# I/ Programme : stratégie et mise en place du jeu automatique

## ✻ *Programme*

But : jouer automatiquement une partie contre un bot ou un autre joueur.
=> objectif atteint : le programme est entièrement automatique, et gagne assez souvent contre PLAY_RANDOM ET NICE_BOT (et en tournoi)

Fichiers constituant le programme :

- connexion.h : le header commun à tous les fichiers, contient le prototype des fonctions et les définitions des structures
- connexion.c : initialise les données du jeu + contient les boucles de jeu (infinie pour jouer soit à l'infini contre un bot en mode Training, soit en tournoi ; et boucle de partie qui permet d'alterner coups de l'adversaire et nos coups) + se connecte et se déconnecte du serveur
- createDisplayGame.c : contient les fonctions permettant d'initialiser le jeu (commencer une partie, récupérer les informations de la map, créer et initialiser les joueurs) ainsi que des fonctions display permettant d'afficher objectif, route ou informations d'un joueur (fonctions obsolètes en mode automatique)
- playSmart.c : contient les fonctions permettant d'automatiser le jeu, et de décider quel(s) coup(s) jouer (pour gagner)
- Makefile : pour la compilation et l'édition de liens

Sauf erreur d'inattention, tous les fichiers et variables sont en anglais.

## ✻ *Stratégie pour tenter de gagner*

Après plusieurs parties contre les 2 bots du mode training, je me suis aperçue que le plus facile pour gagner contre ces derniers n'est pas de multiplier les objectifs, mais de prendre des grosses routes et « d'embêter » l'adversaire (surtout NICE_BOT).

Par conséquent la stratégie retenue est :

- ne piocher que 2 objectifs pour l'intégralité de la partie
- tenter de compléter en 1er celui rapportant le plus de point, puis l'autre ensuite (avec l'algorithme de Dijkstra pour trouver le plus court chemin)
- si un objectif devient impossible, l'abandonner
- à chaque tour, si on peut prendre une route aidant un objectif, on le fait
- sinon, prendre une grosse route (supérieur à 5 wagons contre PLAY_RANDOM, 4 contre NICE_BOT ; si possible soit allongeant un de nos objectifs, soit aux alentours de la route prise au tour d'avant par l'adversaire : si impossible, une « random »).
- si aucune route n'a pu être prise, piocher 2 cartes dans le deck
- enfin, à partir de 3 wagons ou moins, on baisse le nombre de wagons des routes randoms

## II/ Améliorations, problèmes etc..

### ✻ *Victoire ? Pas toujours….*

Le programme, bien que fonctionnel, est loin de gagner toutes les parties contre n'importe quel adversaire. En effet, il a été conçu pour battre les 2 bots du mode Training, donc il n'y a pas eu d'essai avant le tournoi final contre des adversaires ayant des stratégies totalement différentes. De plus, le programme peut gagner avec beaucoup de points ( > 110), mais aussi perdre assez violemment ( < 0).

### ✻ *Améliorations pensées*

- Il n'y a pour l'instant pas de re-calcul de chemin le plus court en cas de prise d'une route que l'on voulait par l'adversaire : on abandonne directement l'objectif, alors qu'il existe d'autres chemins

- On ne pioche que des cartes dans le deck (et non faces visibles), ce qui garde l'élément « chance et hasard » que l'on souhaite supprimer pour s'assurer la victoire

- Décompter le nombre de routes de chaque couleur restantes

=> par exemple si très tôt dans le jeu toutes les routes rouge ont été prises, on voudrait alors prendre des routes multicolores avec les cartes rouges restantes dans notre main

- Connecter nos 2 objectifs pour espérer avoir le chemin le plus long (fonction objectives_connected codée, qui renvoie un booléen est codée, mais pas utilisée par la suite)

- Pour poursuivre la stratégie qui consiste à « embêter » l'adversaire, récupérer et interpréter tous ses coups, pour « comprendre » ses objectifs et le gêner

- On remplit un objectif en suivant le plus court chemin en terme de nombre de wagons nécessaires, avec l'algorithme de Dijkstra : est-ce la meilleure idée ? Ou faudrait-il penser en terme de nombre de coups nécessaires ?

- Le programme a été codé pour la map USA => le rendre universel en changeant des variables (taille de tableaux …)

## Conclusion

Le programme est autonome et relativement « smart » donc ces deux points sont une réussite ☺. Je n'aurais cependant pas refusé un peu plus de temps pour perfectionner stratégie et implémentation. De plus, mon code est selon moi peu « esthétique » : beaucoup de if différents qui ont presque les mêmes conditions et conduisent presque au même résultat, mais séparés.

Annotation post-tournoi : En tournoi, mon bot se défend plutôt bien et ne produit pas d'erreurs. On remarque que la stratégie est bonne, avec les défauts cités plus hauts (ex: avec le bot BOT_PLLLL assez fort, 1ère phase en sa faveur [5,4], mais 2nde phase en sa faveur [5,0]….) => une part encore trop importante de hasard.