LES AVENTURIERS DU RAIL

TP

On se divise en quatre groupes : chacun travaille pendant la première heure sur l'une des parties. La deuxième heure, chaque groupe présente au tableau, pendant un quart d'heure, ce qu'il a fait (en expliquant, évidemment !). Pour cela on déposera sur le répertoire de partage `S:` les codes, de sorte à ce qu'ils soient vidéo-projetés.

Partie I — L'ensemble des villes

On impose une carte de dimensions `LARGEUR` × `HAUTEUR`, rangées dans des variables globales, ainsi qu'une quantité `DISTANCE_MINIMALE` qui définit la distance minimale entre deux villes.

On souhaite générer aléatoirement un certain `NOMBRE_DE_VILLES` (encore une variable globale). Pour chacune d'elles, on choisit des coordonnées aléatoires (en les positionnant suffisamment loin les unes des autres), et un nom aléatoire. On s'efforcera de produire des noms crédibles. Le résultat se présente sous la forme d'une liste de triplets `(`x`, `y`, nom)`.

Enfin s'il reste du temps, on écrira un programme qui permet de déterminer, étant données trois villes A, B et C, si C est « à peu près » sur le segment [A, B]. Au moment de créer les voies ferrées, on imposera lorsque la situation se produit de passer par C plutôt que d'aller directement de A à B, pour éviter les chevauchements de voies.

Partie II — Les chemins de fer

En plus des variables globales précédentes, on introduira ici les quantités `DISTANCE_MAXIMALE`, `TAILLE_BLOC` et `NOMBRE_DE_ROUTES`. Un chemin de fer entre deux villes doit être constitué d'un nombre de blocs compris entre 1 et 6, et tous les blocs doivent avoir la même dimension. La distance entre deux villes doit donc être au moins d'environ un bloc, et (si on choisit de les relier) d'au plus environ six blocs.

On aura besoin d'un programme qui détermine si deux segments se coupent. Les arêtes sont générées aléatoirement, et elles ne doivent pas se couper. Le graphe doit être connexe (toute ville est reliée au réseau de chemins de fer).

On renverra le résultat sous la forme d'une liste de quadruplets `(`i`, `j`, `p`, `c`)` où i et j sont les numéros des villes reliées (on suppose la liste de villes construites), p est le nombre de blocs qui les séparent, et c est la couleur (commune) de ces blocs.

Partie III — Rendu graphique

On suppose connues la liste des villes et la liste des routes. On réalise, avec `matplotlib.pyplot`, la représentation graphique de la carte, avec des points pour les routes, et des rectangles pour les blocs.

On s'intéressera à la fonction `text` (pour afficher les noms des villes), en particulier à ses paramètres facultatifs `horiztonalalignment` et `verticalalignment`, et à la fonction `fill` (pour colorier les rectangles).

Un aspect central est la création d'un programme qui dessine un rectangle, de dimensions fixes, à une certaine position et dans une certaine direction (si on relie les villes A et B, les blocs entre elles doivent avoir des côtés parallèles ou perpendiculaires à la droite (A, B)). On gérera des petits espaces entre les blocs.

Partie IV — Les missions

Une *mission* est la donnée de deux villes à relier. Elle est associée à un nombre de points, correspondant à la distance minimale (en nombre de blocs) pour relier les deux villes.

Une contraite : le nombre de missions ayant pour extremité une ville donnée ne doit pas excéder la valence de cette ville.

On s'intéressera à l'algorithme de Roy, Floyd et Warshall pour calculer la distance entre chaque paire de villes. On suppose évidemment la liste des villes et la liste des routes connues.

Partie V — Initialisation d'une partie et configurations du jeu

Une partie, en plus de la donnée de la liste des villes, de la liste des chemins de fer et de la liste des missions, est la donnée d'une pile de cartes et d'une liste de joueurs (dans l'ordre où ils joueront).

Pour les cartes, on suppose connu l'ensemble des couleurs, le nombre de cartes de chaque couleur et le nombre de jokers multicolores.

Une *configuration* du jeu est la donnée :

- pour le plateau, d'une copie de la liste des chemins de fer, chacun étant étiqueté par l'information `None` (personne n'a posé de train) ou `J` (pour indiquer que le joueur d'indice `J` y a posé ses trains).
- pour chaque joueur de la liste des missions et la liste des cartes de couleur qu'il a en main, ainsi que le nombre de trains qu'il lui reste,
- la liste des missions encore à piocher,
- cinq cartes de couleurs « visibles » et la pile des autres.

On écrira les programmes qui fabriquent la configuration initiale, et le programme qui teste si la partie est terminée.

On écrira également le programme qui « retourne la pioche », c'est-à-dire, lorsque la pile est épuisée, brasse la défosse et en fait une nouvelle pile. Ainsi que le programme qui remplace les cartes visibles si trois jokers y apparaissent.

Partie VI — Simulation d'un coup

Il y a trois sortes de coup : poser des trains, piocher des cartes couleurs et piocher des missions.

Les coups seront modélisés par des uplets ; la première composante sera une constante parmi les trois valeurs (définies dans des variables globales) `PIOCHE_COULEURS`, `PIOCHE_MISSIONS`, `POSE_TRAINS`.

Dans le premier cas, la deuxième composante représente la liste (de taille 1 ou 2) des numéros des cartes piochées (six valeurs possibles, zéro représentant le tas et les nombres de 1 à 5 les cartes visibles).

Dans le deuxième cas, la deuxième composante représente la liste (de taille 1, 2 ou 3) des missions conservées, représentées par leur numéro dans la liste des missions.

Dans le troisième cas, la deuxième composante sera le numéro de l'arête (dans la liste des chemins de fer) sur laquelle le joueur pose ses trains, la troisième composante représentera la couleur des trains posés, et la quatrième composante le nombre de jokers utilisés.

Partie VII — Faire jouer l'ordinateur

On pourra explorer trois pistes : le jeu au hasard, le jeu passif (où l'on cherche simplement à remplir ses missions, avant d'en piocher d'autres quand on les a achevées), et enfin le jeu agressif, où l'on cherche simplement à finir le plus vite possible des missions en barrant un maximum de routes pour les adversaires).

Dans tous les cas on a besoin du programme qui étant donnée une configuration calcule la liste des coups possibles.

Partie VIII — Calcul des scores

Écrire le programme qui étant donnée une configuration du jeu (la configuration finale en particulier) calcule le score de chaque joueur. Il faut compter les trains posés, compter la route la plus longue, et enfin vérifier l'achèvement de chaque mission.

Pour calculer la route la plus longue, on s'intéressera au sous-graphe formé par les trains posés d'un joueur, et on écrira un programme qui dresse la liste des routes à partir d'une ville donnée, et ne passant pas deux fois par la même arête (on pourra représenter le résultat par un arbre). Puis on remarquera que la route la plus longue, si ce n'est pas un cycle, commence et termine forcément par des villes de valence impaire.