Qwirkle

Travail en algorithmie - Planchon & Marzook

I. Fonctionnement des règles du jeu

Dans le but d'établir les règles du jeu nous avons procedé en plusieurs étapes.

I.1. Placer un pion

Pour placer un pion, nous avons divisé le problème en plusieurs sous-parties.
Tout d'abord nous avons établie une fonction qui comptabilise le nombre de voisins sur toute une ligne puis sur toute une colonne :

```
1 FUNCTION calculNombreDeVoisin(grille; x,y, dirInt:INTEGER)
```

Pour pouvoir calculer en colonne ou en ligne, nous attribuons une direction (gauche, droite, en haut, en bas). Pour faire toute la ligne on additionne gauche et droite et de même pour la colonne.

Ensuite, après avoir compté le nombre de voisins en ligne et colonne nous avons établie l'état des lignes et colonnes à l'aide la fonction :

```
1 FUNCTION findEtat(g : grille; x,y, dirInt : INTEGER)
```

C'est-à-dire que nous déterminons si la ligne ou la colonne (en fonction de la direction choisie) de pions formée est en couleur ou en forme, et l'on précise bien sûr la couleur et la forme en question.
Puis, nous établissons à l'aide de l'état de la ligne et de la colonne, les concordances des cases et donc si nous pouvons placer le pion ou non.
Cependant, il ne fallait pas oublier de vérifier si le pion que nous voulons placer est déjà présent en ligne ou en colonne, ou encore si si nous ne relions pas une ligne ou une colonne contenant le même pion de part et d'autre.
Pour cela nous avons penser à créer une fonction injective qui permettrai d'attribuer a chaque pion un entier entre 0 et $\frac{2n*(2n+1)}{2} + n$, avec n le nombre de formes (en supposant qu'il y a autant de forme que de couleurs).

```
1 FUNCTION injection(x, y : INTEGER) : INTEGER;
2 BEGIN
3  injection := ((x + y) * (x + y + 1)) DIV 2 + y;
4 END;
```

De ce fait, cela devient plus simple de vérifier si un pion est dupliqué car nous n'avons pas a comparer la couleur puis la forme et ceux plusieurs fois pour les lignes et colonnes et le pion que l'on veut placer mais seulement à comparer l'image du pion par cette fonction injective par rapport aux autres en plaçant chaque pion dans un tableau (une map comme en c++ aurait été plus optimisé).

I.2. Placer plusieurs pions

Pour placer plusieurs pions il ne faut désormais rajouter seulement quelques règles car les règles pour placer un pion sont déjà établies. Nous avons donc fait face à plusieurs contraintes :
- il faut que les pions placés après le premier soient de la même couleur que celui-ci ou même forme,
Pour cela nous avons procéder de la manière suivante : vérifier si le 2ème est de la même forme ou même couleur puis pour les autres vérifier ces conditions par rapport au premier et au deuxième.
- il faut aussi que les pions soient placés en ligne ou colonne sans discontinuité.
Pour réaliser cela nous avons créé des fonctions vérifiants l'état des pions par rapport aux précédents et la continuité (tous les pions doivent être placé de façon continue sur la même ligne / colonne).

I.3. Comptage des points

Pour compter les points nous avons fait une disjonction des cas. En effet le comptage des points depends du nombre de pions que nous plaçons.
Si nous plaçons un pion il faut juste compter combien de voisins il possède en ajoutant un autre point pour chaque ligne ou colonne a laquelle il appartient.
Cependant si nous placons plusieurs pions il faut compter les points pour chaque pion placé puis ensuite les addtionner. Mais il faut aussi compter le nombre de point qu'on engendré les pions placés en ligne ou en colonne.
Sans oublier qu'il faut rajouter un bonus lorsque l'on forme un qwirkle.

II. Intelligence artificielle par arbre décisionnel

L'itelligence artificielle était un défi. Nous ne voulions pas avoir 15 boucles, nous ne voulions pas une complexité absurde. Nous nous sommes donc penché vers la méthode des arbres décisionnels. L'idée est que l'IA se créer un arbre des différents coups qu'elle peut jouer.

Elle commence par balayer la grille entière (la seule foi) pour trouver toutes les position où un coup peut être joué (avec les pions de la main). Cela constitue la première profondeur de l'arbre. Depuis ces différentes branches, elle regarde dans les 4 directions (en allant au maximum, elle ne fait pas que ses voisins proches) si elle peut jouer un pion, si elle le peut, elle créé une branche et recommence de cette branche.

Nous avons opté pour une méthode récursive pour la création de l'arbre. Cela semblait évident. Une fois l'arbre fini, elle calcul les score. La branche avec le meilleur score est ensuite choisis. L'IA a trouvé quel coup jouer. Cet algorithme est assez optimal. Il donne toujours la meilleur solution possible pour la main et la grille dont elle dispose.

Cependant, cet IA n'est pas capable de jouer dans le futur et de faire de la prédiction. Les algorithmes que nous aurions dut implémenter était trop compliqué. Et ils n'auraient pas été très performant étant donné le nombre de coups jouables au tour prochain (au max n^{n^2}). L'arbre étant assez gourmand en mémoire nous le déalouons à la fin de chaque tour.

III. Utilisation de github

Nous avons utilisé un outil de gestionnaire de fichier : github. Il fut très utile car il nous a permis de pouvoir travailler à plusieurs sur les mêmes fichiers en même temps. Il nous a aussi permis de corriger des erreurs que nous avons fait, en nous permettant d'avoir accès aux vielles version. C'est avec ce projet que nous avons compris la vraie puissance de GitHub avec ses branches. Rien que pour çà ce projet fût enrichissant.

IV. Conclusion

Ce projet s'est révélé très enrichissant. Il nous a permis de mettre en pratique toutes les connaissances que nous avons acquises depuis le début de ce semestre. Nous avons pu developper notre capacité à travailler en groupe et à écouter les idées de chacun afin de pouvoir avancer plus vite ensemble. La cohésion de groupe a été très importante et nous a permis de communiquer et de comprendre plus rapidement. Finalement, ce projet nous a permis de grandir en algorithmique d'augmenter notre niveau pratique et de pouvoir résoudre une plus grande diversité de problèmes.