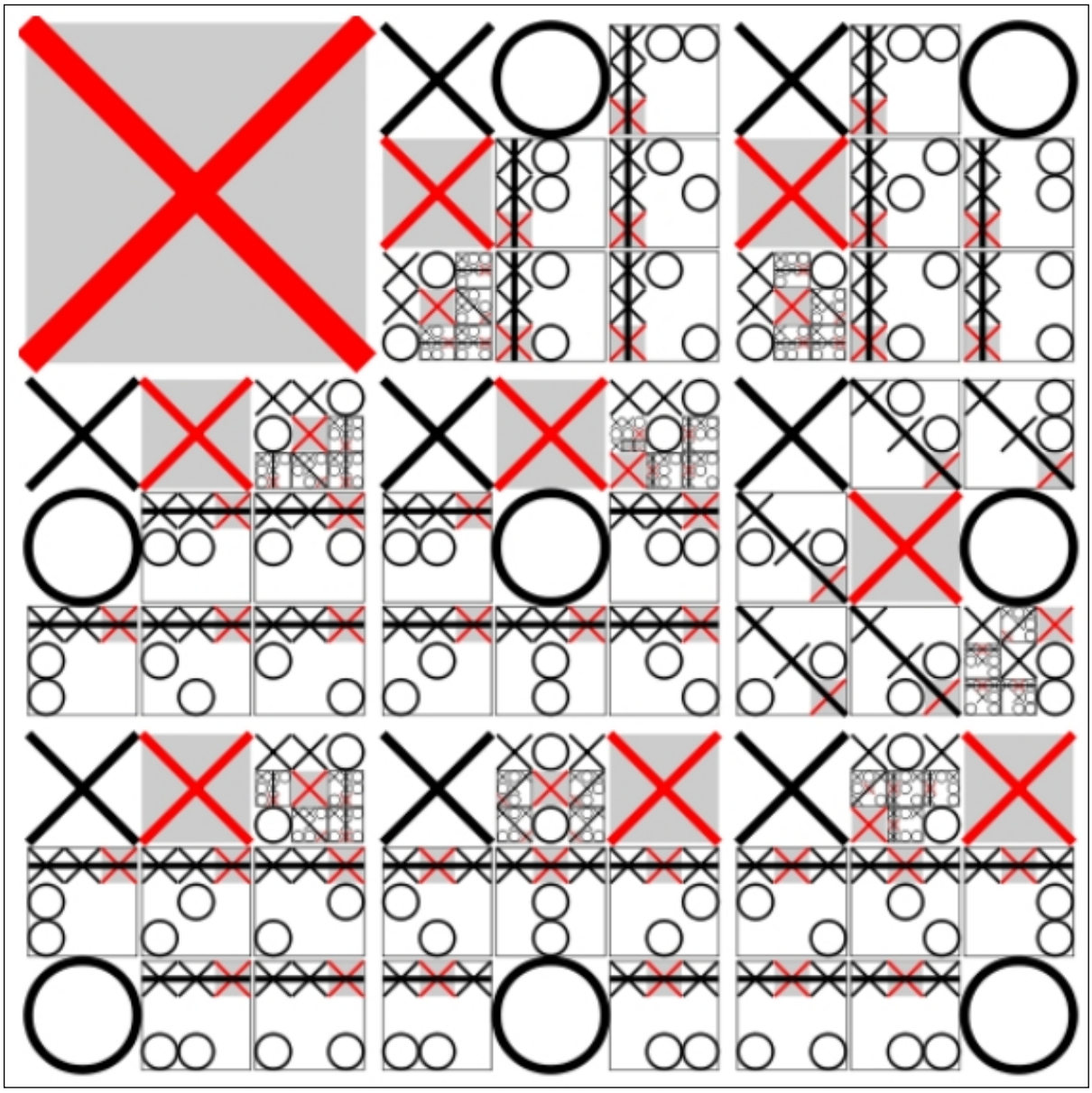
Morpion MENACE

Mohammed EL KHAIRA

2017 / 2018



Introduction

Nous allons programmer un jeu du Morpion dans lequel la machine qui joue est munie d'une certaine "intelligence". En effet, elle doit être capable d'apprendre à jouer seule, à force de répétitions et de renforcements. Plus précisément, nous avons implémenté **deux** modes : dans le premier, la machine joue contre l'utilisateur et apprend à jouer au fur et à mesure des parties ; dans le second, la machine joue un certain nombre de parties contre elle-même pour s'entraîner puis défie l'utilisateur (ce nombre est choisi par l'utilisateur).

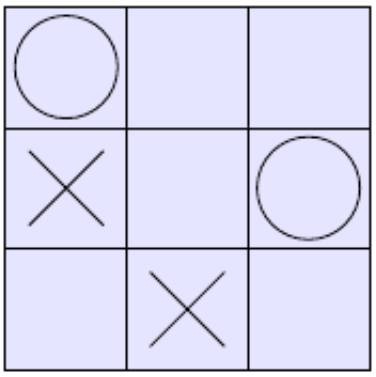
Pour implémenter l'IA de notre Morpion, nous allons nous baser sur les travaux de Donald Michie et de sa "machine" MENACE.

Représentation des grilles

Tout d'abord, nous avons dû déterminer l'ensemble des configurations possibles pour une grille de Morpion. La grille comporte 9 cases et chaque case est occupée soit par une X, soit par un O, soit par un vide. L'ensemble des configurations possibles est donc **3^9 = 19683**.
Il faut cependant noter que dans ces 19683 configurations de grille possibles en théorie, certaines ne sont **pas** valides lors d'une partie de Morpion (par exemple les grilles remplies uniquement de X ou de O).

Ensuite, nous avons déterminé une manière de représenter nos grilles de Morpion. Comme suggéré par l'énoncé du sujet, nous avons choisi d'identifier chaque grille par un numéro, un **id** qui est calculé à partir de **coefficients** propres au contenu des cases et en faisant intervenir la base numérique **3**, en fonction du **numéro** des cases. En effet, les cases sont numérotées de **1 à 9** (de gauche à droite et du haut vers le bas) et sont munies d'un coefficient qui varie selon la nature de leur contenu : **vide = 0, X = 1 et O = 2**. Nous pouvons alors attribuer à chaque case un score de la manière suivante : **score = coefficient * 3 ^ numéro**.
Finalement, pour obtenir l'id d'une grille, il suffit de faire la **somme** des scores des 9 cases.

Voici un exemple :



id = 2 * 3^1 + 1 * 3^4 + 2 * 3^6 + 1 * 3^8 = **8106**

Avec cette méthode de calcul, l'id maximum est 59046 (grille théorique remplie de O).

Représentation de l'ensemble des grilles

Maintenant que nous avons choisi un modèle de représentation des grilles, il reste à déterminer comment représenter notre ensemble de grilles. En effet, au fur et à mesure que les parties se succèdent, chaque grille différente rencontrée doit être stockée en mémoire car chaque grille jouée doit voir son nombre total de billes modifié à la fin de chaque partie.

Pour cela, nous avons décidé d'utiliser un **tableau statique de 3^9 cases** pour stocker les grilles **en fonction de leur id**. En effet, avec ce tableau, nous sommes sûr de pouvoir stocker toutes les configurations de grilles rencontrées lors des différentes parties. Malheureusement, certaines cases resteront vides car comme nous l'avons dit précédemment, certaines des 3^9 configurations théoriques sont invalides dans le jeu du Morpion.

Aussi, nous nous sommes rendu compte qu'avec notre méthode de calcul d'id de grille, beaucoup d'id seront supérieurs à 3^9 - 1. Nous allons donc avoir un **premier problème** pour stocker les grilles dans un tableau dont les indices vont de 0 à 3^9 - 1. Pour remédier à ce problème, nous allons utiliser une **sorte de fonction de hachage** qui à partir d'un id donné renvoie un indice compris entre 0 et 3^9 ; en fait, cette fonction est assez simple, elle renvoie **id = id** % **3^9**.

Cependant, un **second problème** pointe le bout de son nez. En effet, avec notre méthode de calcul, nous avons remarqué que les id seront forcément des **multiples de 3**. Cela va nous poser un problème à cause de notre fonction de hachage car nous allons avoir 3 grilles stockées dans les cases dont les indices sont des multiples de 3 (et l'indice 0 aussi).

Par exemple si nous avons des grilles dont les id valent 3, 19686 et 39369 alors le problème est clair : **3 % 3^9 = 19686 % 3^9 = 39369 % 3^9 = 3**.

De plus, si les id sont uniquement des multiples de 3, cela signifie que pour une case de notre tableau remplie, nous allons avoir deux cases vides qui la suivent.

Pour résoudre ce second problème, nous aurions pu stocker nos grilles dans une table de hachage où les listes sont composées de 3 grilles. Mais dans ce cas, nous aurions eu deux listes vides pour une liste de 3 éléments, en plus des cases vides dû aux grilles invalides.

Nous avons donc préféré faire **d'une pierre deux coups** : nous avons modifié notre fonction de hachage afin qu'elle renvoie :

- **id % 3^9 si id < 3^9 ;**
- **(id % 3^9) + 1 si 3^9 <= id < 3^9 * 2 ;**
- **(id % 3^9) + 2 si id > 3^9 * 2.**

De ce fait, si nous reprenons l'exemple précédent, nous allons avoir :

- id = 3 —> indice 3 du tableau ;
- id = 19686 —> indice 4 du tableau ;
- id = 39369 —> indice 5 du tableau.

Finalement, le seul petit défaut de cette notre représentation, sous forme de tableau de 3^9 cases, est la présence de certaines cases vides dû aux grilles invalides ainsi que l'espace mémoire occupé qui est assez important.

Représentation des billes et tirage au sort

Comme nous l'avons dit en introduction, notre IA de Morpion est basée sur l'apprentissage par renforcement ; plus précisément, nous avons implémenté le modèle "boîtes d'allumettes" de la machine MENACE.
Ce modèle attribut à chaque case des grilles une bille de couleur pour l'identifier. Pour jouer un coup, l'IA tire au sort une bille puis joue sur la case correspondante. Donc, à chaque fois que nous souhaitons **renforcer** un coup, il suffit **augmenter** le nombre de billes de la couleur concernée (et de retirer les billes pour éviter le coup).

Dans notre cas, nos cases sont **numérotées**, c'est ce qui fera office de couleur. Nous avons ensuite pris en compte le **nombre de billes par cases**.

Finalement, pour jouer un coup, nous avons écrit une fonction qui tire au sort un **chiffre** au hasard entre **1 et 9** ; à chaque chiffre est associé un **poids/coefficient** qui est le nombre de bille de la case numérotée. **Plus le nombre de bille d'une case est grand, plus la probabilité de tirage du chiffre, représentant le numéro de la case associé, augmente**.

De plus, ce nombre de bille nous permet de savoir qu'elle case est jouable au prochain coup ; en effet, si nbBille = 0, la case n'est pas jouable, sinon elle l'est.

Structures de données choisies

D'après tout ce qui précède, voici les structures de données que nous avons choisies :

typedef grille_t memoire[3^9] // notre tableau statique contenant l'ensemble des grilles

```
typedef struct case_s
{
  uint32_t num;          // numéro de la case (joue aussi le rôle de la couleur des billes)
  uint32_t coeffEtat;    // VIDE = 0, X = 1 et O = 2
  uint32_t nbBille;      // permet de symboliser le poids des numéros lors du tirage
}case_t;
```

```
typedef struct grille_s
{
  case_t cases[9];          // les 9 cases de la grille pour faire le tirage et calculer l'id
  uint32_t id;              // indice de la grille dans le tableau
  uint32_t jouee;           // permet de garder une trace des grilles rencontrées
  uint32_t renforcee;       // évite que les grilles soient renforcées plusieurs fois
  uint32_t billeSortie;     // permet de retenir le numéro de la case jouée (renforcement)
  uint32_t equiv[8];        // tableau stockant l'id des grilles équiv dans l'ordre établie
  uint32_t finale;          // permet de déterminer le gagnant de la partie si terminée
}grille_t;
```

Nous avons déjà abordé le fonctionnement de la structure case précédemment.
Concernant la structure grille, nous allons détailler davantage.

Lorsque l'IA doit jouer un coup, elle doit effectuer un tirage. Or, ce tirage doit dépendre de toutes les parties précédentes. C'est pourquoi le tirage est effectué à partir des grilles stockées en mémoires dans notre tableau statique. Cela explique le fait que nous avons besoin d'accéder aux **cases** de notre grille afin d'effectuer les tirages. De plus, pour retrouver la grille actuelle dans le tableau de grilles, l'IA doit calculer l'id de cette grille ; donc cela explique aussi la présence des 9 cases.

Les champs **jouee** et **billeSortie** sont utilisés à la fin de la partie, lorsqu'il faut renforcer l'IA ; en effet, jouee permet de retenir les grilles rencontrées au cours de la partie ("boîtes d'allumettes ouvertes") et billeSortie la case qui a été jouée ("couleur de la bille tirée").

Le champ **final** permet à l'IA d'arrêter de jouer et de passer à la phase de renforcement.

Aussi, le **tableau equiv** permet d'identifier les grilles équivalentes par rotation et symétrie verticale. En effet, ce tableau de **8** entiers contient les 7 id des grilles équivalentes à une grille donnée, en plus de son propre id ; cela vient du fait que dans le jeu du Morpion, le nombre de grilles équivalentes est d'au plus 7 pour une grille donnée. Si nous considérons une grille initiale, il existe 3 équivalentes par **rotation** (90°-180°-270°) ainsi que les 4 **symétriques** aux 4 précédemment citées (mais il se peut que des grilles équivalentes soient les mêmes, d'où le champ renforcee).

Pour pouvoir déterminer le lien entre 2 grilles, nous avons adopté la **convention** suivante concernant l'ordre des id dans le tableau equiv :

Indice du tableau equiv	0	1	2	3	4	5	6	7
Exemple d'id stocké	6810	6858	378	978	254	4626	5346	326
Nature de la grille associée	rotation 0°	sym 0°	rotation 90°	sym 90°	rotation 180°	sym 180°	rotation 270°	sym 270°

Avec cette convention, nous savons que les indices 2-4-6 contiennent l'id des rotations et 1-3-5-7 ceux des symétriques. Il faut cependant noter que l'ordre des id du tableau equiv est propre à chaque grille équivalente puisque l'id initial (rotation 0°) ne sera pas le même. Nous avons donc écrit une fonction qui adapte l'ordre des id du tableau equiv en fonction de l'id de la grille concernée.

Enfin, le champ renforcee permet de garder une trace des grilles déjà renforcée une fois. En effet, notre fonction de renforcement renforce toutes les grilles équivalentes de la grille passée en paramètre. Or, il peut arriver que nous ayons des id redondant dans notre tableau equiv : par exemple, pour une grille contenant une croix dans le coin supérieur gauche, sym0° et rot90° auront le même id. Pour ne pas renforcer deux fois, il nous suffit de mettre à 1 ou 2 le champ renforcee après le premier renforcement.

Finalement, notre structure grille contient un tableau **cases** de 9 cases, un tableau **equiv** de 8 entiers, une entier **id** compris entre 0 et 3^9 - 1, un entier **billeSortie** compris entre 1 et 9, 2 entiers **jouee** et **rendorcee** pouvant valoir 0, 1 ou 2 (VIDE, X, O) et enfin 1 entier **finale** compris entre 0 et 3 (ENCOURS, XGAGNE, OGAGNE, EGALITE).

Déroulement d'une partie

Voici le déroulement d'une partie de Morpion contre l'IA que nous avons programmée (nous avons supposé que l'utilisateur joue les X et l'IA les O):

- Au début la grille est vide. Un tirage au sort détermine qui de l'utilisateur ou de l'IA joue en premier.
- Supposons que l'IA joue d'abord. Elle commence par calculer l'id de la grille qu'elle a en face d'elle, dans ce cas la grille vide, puis recherche cette grille en mémoire (le tableau statique).
- Si cette grille n'existe pas, l'IA va alors mettre à jour ses champs equiv et finale avant de l'ajouter à sa mémoire. La mise à jour du tableau equiv va permettre de renforcer les grilles équivalentes lors du renforcement de la grille actuelle.
- Ensuite elle fait le tirage au sort à partir de la grille initialement ajoutée.

- Si la grille existe, l'IA va la rechercher dans sa mémoire puis effectuer le tirage au sort avec coefficients pour déterminer sur qu'elle case elle doit placer son O.
- L'IA va ensuite marquer cette grille et toutes les équivalentes comme jouées et retenir la case tirée au sort (billeSortie).
- Après cela, c'est à l'utilisateur de jouer, il place sa X où il le souhaite.
- L'IA refait ensuite la même chose que précédemment.
- Une fois la partie terminée (champ finale différent de 0), l'IA passe à la phase de renforcement. Elle va soit augmenter, soit diminuer le nombre de billes des cases tirées au sort pour les grilles jouées.
- L'augmentation aura lieu en cas de victoire de l'IA (+3) ou d'égalité (+1) car ces deux issues sont celles qui nous voulons favoriser.
- En cas de défaite, l'IA diminue le nombre de billes des cases tirées au sort dans les grilles correspondantes car elles l'ont menée à la défaite.
- À chaque étape du renforcement, l'IA renforce aussi les grilles équivalentes à la grille qui vient d'être renforcée, grâce au tableau équiv qui contient leur id. Grâce à la convention adoptée quant à l'ordre dans equiv, nous pouvons transformer billeSortie pour l'adapter aux grilles équivalentes.
- Ce déroulement est répété à chaque partie.

Découpage de notre programme

Nous avons choisi de découper notre programme en 5 fichiers sources et 4 headers associés (sauf pour le main bien évidemment):

- case_t.c : contient les fonction de manipulation des cases;
- grille_t.c : contient les fonction de manipulation des grilles;
- ia.c : contient les fonction de manipulation du tableau statique mémoire;
- morpion_console.c : contient les fonction d'affichage en mode console ainsi que les fonctions de jeu (mode1, mode2, morpion_ia_jouer…);
- main.c : fonction principale pour lancer le jeu.

En plus de ces fichiers, un fichier texte save.bin est présent. Il contient la sauvegarde de la mémoire de l'IA. Ce fichier est crée lorsque l'utilisateur quitte le jeu et est chargé en début de jeu s'il le demande. Les fonctions de sauvegarde et de chargement sont incluses dans le fichier ia.c.
Pour la sauvegarde et le chargement, nous utilisons les fonctions fread() et fwrite(). Cela nous permet d'avoir une grande efficacité, simplicité et rapidité lors de ces étapes, même si le fichier save.bin n'est pas lisible par un humain.