|  |
| --- |
|  |
| **Rapport d’analyse** |
|  |
| Projet de génie logiciel |

**Nom du projet : ProGPS**

19 octobre 2007

Groupe :

BERTHOMME Olivier

DUBOIS Mickaël

LERICHE Clément

VAN DROMME Julien

Rapport d’analyse

Projet de génie logiciel

# Généralités :

Nous avons choisi de réaliser ce projet de façon intuitive pour l’utilisateur et sans réaliser des fonctionnalités précises et complexes.

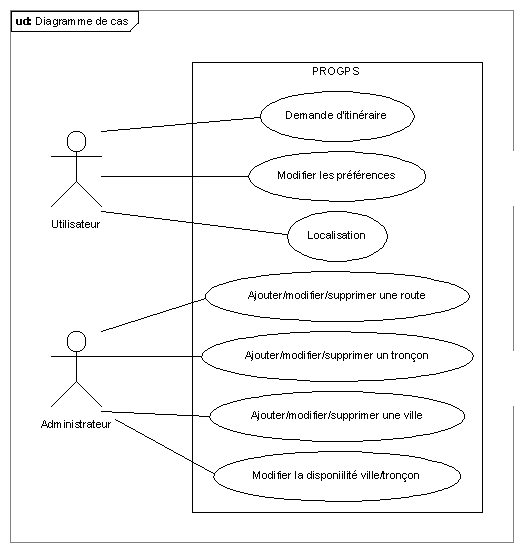
Le logiciel permettra ainsi à l’utilisateur de calculer un itinéraire entre 2 villes en respectant des préférences. L’utilisateur donne ses instructions puis valide son choix. Le logiciel lui propose alors les 5 meilleures solutions. L’utilisateur en sélectionne une et le système lui indique la prochaine ville où se rendre. Dès lors, commencent les phases de localisation durant lesquelles l’utilisateur indique au système la ville dans laquelle il se trouve, le système lui retournant la prochaine ville à atteindre.

Le logiciel possède des fonctionnalités pour l’administrateur du système. L’administrateur accède à des fonctionnalités qui lui sont réservées : ajouter/modifier/supprimer une ville/tronçon/route ; mais aussi rendre indisponible une ville ou un tronçon (ce qui permet de modéliser les changements du trafic). Ces opérations sont prises en compte par le système immédiatement. Un utilisateur peut donc être détourné d’un itinéraire pour éviter une ville ou une route.

En termes d’implémentations, nous ne mettons pas en place un système de client-serveur ni une base de données pour enregistrer le réseau routier (malgré tout, cette solution n’est pas exclue pour le moment).

# Diagrammedeclasses.pngDiagramme de classes :

# Digramme de cas d’utilisation



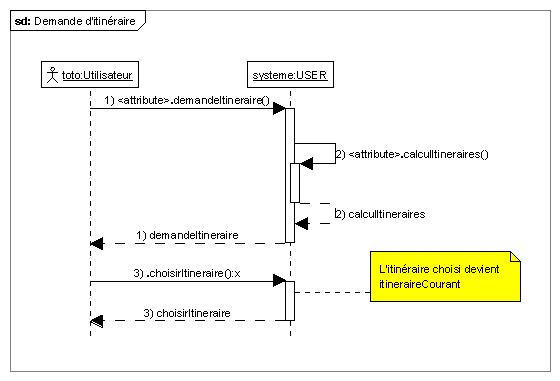
Ce diagramme regroupe toutes les fonctionnalités de notre logiciel. Certaines fonctionnalités sont disponibles pour un utilisateur, d’autres pour un administrateur.

L’utilisateur peut demander un nouvel itinéraire, être localisé (et donc obtenir la destination suivante). De plus, il peut personnaliser l’ordre des préférences utilisées lors du calcul d’itinéraire ; cela permet à l’utilisateur d’indiquer les préférences fortes et celles qui le sont moins.

L’administrateur a accès aux fonctions de modifications du réseau routier. Il peut ajouter, modifier et supprimer une route, une ville ou un tronçon.   
Il peut aussi changer la disponibilité d’un de ces éléments ; c’est à dire indiquer de façon temporaire que la ville ou le tronçon est indisponible.

# Diagrammes de séquences

## Demande d’itinéraire

Diagramme de demande d’itinéraire

**Paramètres des fonctions :**

demanderItineraire(Ville ville1, Ville ville2, Collection parametres)

calculItineraires(Ville ville1, Ville ville2, Collection parametres)

choisirItineraire (Itineraire it)

**Etape 1** : Lors d’une demande d’itinéraire, l’utilisateur choisit les deux villes de départ et d’arrivée de son itinéraire. Il choisit également ses préférences, qui sont regroupées dans une collection ordonnée selon l’ordre de ses préférences.

**Etape 2** : Le système appelle alors la fonction de calcul des itinéraires possibles entre les deux villes, et en prenant en compte des préférences de l’utilisateur : *calculItineraires()*. Elle remplit la collection *itinerairesCalcules* du système avec les itinéraires qu’elle a trouvés, triés selon l’ordre de pertinence de respect des préférences de l’utilisateur.

La fonction globale *demanderItineraire()* retourne un booléen (vrai si la demande a correctement été traitée, faux sinon).

**Etape 3** : Après cela, l’utilisateur choisit parmi les itinéraires proposés celui qu’il veut suivre. C’est ce que fait la fonction *choisirItineraire()*. L’objet *itineraireCourant* du système devient l’itinéraire que l’utilisateur a choisi. Cette fonction retourne un booléen (vrai si l’itinéraire choisi a bien été pris en compte, faux sinon).

## Localisation.jpgLocalisation

Diagramme de localisation de l’utilisateur

**Etape 1** : L’utilisateur lance la méthode de localisation qui prend en paramètre la ville dans laquelle il se trouve (*villeActuelle*).

**Etape 2** : Le système vérifie si la ville donnée par l’utilisateur est différente de la ville d’arrivée. Si ce n’est pas la cas, le système lance la méthode de fin (*finItineraire()*).

**Etape 3** : Le système vérifie si la ville dans laquelle l’utilisateur se trouve est bien la ville que le système lui avait indiqué (*villeSuivante*). Si c’est le cas, le système demande à l’itinéraire quel est la prochaine ville et actualise *villeSuivante*. Si l’utilisateur s’est trompé de destination, le système recalcule l’itinéraire et actualise la ville à suivre.

Le système indique à l’utilisateur si tout c’est bien déroulé.

## C:\Documents and Settings\Test\Bureau\Ajouterroute.pngAjouter une route

Diagramme d’ajout d’une route dans le réseau routier

L’ajout d’une route dans le système se fait par l’administrateur. Il a juste à fournir toutes les informations nécessaires à la création de la route : nom et type. Si la route existe déjà, elle ne sera pas créée.

## Modifierroute.jpgModifier une route

Diagramme de modification d’une route

**Paramètres des fonctions :**

modifierRoute(Route rt, String nom, String type)

existeDejaRoute(Route rt)

modifierInfosRoute(String nom, String type)

**Etape 1** : L’administrateur lance la méthode modifierRoute() avec les nouvelles caractéristiques de la route.

**Etape 2** : Avant de pouvoir modifier une route (donc son nom et son type), il faut vérifier qu’une route avec le même nom que son nouveau nom n’existe pas déjà, car le nom de la route est son identifiant unique. C’est ce que fait la fonction *existeDejaRoute()*, qui compare la route donnée en paramètre à toutes les routes de la collection *lesRoutes* (selon le nom). Cette fonction renvoie un booléen (vrai si il existe déjà une route qui porte ce nom, faux sinon).  
**Etape 3** : Si la fonction *existeDejaRoute()* a renvoyé faux, alors on peut modifier les informations de la route rt. C’est ce que fait la fonction *modifierInfosRoute()*.  
La fonction *modifierRoute()* renvoie elle aussi un booléen (vraie si la route a correctement été modifiée, faux sinon).

## SupprimerRoute.jpgSupprimer une route

Diagramme de suppression d’une route

**Paramètres des fonctions :**

supprimerRoute(Route rt)

vaTraverserTroncon(Troncon t, Ville villeCourante)

retirerTronconRoute(Troncon t)

retirerRouteCollection(Route rt)

rafraichirItineraire()

**Remarque** : Lorsque l’on supprime une route du système, on supprime tous les tronçons existants de cette route. Cela peut donc avoir des effets si un utilisateur utilise au même moment le système GPS et si l’itinéraire prévu traverse un ou plusieurs tronçons de la route que l’on supprime, auquel cas il faudra rafraichir l’itinéraire en tenant compte de la suppression de cette route (donc de ses tronçons).

**Etape 2** : Il faut donc avant toute chose vérifier si l’itinéraire courant que suit l’utilisateur (objet *itineraireCourant*) ne va pas devoir traverser dans le futur au moins un tronçon de la route à supprimer. C’est ce que fait la fonction *vaTraverserTroncon()*, qui pour le tronçon passé en paramètre, vérifie si celui-ci sera traversé ultérieurement par l’itinéraire. La ville courante de l’itinéraire est passée en paramètre pour pouvoir identifier où en est l’utilisateur dans son itinéraire. En effet, si celui-ci comportait un ou plusieurs tronçons de la route supprimée mais que l’utilisateur l’a (les a) déjà traversé(s), alors il n’y a aucune incidence sur le reste de son itinéraire, et celui-ci n’a pas besoin d’être recalculé.

**Etape 3** : Une fois cette vérification faite, on la garde de côté, puis on retire chaque tronçon de la route à supprimer avec la fonction *retirerTronconRoute()*. Une fois cela fait, on supprime les objets tronçons du système.

**Etape 4** : La route peut désormais être retirée elle aussi de la collection du système, et supprimée à son tour.

**Etape 5** : Enfin, si la fonction *vaTraverserTroncon()* a retourné au moins une fois vrai, alors il faut rafraichir l’itinéraire courant de l’utilisateur, ce que va faire la fonction *rafraichirItineraire()*.

La fonction globale *supprimerRoute()* retourne un booléen (vrai si la route a bien été supprimée, faux sinon).

## Ajoutertroncon.jpgAjouter un tronçon

Diagramme d’ajout d’un tronçon dans le système

**Etape 1** : L’administrateur lance la procédure avec la méthode *ajouterTronçon*() qui prend en paramètres :

* Les deux villes extrémités du tronçon qui existent forcément (contrôle IHM)
* Le nom d’une route (si cette route n’existe pas, le système la créera)
* Les attributs propres au tronçon : Vitesse, tronçon payant, avec radar, touristique, sa disponibilité actuelle et sa longueur.

**Etape 2** : Le système commence par vérifier si la route existe déjà (*existeDejaRoute*). Si la route n’existe pas, le système demande alors à l’utilisateur le type de cette route puis il la crée et l’ajoute à la collection de routes (*sesRoutes*).

**Etape 3** : Le système crée le tronçon et lance la méthode qui lie le tronçon à une route (*ajouterTronçonRoute*).

Le système renvoie alors à l’utilisateur le résultat de la procédure d’ajout (réussi ou non).

## ModifierTroncon.jpgModifier un tronçon

Diagramme de modification d’un tronçon dans le système

**Etape 1**: L’administrateur lance l’opération de modification par la méthode *modifierTroncon*() qui prend en paramètres :

* Les 2 villes extrémités du tronçon qui existent forcément (contrôle IHM)
* La route à laquelle le tronçon appartient
* Les nouvelles caractéristiques propres du tronçon (vitesse, radar etc…)
* La route à laquelle il doit appartenir

**Etape 2** : Le système recherche le tronçon en le demandant à l’objet route

**Etape 3**: Le système lance la méthode *modifierInfosTroncon()* pour que le tronçon mette à jour ses caractéristiques propres.

**Etape 4** : Si l’administrateur change la route à laquelle le tronçon appartient, le système retire le tronçon de la route (*retirerTronconRoute()*) puis lance la méthode qui fait le lien entre le tronçon et sa nouvelle route (*ajouterTronconRoute()*).

Le système indique à l’administrateur si tout c’est bien déroulé.

Remarque :

La séquence de modification du tronçon ne permet pas de modifier les 2 villes qui encadrent le tronçon. Cela pourrait être pris en compte mais il faudrait alors prévoir le re-calcul des itinéraires si l’itinéraire utilise ce tronçon.

## SupprimerTronçon.jpgSupprimer un tronçon

Diagramme de suppression d’un tronçon

**Etape 1** : L’administrateur lance la méthode supprimerTroncon() qui prend en paramètre la route du tronçon et les 2 villes extrémités.

**Etape 2** : Le système vérifie si le tronçon à supprimer ne fait pas partie de l’itinéraire actuel.

**Etape 3** : Le tronçon est délié à sa route puis supprimé du système.

**Etape 4** : Si la méthode *vaTraverserTroncon()* avait retourné vrai, on recalcule l’itinéraire.

On indique à l’administrateur que tout c’est bien déroulé.

## Ajouterville.jpgAjouter une ville

Diagramme d’ajout d’une ville

**Paramètres des fonctions**

ajouterVille(Ville v)

existeDejaVille(Ville v)

**Etape 2**: Avant d’ajouter une ville à la collection des villes du programme, il faut vérifier si celle-ci n’existe pas déjà, c’est-à-dire si le système n’a pas déjà une ville dans sa collection avec le même nom. C’est ce que fait la fonction *existeDejaVille()*, qui compare la ville donnée en paramètre à toutes les villes de la collection *sesVilles* (selon le nom, qui est l’identifiant unique d’une ville). Cette fonction renvoie un booléen (vrai si il existe déjà une ville portant ce nom, faux sinon). Si la ville n’existe pas, elle est créée et ajoutée à la collection.

La fonction *ajouterVille()* renvoie elle aussi un booléen (vrai si la ville a correctement été ajoutée, faux sinon).

## ModifierVille.jpgModifier une ville

Diagramme de modification d’une ville

**Etape 1** : L’administrateur lance la méthode modifierVille() qui prend en paramètre la ville à modifier et les informations la concernant.

**Etape 2** : Le système lance la méthode de modification des informations propre à la ville.

Le système retourne à l’administrateur si tout c’est bien déroulé.

## C:\Documents and Settings\Test\Bureau\Supprimerville.pngSupprimer une ville

Diagramme de suppression d’une ville

**Etape 1** : L’administrateur doit fournir en entrée l’identifiant de la ville qu’il veut supprimer. Celle-ci sera recensée directement dans l’IHM.

**Etape 2** : On tests si un itinéraire doit passer par la ville à supprimer.

**Etape 3** : Ensuite, tous les tronçons qui avaient pour extrémité la ville qu’on supprime seront supprimés de la route à laquelle ils appartiennent ; puis ont les supprimes du système.

**Etape 4** : On supprime la ville de la collection.

**Etape 5** : Les itinéraires sont recalculés au besoin.

## C:\Documents and Settings\Test\Bureau\Changerdispotronçon.pngChanger la disponibilité d’un tronçon

Diagramme de changement de la disponibilité d’un tronçon

L’administrateur peut choisir à tout moment de changer la disponibilité d’un tronçon. Ainsi, le booléen « disponible » de l’objet tronçon passera à *faux*, et toutes les villes liées à ce tronçon deviendront inaccessibles si elles deviennent isolées (leur état passera également à *faux)*.

L’utilisateur sera averti et pourra décider de recalculer un itinétaire.

## Changerdispoville.jpgChanger la disponibilité d’une ville

Diagramme de modification de la disponibilité d’une ville

**Etape 1** : L’administrateur lance la procédure par l’appel à la méthode *changerDispoVille()*.

**Etape 2**: Le système vérifie si l’itinéraire courant passe par cette ville.

**Etape 3** : Le programme change la disponibilité de la ville.

**Etape 4** : Si l’itinéraire devait passer par cette ville, il est recalculé.

On retourne à l’administrateur le résultat de la procédure.