Rapport projet MENACE

(Structures de données)

I) Représentation d'un terrain

Afin de représenter une configuration de grille de morpion dans le programme, un simple tableau a deux dimensions de taille 3x3 suffirait. Mais pour l'algorithme MENACE, qui nécessiterait de représenter toutes les configurations possibles de grilles, cette méthode est peu efficace car peu compacte en mémoire.

Pour représenter alors un terrain, nous aurions pu le représenter sous forme d'un entier, qui une fois converti en base 3, nous donnerait la valeur de chacune des 9 cases d'un terrain (0 pour vide, 1 pour une croix et 2 pour un cercle par exemple).

Cependant, nous avons décidé de représenter un terrain sous la forme d'un entier, codé sur 18 bits, qui une fois converti en binaire nous donnerait la valeur des cases du terrain. Une seule case est donc codée sur 2 bits,qui nous permettrait de représenter 4 états possibles de la case mais nous en avons besoin que de 3. Ce choix a été réalisé de manière à grandement simplifier les opérations réalisées sur l'entier représentant un terrain, à l'aide des opérateurs de décalage bit par bit « >> » et « << », ces dernier remplaçant les boucles.

II) Représentation des billes

A chaque case d'une configuration de grille est attribué un nombre qui représente la probabilité de l'IA de jouer sur cette case. Le but de l'IA est de « raffiner » ces billes pour réduire la probabilité de jouer des coups qui mèneront à une défaite et augmenter celles des coups qui mèneront à une victoire.

Pour notre programme un seul entier est attribué à un terrain pour représenter les billes. Cet entier est codé sur 64 bits, dont les 7 premiers bits de poids faible représentent les billes de la 1ère case, les 7 bits suivants à celles de la case 2, et ainsi de suite jusqu'à la 9ème case. Une case peut donc avoir au maximum 127 billes.

III) Mise en place de la structure

Dans cet algorithme, il faut stocker tous les terrains possibles qu'on peut rencontrer lors d'une partie, pour permettre à l'IA de retenir toutes les billes de chaque terrain et de les modifier à souhait. Pour cela,une structure de données reliant les terrains (ou les boîtes) entre eux est nécessaire.

a) Arbre

La structure à adopter a été le choix le plus difficile car cela influence sur toute la méthodologie du programme. En premier abord, nous avons décidé de réaliser un arbre. Chaque nœud de l'arbre est une boîte à laquelle sera associé un terrain possible et un nombre de billes.Un nœud a alors un unique père(sauf le terrain vide qui est la racine de l'arbre) et un ou plusieurs fils,dont les terrains sont ceux obtenus en ayant joué un coup à partir de la boîte d'avant. Une boîte dont le terrain est une grille pleine ne peut donc pas avoir de fils.

Cependant ,cette structure pose problème au niveau du tri entre les configurations. En effet, nous souhaitions qu'a chaque nœud soit associé un état de terrain, dont tous les terrains correspondant à une rotation ou une symétrie de ce terrain sont associés à cet état de terrain.//a finir

b) Tableau

En deuxième lieu nous avons envisagé de construire un tableau où l'on stockerai tous les nœud. Afin d'éliminer les rotations et symétries il nous fallait une fonction qui aurait permis de trouver l'indice de l'état du plateau mais que cet indice ne soit pas changer par une rotation ou une symétrie du terrain. Après avoir réfléchie nous avons ne pas pouvoir trouver une fonction satisfaisant ces critères dans un délais raisonnable.

c) Graphe

Enfin,nous avons finalement décidé de reprendre la structure de l'arbre en la modifiant, afin d'obtenir un graphe.

Chaque nœud de la structure (voir figure 1) contient les mêmes informations que la structure de l'arbre : une boîte, c'est-à-dire un entier représentant toutes les billes d'un terrain, et un entier représentant un terrain possible associé à la boîte. Cependant,ces nœuds sont reliés d'une manière un peu différente :
Quand plusieurs terrains sont équivalents après des rotations ou symétries , ils seront tous représenté par un même terrain. Donc,un nœud peut avoir plusieurs pères.

```
20  typedef struct _boite
21  {
22    int32_t terrain;
23    int64_t bille;
24    struct _boite **suivants;
25  } boite;
```

Figure 1 : structure d'une boîte

Cette structure semblait pratique pour l'utiliser dans le moteur de jeu du programme, car il est facile de se déplacer de nœud en nœud au fur et à mesure d'une partie et de suivre un chemin dans le graphe pour mettre les billes à jour. Mais il s'est avéré qu'elle a imposé des complications pour les opérations concernant le graphe entier (c'est-à-dire pour le charger depuis un ficher, le sauvegarder, voir le libérer de la mémoire).

En effet, si deux nœuds(ou plus) peuvent avoir un fils commun, le parcours de l'arbre en récursivité pose comme problème de passer deux fois (ou plus) par le même nœud, et donc de sauvegarder, charger ou libérer plusieurs fois ce nœud. Il a donc fallu sauvegarder les adresses des nœuds par lesquels on est déjà passé par récursivité dans un tableau transmis en paramètre des fonctions récursives. Ce qui n'est pas très intéressant en terme d'espace utiliser (presque 2 fois le graphe) ni en terme de calcule (on doit parcourir le tableau a chaque fois que l'on veut agir sur un nœud.

III) Problèmes rencontrés

A chaque structure énoncé si dessus nous avons commencé à développé une partie du code si qui nous à fait avoir (en plus de prendre du retard) un code très hétérogène qui lorsque nous avons dut l'assemblé à partir des différentes fonctions, il fonctionnait mal.

De plus la structure choisie est trop complexe pour réussir un débogage correct avec le retard pris à choisir la structure. On aurais sûrement dus faire un arbre et stocker toutes les symétries et chercher à les mettre à jour plutôt que de les supprimer.

La structure étant complexe, la plupart des problèmes rencontrés n'était pas des erreurs de compilation ou des erreurs « grave » (segmentation fault, …) mais des erreurs dans le déroulement du programme avec des comportements qui ne correspondait pas à ce qui était attendu.

Des exemple d'erreur:

-une expression du style 1<<63 renverra 0 car elle est stockée sur 32bits même si on écrit int64_t nb = 1<<63 ;

-Une autre erreur que nous avons mis longtemps a trouver est que nous ne pouvons pas afficher les terrains correctement car on ne les stocke pas. Donc d'un terrain à l'autre, les terrains affichés seront des symétries ou des rotations l'un de l'autre.

Malheureusement nous n'avons pas eu le temps de trouvé toutes les erreurs qui doivent encore être nombreuses dans notre codes.