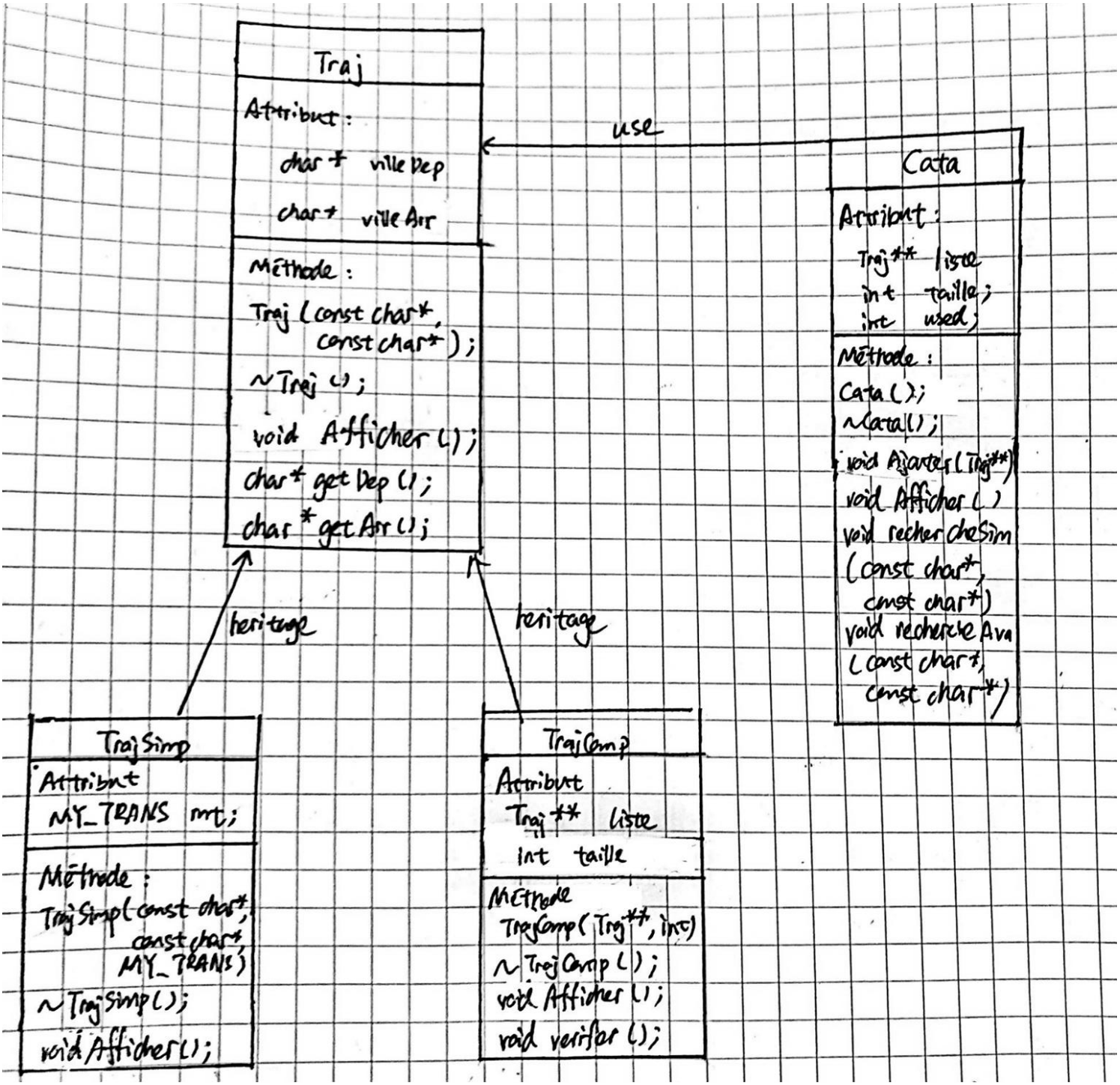
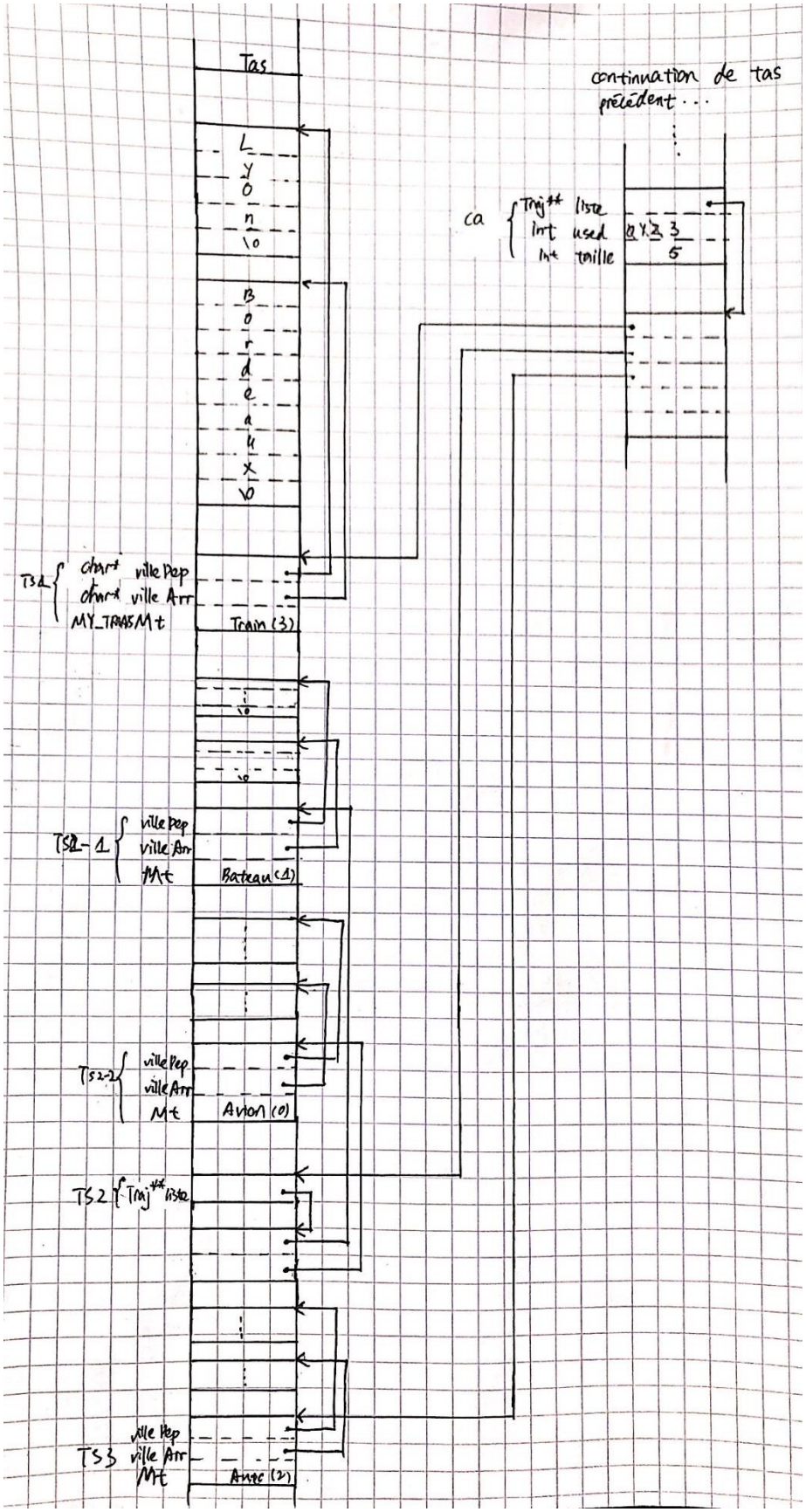
Class


Traj
Attribut :
char * villeDep
char * villeArr
Méthode :
Traj (const char*, const char*);
~Traj ();
void Afficher ();
char* getDep ();
char* getArr ();
use
Cata
Attribut :
Traj** liste
int taille;
int used;
Méthode :
Cata ();
~Cata ();
void Ajouter (Traj**)
void Afficher ()
void rechercheSim (const char*, const char*)
void rechercheAva (const char*, const char*)
heritage
heritage
TrajSimp
Attribut
MY_TRANS mrt;
Méthode :
TrajSimp (const char*, const char*, MY_TRANS)
~TrajSimp ();
void Afficher ();
TrajComp
Attribut
Traj** liste
int taille
Méthode
TrajComp (Traj**, int)
~TrajComp ();
void Afficher ();
void verifier ();

Partie Memoire



Chaque trajet, soit simple, soit composé, est dans le tas de mémoire, c'est-à-dire chaque fois après avoir saisi un trajet, on crée un new trajet dans le tas et un const pointeur dans le tas qui pointe vers le trajet. Puis le pointeur est enregistré dans le catalogue.

Listing

main.cpp	Fonction main, contenant l'interface et la vérification de la validité du trajet composé
Traj.h	Classe mère, utilisée dans la classe Cata, aussi utilisée dans la classe TrajComp
Traj.cpp	
TrajSimp.h	Classe fille de la classe Traj, représentant le trajet simple
TrajSimp.cpp	
TrajComp.h	Classe fille de la classe Traj, représentant le trajet composé, utilisant la classe Traj pour enregistrer les trajets intermédiaires
TrajComp.cpp	
Cata.h	Classe de catalogue, dont les fonctions/méthodes utilisées dans le main sont toutes dans cette classe, utilisant la classe Traj pour enregistrer les trajets différents
Cata.cpp	

Conclusion

Dans un premier temps, nous avons conçu l'héritage comme : La classe Trajcomp est une classe fille de la classe Trajsimp, en gardant l'attribut Moyen de transport privé pour la classe Trajsimp.

Après nous avons reconsidéré l'héritage en créant une classe Traj, qui est la classe mère de la classe Trajsimp et la classe Trajcomp, ce qui correspond à la conception sémantique 'Trajsimp est bien un Traj, Trajcomp est aussi un Traj'. Par contre 'Trajcomp est un Trajsimp' n'est pas correcte.

Pendant la correction des fuites de mémoire, il faut faire attention à l'appelle de la méthode de la classe mère pour éviter double suppression ou aucune suppression. Une solution prudente est de ne pas l'utiliser et réécrire dans la classe fille.

D'ailleurs il ne pas possible de passer une variable const dans une fonction dont le paramètre formel correspondant est non-const.

Selon notre conception, la vérification de la validation d'un trajet composé est faite après avoir saisi les trajets simples qui constituent le trajet composé. Par contre si le trajet composé n'est pas valide, il n'est pas dans le catalogue et donc sa destruction ne sera pas appelée. Du coup notre solution est que quand on saisit le Trajcomp, on tape la villeDep, et puis on tape seulement les villeArr de chaque trajet simple, et les Trajsimp seront générés automatiquement. Alors il n'aura jamais le problème d'un Traj non valide.

Dans la conception de la recherche avancée, on a commencé avec une idée de construire un graphe. Mais la réalisation du graphe demande beaucoup de temps, surtout cela demande également la construction de la structure de donnée ArrayList ou LinkedList (en fait pas nécessaire). D'ailleurs transformer les trajets en vertex avec une adjacent matrix ou une adjacent list et puis retransformer les vertex parcourus en trajets est redondant et non-nécessaire. En outre, le TP ne demande que la recherche des chemins possibles au lieu d'une meilleure solution, où les trajets peuvent être weighted.

Du coup on a choisi un algorithme de backtracking qui élimine tout suite les trajets dont la villeArr n'est pas une villeDep dans aucun trajet.

A partir de cette idée, on a d'abord construit une méthode renvoyant un int qui s'ajoute dans chaque return de l'algorithme récursif, est qui exprime finalement le nombre de chemins possible entre 2 villes. C'est tout à fait possible, mais difficile à débugger parce que l'addition est implicite, et puis on a rencontré pas mal de problème de segmentation fault dans la gestion de mémoire, car les conditions de return sont différentes, et influencent les recherches à partir des vertex précédents (rollback) en passant les paramètres (e.g. la liste des trajets enregistrés, l'index, la ville de rollback, la somme des chemins possibles etc.).

Ayant discuté avec nos camarades, on modifie la méthode qui est maintenant une méthode void. L'avantage est que la fin d'une feuille de recherche, soit arriver à la destination, soit ne pas avoir un trajet suivant, soit trouver un cycle (réarriver à une ville déjà visitée précédemment), est simple et implicite.