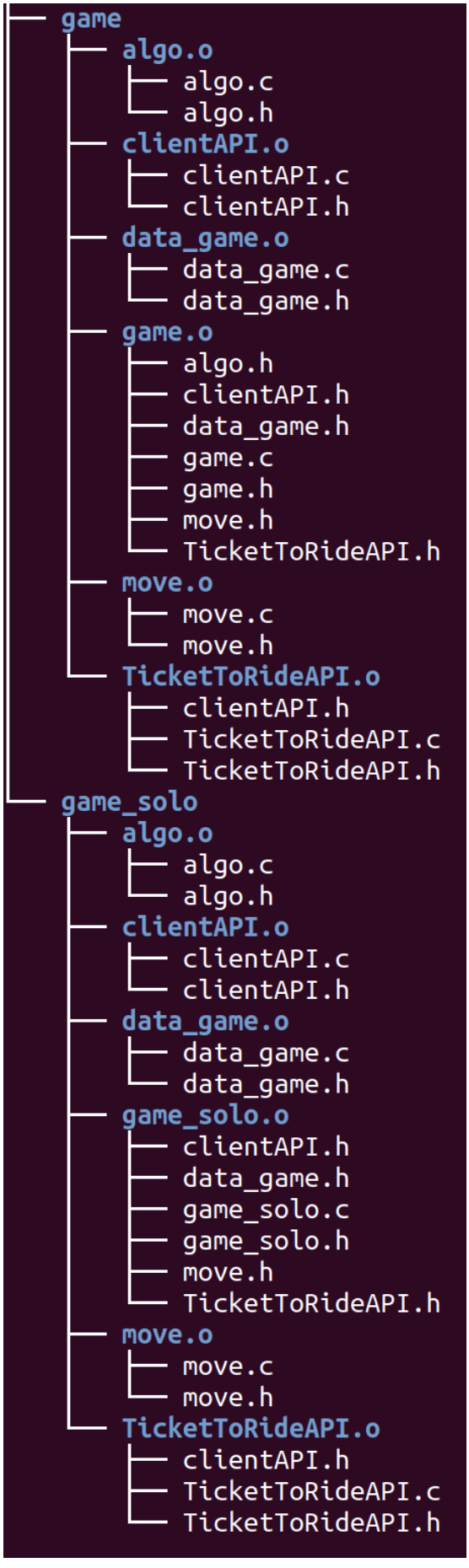
Compte rendu : Les Aventuriers du Rail



Afin d'avoir une meilleure vue d'ensemble lorsque je vais expliquer comment fonctionne mon programme, j'ai généré un arbre de dépendance en tant qu'illustration (avec la commande « *tree* » et en agençant les fichiers dans des dossiers en observant mon *Makefile,* qui devra être placé dans le même dossier que l'ensemble de mes fichiers).

J'ai distingué les 2 phases de mon projet en créant la possibilité de générer 2 exécutables **game** et **game_solo**, il suffit de taper **make game_solo** dans le terminal pour créer un exécutable **game_solo** avec lequel on pourra jouer tout seul et seulement **make** pour générer l'exécutable **game**.

Comme on peut le voir, la principale différence en terme de fichier liés est l'ajout un fichier **algo.o** pour **game** (l'exécutable qui contient le code de notre bot), celui-ci contenant les algorithmes de jeu tels que l'algorithme de *Dijkstra* ou encore une fonction récursive *maxToMin* permettant de remplir un tableau avec les couleurs des cartes qu'il nous faut par ordre de priorité (on range donc par ordre décroissant l'indice associé au nombre de cartes de chaque couleur qu'il nous faut pour remplir nos objectifs) *(cf **algo.c** lignes **80-95**)*.

Bien évidemment le contenu du *main()* du fichier **game_solo.c** a été adapté en un fichier **game.c** pour que le programme joue automatiquement. Mise à part l'utilisation d'une fonction maxToMin utilisé dans le but de choisir une carte appropriée parmi les cartes face visible, une autre partie de mon code peut paraître singulière *(cf **game.c** lignes **202** -**216**)*, celle dans laquelle je sauvegarde la liste des villes faisant parti du plus court chemin trouvé entre une source et une destination.

Afin d'accomplir cela, j'ai utilisé l'algorithme 3 du polycopié d'aide et j'ai pris en paramètre de ma fonction *printTrack,* un tableau *ShortestTrack*dans le but de remplir celui-ci avec les chemins qui sont affichés en parallèle dans la boucle while, cette fonction retourne par la même occasion le nombre de routes entre la source et la destination (je sais que l'on peut récupérer ce nombre avec *D[dest]* en suivant la consigne, mais la valeur récupéré était erronée malgré le fait que le plus court chemin soit correct) *(cf **algo.c** lignes **55-78**)*.

Quant aux fichiers **move.c** et **move.h**, je me suis grandement inspiré de ceux que vous nous aviez donnés comme exemple, en choisissant d'utiliser des switch case comme j'avais fait dans mes premiers codes, et en ajoutant en paramètre de ma fonction *PlayOurMove* un tableau d'objectifs *obj* pour sauvegarder le tableau d'objectif généré par l'appel de la fonction *drawObjectives.* J'ai utilisé cette méthode car après le choix des objectifs que l'on veut garder ou non, si on veut récupérer les bons objectifs à l'aide du tableau d'objectifs contenus dans *move->drawObjectives.objectives*, les objectifs récupérés sont faux.

```
86  /* a 'draw objectives' move */
87  typedef struct{
88      t_objective objectives[3];  /* returned ob
89  } t_drawObjectives;
90
91  /* a 'choose an objective' move */
92  typedef struct{
93      int nbObjectives;           /* returned nu
94      int chosen[3];              /* array of bo
95  } t_chooseObjectives;
96
97  /* a move */
98  typedef struct{
99      t_typeMove type;
100     union{
101         t_claimRouteMove claimRoute;
102         t_drawCard drawCard;
103         t_drawBlindCard drawBlindCard;
104         t_drawObjectives drawObjectives;
105         t_chooseObjectives chooseObjectives;
106     };
107 } t_move;
```

Donc après de long tests, l'erreur a été ciblée sur le fait que la structure *t_move* contient une union, et lorsque l'on passe t'un type à l'autre de l'union comme on le fait en générant un objectif et en en choisissant, (*t_drawObjectives -> t_chooseObjectives)* des données contenues dans le tableau d'objectifs *objectives* qui appartient à la structure *t_drawObjectives* sont « écrasées » lors de l'utilisation du tableau *chosen* qui est dans la structure *t_chooseObjectives. (cf* ***TicketToRide.h*** *lignes* ***87-107*** *ou capture d'écran à gauche).*

À noter que j'ai soigné l'ergonomie du programme afin que l'utilisateur se sente familier avec celui-ci (mais vous le verrez par vous-même ;)

Ce que j'ai trouvé extrêmement frustrant dans ce projet est le fait de ne pas le voir aboutir, je me suis acharné en y mettant beaucoup de mon temps (malgré la charge de travail conséquente dans les autres UE), mais tout ceci pour que je ne réussisse pas à faire fonctionner mon programme 😢. Mais je suis toujours motivé pour l'achever dans la semaine à venir, une aide de votre part serait alors la bienvenue 😁.

Pourquoi mon programme ne fonctionne pas ?

Le serveur me renvoie une erreur lorsque je veux posséder une route, au moment ou je la réclame j'ai l'erreur suivante : « [sendCGSMove] The data given to claim a city are incorrect » et lorsque j'ai réessayé de jouer seule, il m'arrivait (pas tout le temps) d'avoir des erreurs me disant que je n'avais pas les éléments nécessaires pour récupérer la route alors qu'en revérifiant tout était bon…

Point critique : *qu'est ce qui ne va pas ou/et pourrait être amélioré* ?

- Prendre en compte le fait qu'il faut prendre le plus de chemins courts quand on approche des derniers tours du jeu (donc en dessous d'un certain nombre de wagon restant) afin de gagner le plus de points.
- Ne pas jouer 2 fois pendant le dernier tour (afin de laisser l'adversaire piocher une carte après nous)
- Améliorer l'algorithme qui choisit les routes à prendre, afin de prendre les routes qui permettent de finir un objectif en priorité par exemple *(cf* ***game.c*** *lignes* ***107-170****).*
- Et SURTOUT résoudre le problème de prise de route.