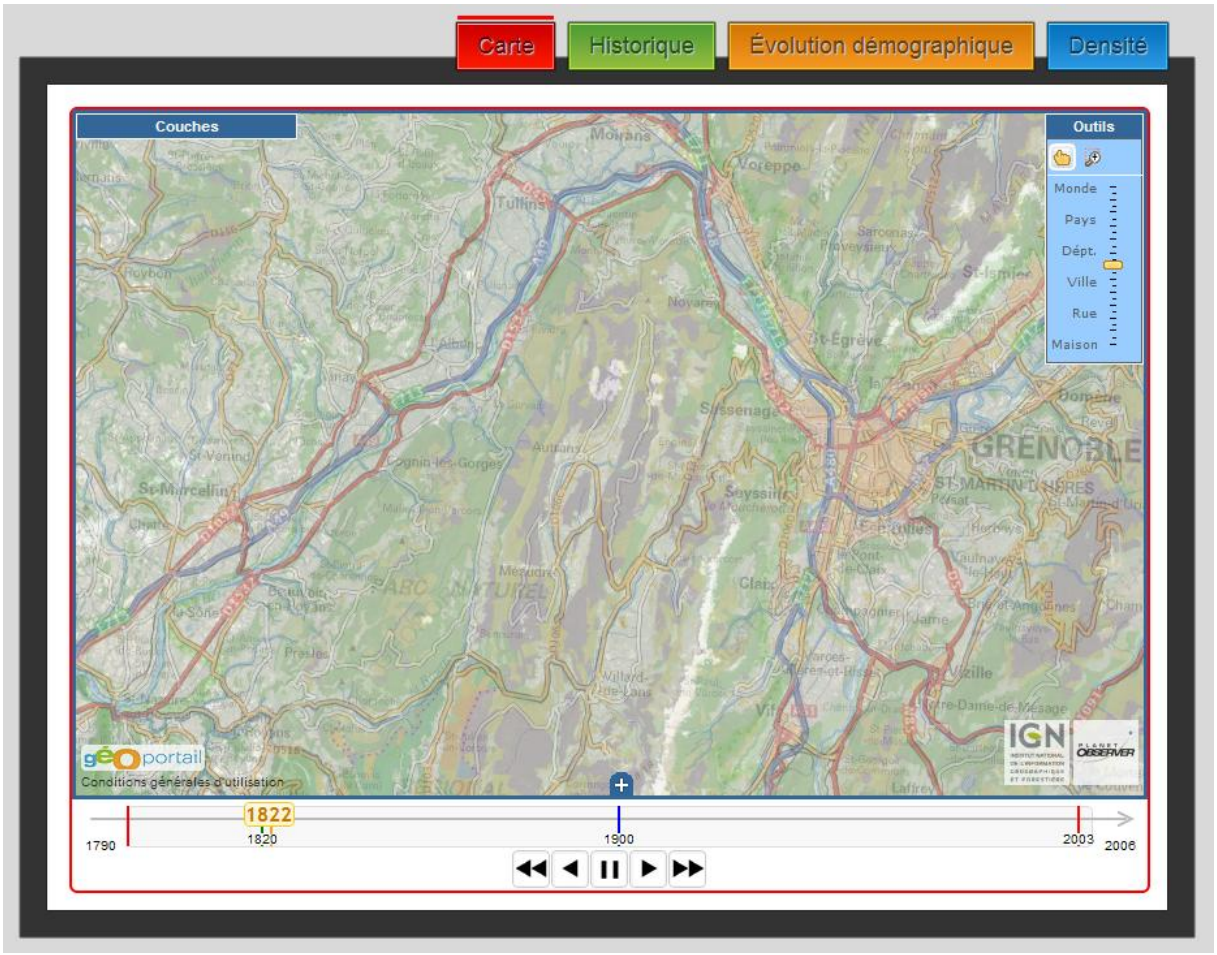

Figure 43. Disparition des géométries avec deux dates trop proches.

Lorsque le curseur est placé sur l'année 1822 en venant de la gauche, la couche de 1822 normalement affichée en traits pleins orange n'apparaît pas. Cependant en dépassant cette date, le comportement redevient normal.

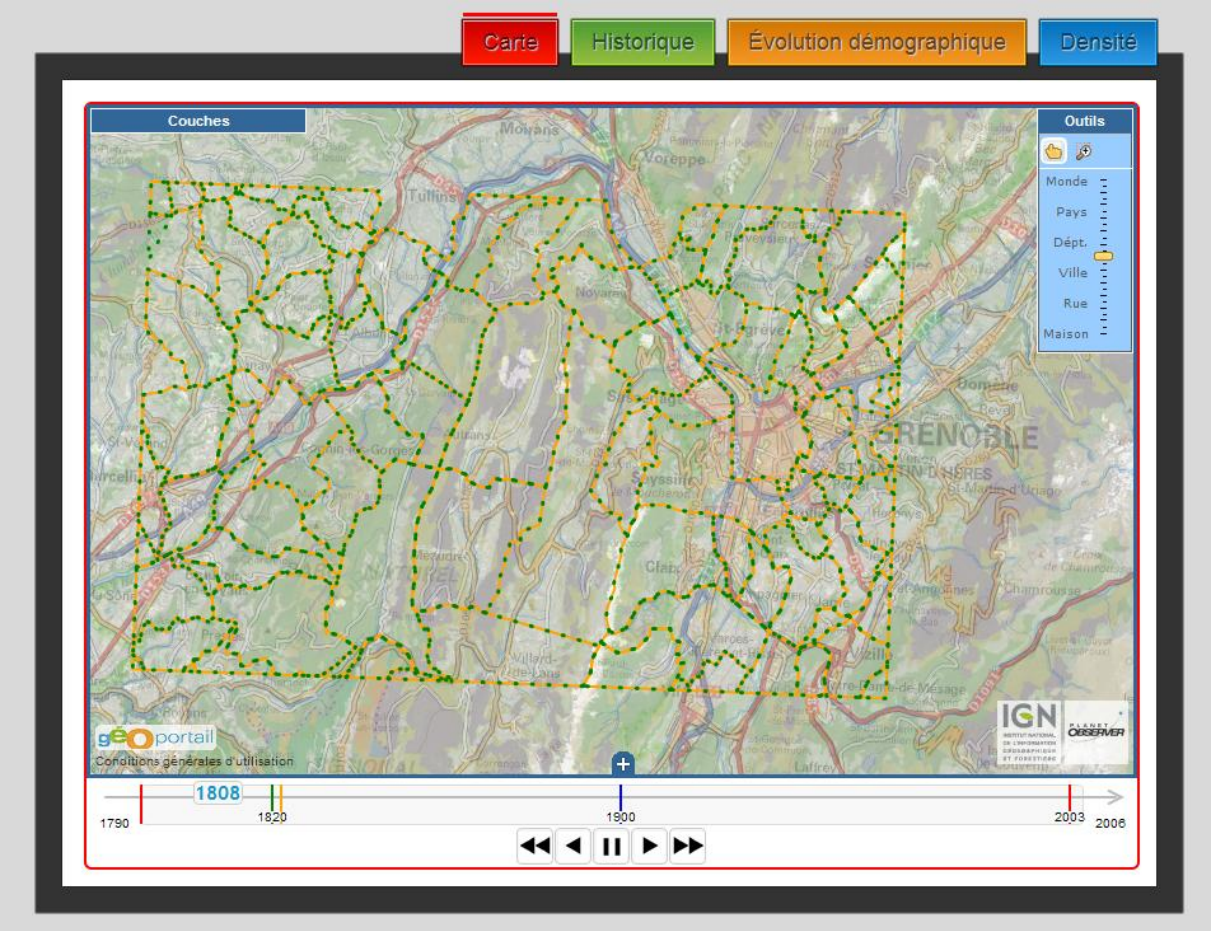

Figure 44. Erreur d'affichage des géométries avec deux dates trop proches.

Lorsque le curseur dépasse 1822 et revient en arrière en dépassant 1820 (1808 sur la Figure 44), la couche rouge pointillée de 1790 devrait apparaître mais c'est l'état des couches entre 1820 et 1822 qui apparaît jusqu'à ce que les couches soient rechargées à mi chemin entre 1790 et 1820, le comportement redevient alors normal.

Pour s'assurer que ce problème ne survienne pas, il faut au moins 3 années de différence entre deux dates consécutives car les couches s'actualisent à mi-chemin entre deux dates. Or il n'y a qu'une seule année intermédiaire entre 1820 et 1822, il n'y a donc pas deux intervalles situés l'un entre 1820 et 1821 et l'autre entre 1821 et 1822 mais seulement un seul. Le changement des couches ne peut donc pas s'effectuer de manière progressive.

L'algorithme ne permet donc pas que deux dates soient trop proches, en forçant la distance minimale à trois ans. Si l'écart entre deux dates est inférieur à cette distance, il est automatiquement ajusté ce qui visuellement ne donne plus aucune différence entre un écart de un, deux ou trois ans. Cependant la date affichée sur le repère est toujours la vraie date avant changement.

4.4. Onglet densité

L'onglet Densité montre l'évolution du rapport entre le nombre d'habitants et la surface d'une commune requêtée au cours du temps à l'aide d'un graphique en colonnes. Cet onglet vient compléter les données représentées dans l'onglet Carte et dans l'onglet Démographique. En effet, si

la démographie est représentée quantitativement sur ces onglets, la surface des communes en revanche n'est représentée que de manière relative par les géométries affichées dans l'onglet carte. Comme pour les autres onglets, la densité est calculée pour toutes les communes qui interagissent avec la commune requêtée au travers d'évènements. La densité, permet de rendre compte de l'importance des communes à la fois d'un point de vue territorial et d'un point de vue du peuplement.

4.4.1. Algorithme

Dans un premier temps, l'algorithme doit récupérer les données nécessaires à l'affichage du graphique depuis le fichier JSON. Il parcourt toutes les communes contenues dans ce fichier afin d'analyser leur démographie. Pour ce faire, l'algorithme parcourt la liste des recensements de chaque commune. Le but est de constituer un tableau contenant les valeurs des densités affichées en ordonnées du graphique.

Pour chaque recensement de chaque commune, l'algorithme teste si le recensement est lacunaire ou si la surface n'est pas connue, aux quels cas, la densité est tenue pour étant égale à -1. Si la population et la surface sont connues, la valeur de la densité est calculée en faisant le rapport de l'un par l'autre. La valeur est ensuite stockée dans un tableau. Ce tableau existe pour chaque commune du graphique.

Dans le même temps, se constitue un tableau des dates des recensements. Les communes n'ayant pas toujours les mêmes dates de recensement, toutes les dates des recensements de toutes les communes sont parcourues. Chacune d'elles est gardée en mémoire dans le tableau en excluant toutefois les doublons grâce à la méthode *contains* qui permet de savoir si le tableau contient déjà une date. Le tableau ainsi formé représente les dates de tous les recensements de l'ensemble de ces communes. Ces dates constituent les données en abscisses du graphique. Ce tableau existe une fois pour toutes les communes du graphique.

Encore dans le même temps, l'algorithme remplit un tableau de noms officiels pour chaque commune à chaque date. Le but de ce tableau est de pouvoir faire apparaître sur le graphique le nom qu'une commune possède à la date de chaque recensement pour tenir compte des différents changements de noms qui se sont produits au cours du temps. Ce tableau existe pour chaque commune du graphique.

Finalement, pour chaque commune, les tableaux des densités et des noms officiels sont enregistrés dans un tableau constituant les séries de données du graphique. Chaque valeur de ce tableau est un objet JavaScript, composé d'un attribut *name* contenant le tableau des noms officiels et d'un attribut data contenant les densités. D'après *Highcharts* l'attribut *name* est habituellement une seule chaîne de caractères qui est le nom de la série de données. Ici, ce nom serait le nom de chaque commune, mais pour prendre en compte l'évolution des toponymes, on a besoin de mémoriser tous les noms à chaque époque.

4.4.1.1. Mise en place du graphique

La bibliothèque *Highcharts* permet de créer automatiquement un graphique en précisant simplement d'une part, tous ses attributs de style et de format et d'autre part, les séries de données que le graphique doit représenter.

De même que pour l'onglet Démographique, *Highcharts* permet de représenter les différentes séries

de données en les différenciant par des couleurs. Cependant, dans le cas du graphique en colonnes, il n'y a pas d'identité visuelle propre à chaque série autre que la couleur car les rectangles ne peuvent pas comme les points présenter chacun un style différent sans troubler l'harmonie du graphique et gêner la lisibilité. Ainsi ce type de graphique peut représenter 9 séries sans redondance visuelle, puisque de base, *Highcharts* propose 9 couleurs différentes. Au-delà, deux séries peuvent se confondre, bien que l'emplacement de chacune d'elles doive normalement permettre à l'utilisateur de faire la différence.

Contrairement à un graphique représentant des courbes, ici un zoom à la fois horizontal et vertical ne produit pas un bon effet car il faut que l'utilisateur puisse toujours voir à la fois la base et le sommet de chaque rectangle. En conséquence, nous empêchons l'utilisation du zoom vertical pour ne laisser que le zoom horizontal (Figure 45).

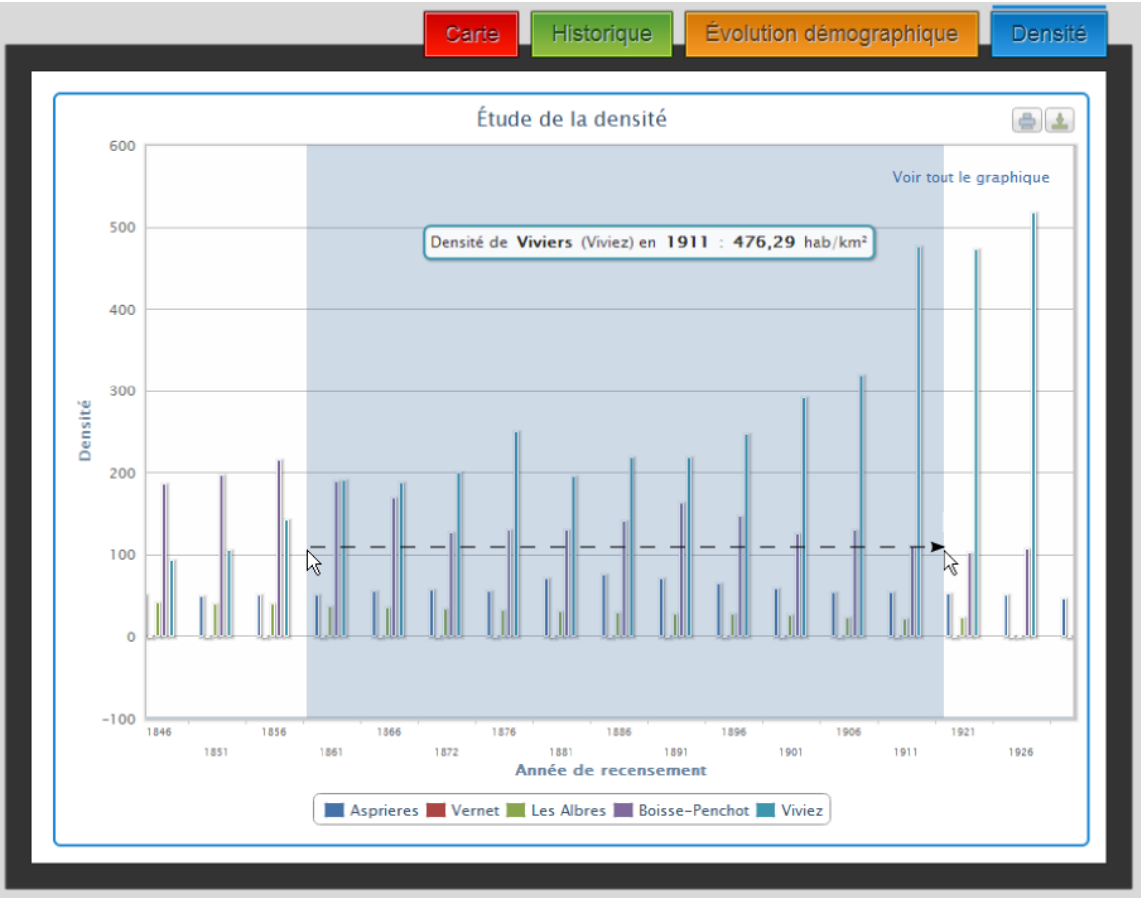

Figure 45. Zoom horizontal.

En ce qui concerne le style de l'axe des abscisses, les dates sont affichées sur deux lignes parallèles l'une au dessous de l'autre car l'espace dans lequel tient le graphique étant réduit, si les séries présentent un grand nombre de dates, elles se confondent à l'affichage. La première date est affichée sur la ligne du haut, puis la seconde date est affichée sur la ligne du bas, la troisième est affichée sur la ligne du haut et ainsi de suite en alternant une fois sur deux. Cet agencement permet d'améliorer la lecture des dates lorsque le graphique est regardé dans sa globalité (Figure 46).

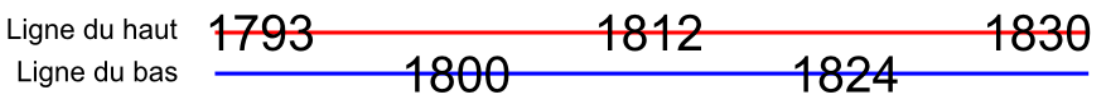

Figure 46. Schéma de l'agencement des dates.

Le titre des séries de données affichées dans la légende du graphique doit normalement être le nom de chaque commune. Cependant nous avons mémorisé le nom de chaque commune à chaque date. Nous choisissons d'écrire dans la légende le nom de la commune le plus récent car c'est le nom sous lequel la commune est connue aujourd'hui ou en tout cas jusqu'en 2006. *Highcharts* permet de reformater chaque élément du graphique au moyen de fonctions qui sont appelées dans des circonstances particulières. Dans le cas de la légende, l'algorithme définit une fonction qui est appelée au moment où le nom de chaque série est affiché dans la légende lors de la construction du graphique. Cette fonction permet d'afficher la première valeur du tableau des noms officiels pour chaque commune ce qui correspond au nom le plus récent.

Nous reformatons également le contenu de la bulle affichée lorsque la souris passe sur un rectangle. Par défaut, l'abscisse et l'ordonnée du rectangle sont affichés ce qui correspond ici à la date du recensement et à la densité de la commune. La bulle contient ici le nom de la commune, la date du recensement et la densité si elle est connue sinon un message indique qu'elle est inconnue. De même que pour le graphique de l'onglet Démographique, la fonction *getByAttribute* (voir la partie sur l'onglet Démographique) est utilisée pour retrouver le bon nom de la commune à la date du recensement en recherchant dans les dates et les noms officiels des communes sauvegardés précédemment. Ainsi le nom de la commune qui est affiché est celui qu'elle portait lors du recensement. Le nom actuel de la commune affiché dans la légende est aussi ajouté entre parenthèses à côté de l'ancien nom, comme l'illustre par exemple la Figure 47.

Figure 47. Bulle affichant l'ancien nom d'une commune.

Les recensements pour lesquels la surface ou la populations sont lacunaires présentent des rectangles qui partent de l'ordonnée 0 et se dessinent vers le bas dans la partie négative puisque leur densité est égale à -1 (Figure 48).

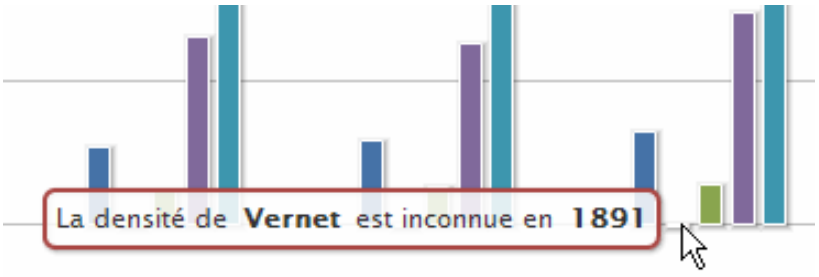

Figure 48. Bulle affichant une densité inconnue.

4.4.2. Validation

4.4.2.1. Test positif

La Figure 49 ci-dessus montre le graphique de densité calculé pour la commune de Viviez. Chaque rectangle vertical représente une densité à une date de recensement. Pour chaque recensement il y a autant de densités que de communes étudiées. Chaque commune est repérée grâce à sa couleur indiquée dans la légende. A l'affichage du graphique, seuls les 10 premiers recensements sont affichés. En effet, dans le cas où il existe plus de 20 recensements et plus de 4 communes, l'affichage des barres du graphique est condensé et n'est plus lisible. L'utilisateur peut voir le graphique dans sa globalité en cliquant en haut à droite sur le bouton « Voir tout le graphique ».

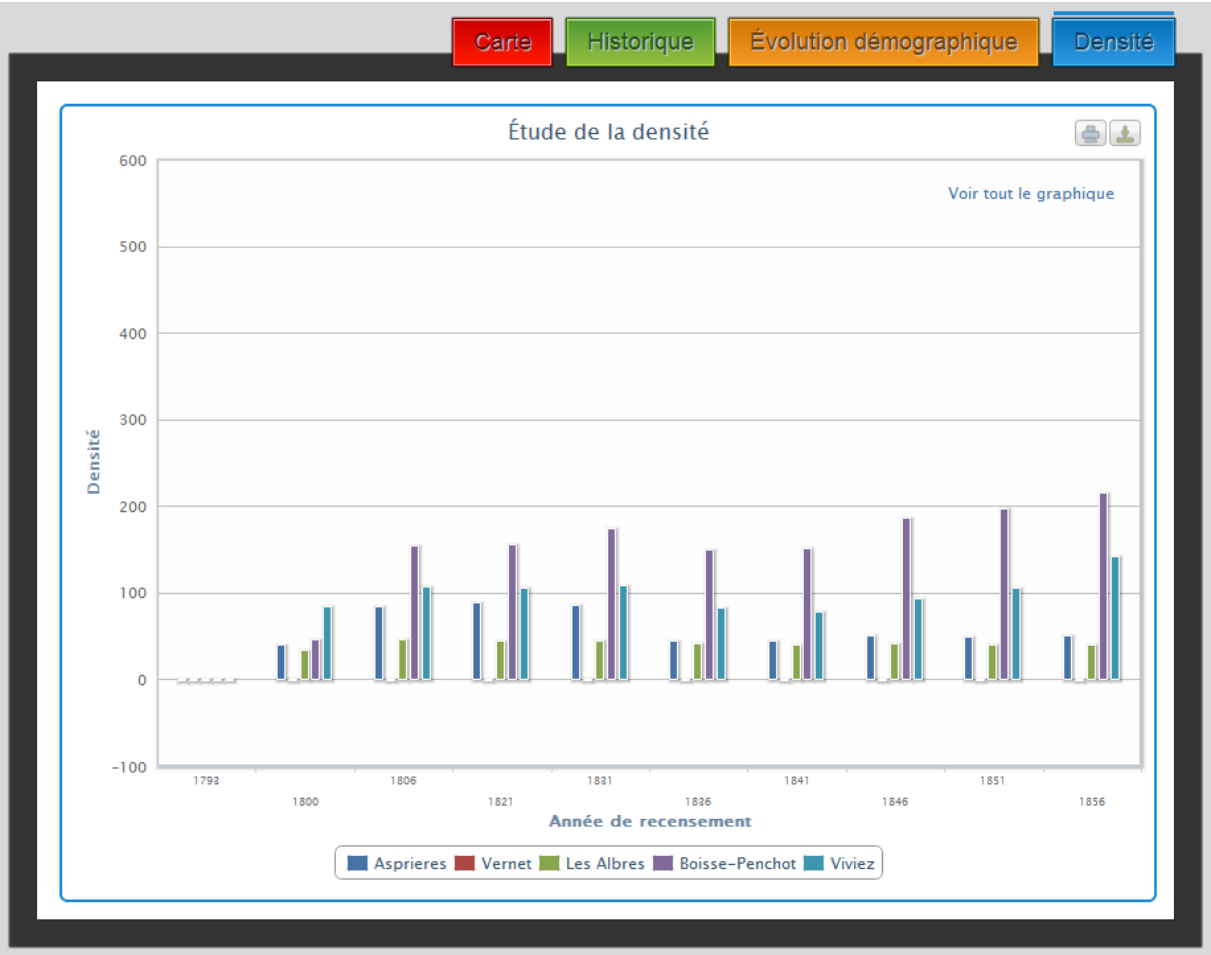

Figure 49. Résultat sur la commune de Viviez.

L'utilisateur peut, en cliquant sur le nom des communes dans la légende, faire apparaître ou disparaître les séries. En faisant disparaître des séries, les rectangles sont redimensionnés pour optimiser l'occupation de l'espace sur la largeur (Figure 50).

En cliquant sur le bouton « Voir tout le graphique », tous les recensements s'affichent à l'écran. Les recensements aux valeurs négatives sont ceux pour lesquels les surfaces ou les populations des communes sont inconnues (Figure 51).

Figure 50. Désélection de séries.

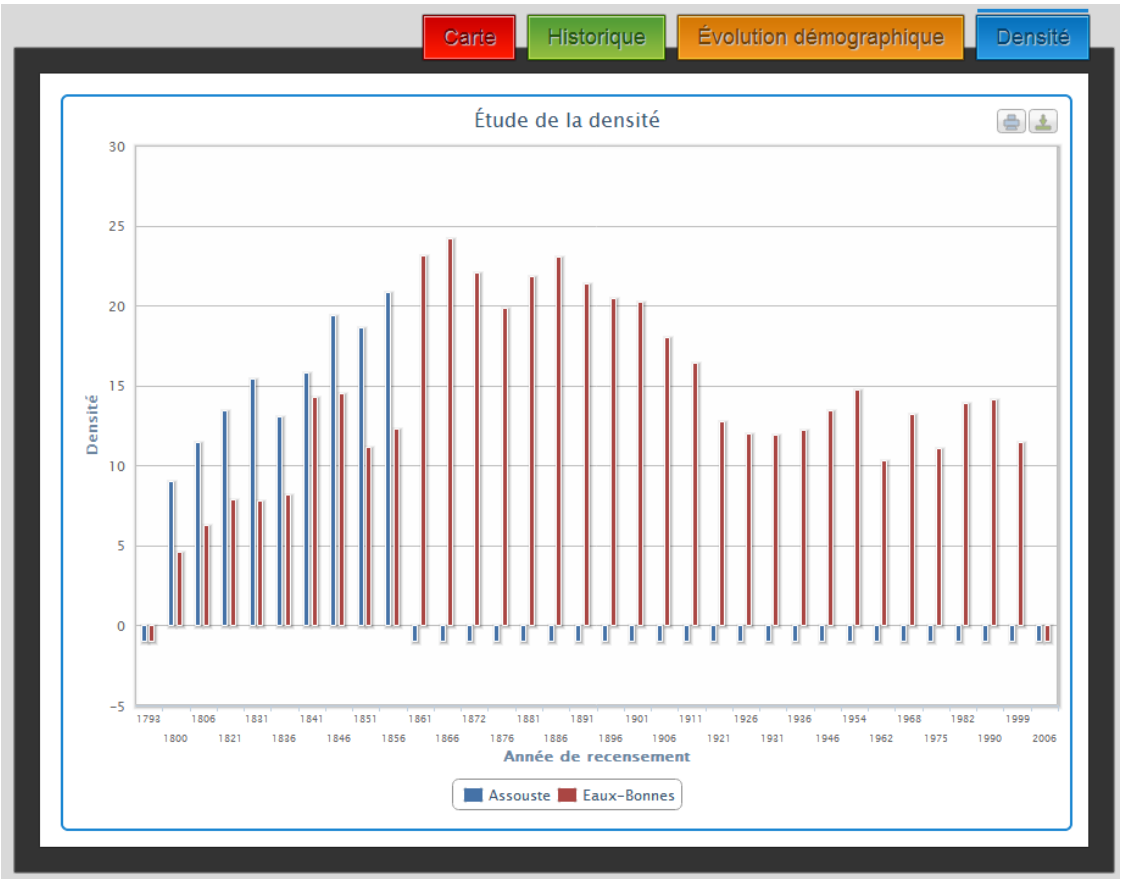

Figure 51. Graphique entier. Cas de la commune de Eaux-Bonnes.

4.4.2.2. Tests aux limites

La Figure 52 illustre un exemple avec plus de 30 recensements et 5 communes :

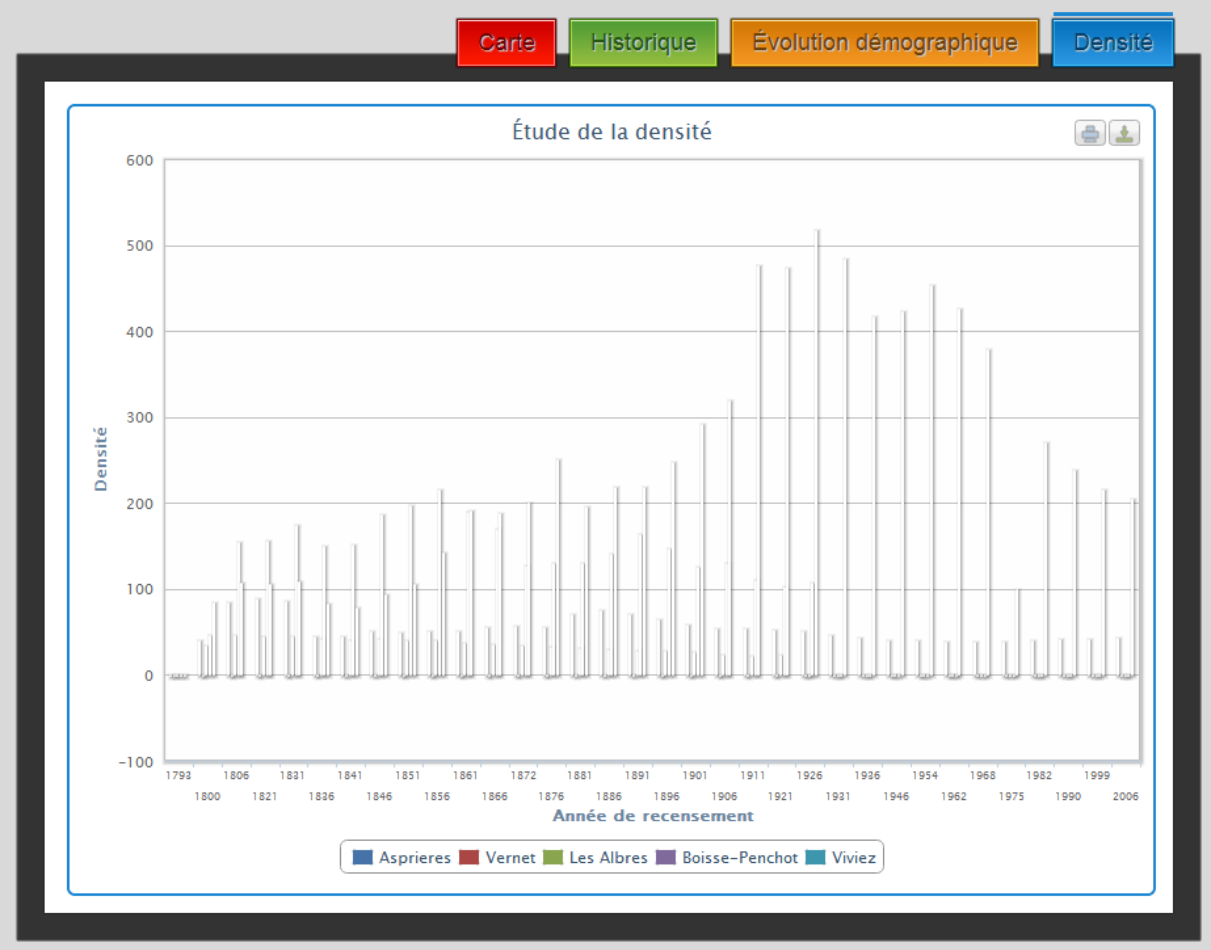

Figure 52. Graphique entier, cas de la commune de Viviez.

Le graphique entier présente 34 recensements avec 5 séries par recensements. Les rectangles sont si compactés que les couleurs ne sont plus affichées ce qui empêche d'analyser les données. Pour résoudre ce problème, l'utilisateur doit zoomer sur une portion du graphique et se limiter à visionner certains recensements à la fois ou bien désélectionner certaines séries (Figure 53).

En désélectionnant certaines communes (Figure 54), la largeur des rectangles diminue, les couleurs des séries apparaissent et le graphique devient lisible.

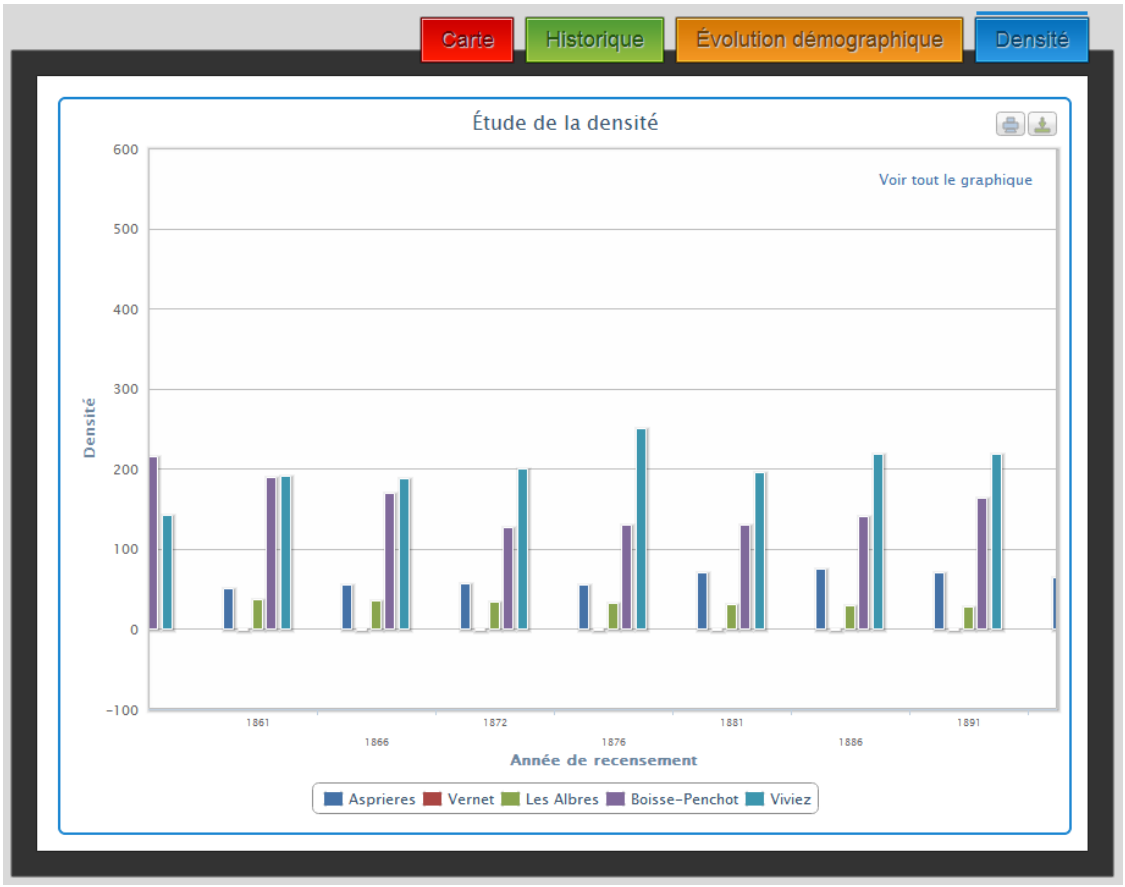

Figure 53. Zoom sur le milieu du graphique.

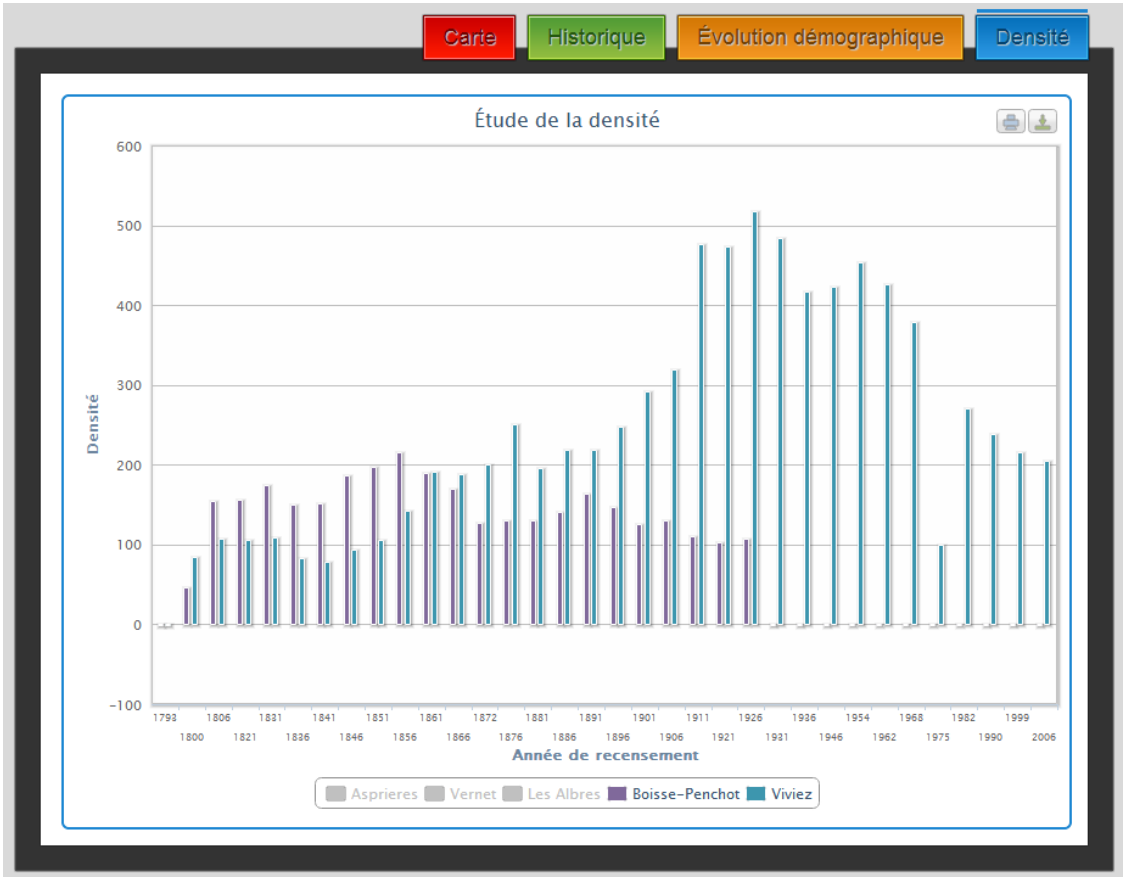

Figure 54. Désélection de communes.

5. Conclusion

5.1. Premier prototype en ligne

Les développements concernant l'exploration dynamique de l'information ont été réalisés et un premier prototype permettant de tester les différentes fonctionnalités proposées autour de 5 différentes communes a été mis en ligne à l'adresse suivante :
http://www.rotefabrik.free.fr/a543735d8de817bba30acdbdc2374596/geopeuple_web_V2/index.html

Ce premier prototype a été présenté aux différents partenaires du projet, en particulier aux historiens de l'EHESS. Les retours en termes de fonctionnalités sont très positifs. Il demeure toutefois nombre d'améliorations sur un plan esthétique à apporter, plan qui n'a pas encore été abordé jusqu'ici dans le projet.

De plus, un premier prototype fonctionnel de l'API Web RESTful a également été implémenté, mais non encore mis en ligne.

Au final, il reste quelques mois (a minima deux) pour que l'ensemble des fonctionnalités soient implémentées – incluant la page de recherche et en particulier la recherche par nom de commune, la fin de l'API et les snippets HTML correspondant côté client –, les manuels utilisateurs écrits – incluant l'utilisation du site et celui de l'API – , et la mise en ligne réalisée, cette dernière devant se faire sur le site du TGE ADONIS.

5.2. Déploiement du site Web

Sur la base de la proposition de visualisation décrite dans le paragraphe 3.3.2, l'organisation des fichiers utilisés par le site est la suivante :

Répertoire de l'application :

- **onglet.html**, conteneur HTML des onglets,
- Répertoire **js**, contient les fichiers JavaScript des bibliothèques et des scripts de l'application :
    - Répertoire **libraries**, contient les différentes bibliothèques utilisées par les scripts de l'application, *jQuery*, *jQuery UI*, *Apprise*, *Raphaël*, *Highcharts* et un fichier qui définit des surcharges de méthodes contenues dans les précédentes bibliothèques ou des méthodes natives du langage JavaScript,
    - **script_onglets.js**, le script affecté à la gestion des onglets,
    - **carte.js**, le script de l'onglet Carte,
    - **historique.js**, le script de l'onglet Historique,
    - **demographie.js**, le script de l'onglet Évolution démographique,
    - **densite.js**, le script de l'onglet Densité.
- Répertoire **json**, contient les fichiers JSON contenant les données utilisées par l'application.
- Répertoire **css**, contient les feuilles de styles appliquées au site :

- **apprise.min.css**, définit le style des fenêtres pop-up de la bibliothèque JavaScript *Apprise*,
- **jquery-ui-1.8.22.custom.css** définit le style des éléments visuels *jQuery UI*,
- **styles_onglets.css** définit le style de la page affichant les onglets,
- **carte.css** définit des règles de style spécifique au contenu de l'onglet carte,
- **webkit_scrollbar.css** définit un style amélioré pour les barres de défilement spécifiquement pour les navigateurs basés sur *Webkit* (Chromium et Safari notamment).

- Répertoire **img**, contient les fichiers images utilisées pour l'aspect graphique des onglets et des éléments *jQuery UI*.

6. Bibliographie

Andrienko N., Andrienko G., (2005), Exploratory analysis of spatial and temporal data. Springer-Verlag, 715 p.

Antoni, J-P., Klein O. et Moisy S., (2004) « Cartographie interactive et multimédia : vers une aide à la réflexion géographique », Cybergeo : European Journal of Geography. http://cybergeo.revues.org/2621

Motte C., Séguy I., Théré C. (2003) Communes d'hier, communes d'aujourd'hui. Les communes de la France métropolitaine, 1801-2001. Dictionnaire d'histoire administrative. Institut National d'Études Démographiques, 2003.

Motte C., Pélissier J.-P. (2003) Géonomenclature historique des lieux habités, Direction des archives de France, 2003.

Pison, G., Mathian, H., Plumejeaud, C., Gensel, J., (2011) Exploring world demography on line. 2011, 25ème Conférence Cartographique Internationale - Paris, France, July, 2011. http://www.ined.fr/fr/tout_savoir_population/cartes_interactives/

Plumejeaud C., (2011). Modèles et méthodes pour l'information spatio-temporelle évolutive. Thèse d'informatique. Université de Grenoble, 22 Septembre 2011. http://membres-liglab.imag.fr/plumejeaud/These/these.pdf

Plumejeaud C., Mathian, H., Gensel, J., Grasland, C., (2011), Spatio-temporal analysis of territorial changes from a multi-scale perspective, International Journal of Geographical Information Systems, 23-08-2011, DOI:10.1080/13658816.2010.534658

Plumejeaud C., (2012). *L2.3-4 Intégration des données démographiques*. Rapport du projet GéoPeuple.

Rat Patron, Agnès, (2011) Spécification de GeoNomenclature, document d'étude. Mise à jour le 2 Mars 2011.

7. Annexes

7.1. Fichier GeoJSON


```
{
 "communeList":[
  {
   "id":40860,
   "evolution":[
    {
     "an":1793,
     "dateAffichage":"1793-01-01",
     "codeInsee":"12 3 01 305",
     "motif":0,
     "nomOfficiel":"Viviers",
     "autreToponymeList":[
      "Viviez"
     ]
    },
    /* ... */
   ],
   "eventList":[
    {
     "id":7939,
     "motif":32,
     "motifGenerique":3,
     "nomMotif":"Commune absorbante de la fusion",
     "an":"1833",
     "dateAffichage":"1833-01-01",
     "commentaire":"null",
     "entitesList":[
      {

       "id":4688,
       "motif":31,
       "init":true,
       "end":false
      },
      {
       "id":40860,
       "motif":32,
       "init":true,
       "end":true
      }
     ]
    },
    /* ... */
   ],
   "demographie":[
    {
     "pop":516,
     "date":1793,
     "ordre":1,
     "estLacunaire":false,
     "codeLacune":"",
     "surface":"-1"
    },
    {
     "pop":585,
     "date":1800,
     "ordre":2,
     "estLacunaire":false,
     "codeLacune":"",
     "surface":"7"
    },
    /* ... */
   ],
```

```
    /* ... Communes suivantes ... */
   ]
  },
  polygones":{
   "BBox":{
    "type":"Polygon",
    "coordinates":[
     [
      [632546, 6379227],
      [632546, 6386124],
      [637223, 6386124],
      [637223, 6379227]
     ]
    ]
   },
   "layers":[
    {
     "id":0,
     "date_debut":"1793",
     "geometries":[
      {
       "id_entite":4688,
       "contour":{
        "type":"MultiPolygon",
        "coordinates":[[[[636647, 6385767], [636583, 6385808], /* ... */]]]
       }
      }, /* ... geometrie suivante ... */
     ]
    }
   ],
   /* ... couches suivantes ... */
  }
}
```

Figure 55. Extrait d'un fichier JSON.

7.2. Table des motifs d'événements

code	nom
1	Changement de nom
2	Création
3	Suppression
4	Changement d'appartenance
5	Transfert de centre (chef-lieu)
6	Transfert de parcelles
10	Changement de nom
11	Changement de nom dû à une fusion (simple ou association)
12	Changement de nom dû à un rétablissement
13	Changement de nom dû au changement de nom du chef-lieu
14	Changement de nom dû au transfert du chef-lieu
20	Création
21	Rétablissement
22	Commune ayant donné des parcelles pour la création
23	Commune se séparant
24	Création d'une fraction cantonale
30	Suppression
31	Fusion

32	Commune absorbante de la fusion
33	Fusion - association
34	Commune absorbante de la fusion-association
35	Fusion-association se transformant en fusion simple
36	Commune-pôle de la fusion-association qui s'est transformée en fusion simple
37	Suppression de la fraction cantonale
39	Commune de rattachement
40	Changement de région
41	Changement de département
42	Changement d'arrondissement
43	Changement de canton
50	Transfert de chef-lieu de commune
51	Transfert de chef-lieu de canton
52	Transfert de chef-lieu d'arrondissement
53	Transfert de chef-lieu de département
54	Transfert de chef-lieu de région
55	Ancien chef-lieu
56	Nouveau chef-lieu
60	Cession de parcelles avec incidence démographique
61	Cession de parcelles sans incidence démographique
62	Réception de parcelles avec incidence démographique
63	Réception de parcelles sans incidence démographique
64	Commune ancienne transférée
70	Changement de code

7.3. Table des motifs des lacunes

Code Lacune	Nom	Définition
	Pas de lacune	Information complète - pas de lacune.
0 hab.	Inhabité	Entité inhabitée.
...	Inexistence	Entité n'existant pas à cette date.
adm.	Administratif	Entité recensée avec une autre.
lac.	Lacune	Entité oubliée sur la publication du recensement.
ill.	Illisible	Information illisible.
vide	Vide	Information non disponible actuellement.
abs.	Absence	Document (ou pages) disparu(es).
NULL	Null value	L'information est inconnue, et rien au sujet de la lacune.
xxx	Autre lacune	Le code lacune n'était pas connu.