Vingere Camille
Hasna Id-Mouden
Polytech Lille
IMA3
2019-2020

Rapport programmation

JEU OTHELLO

Sommaire

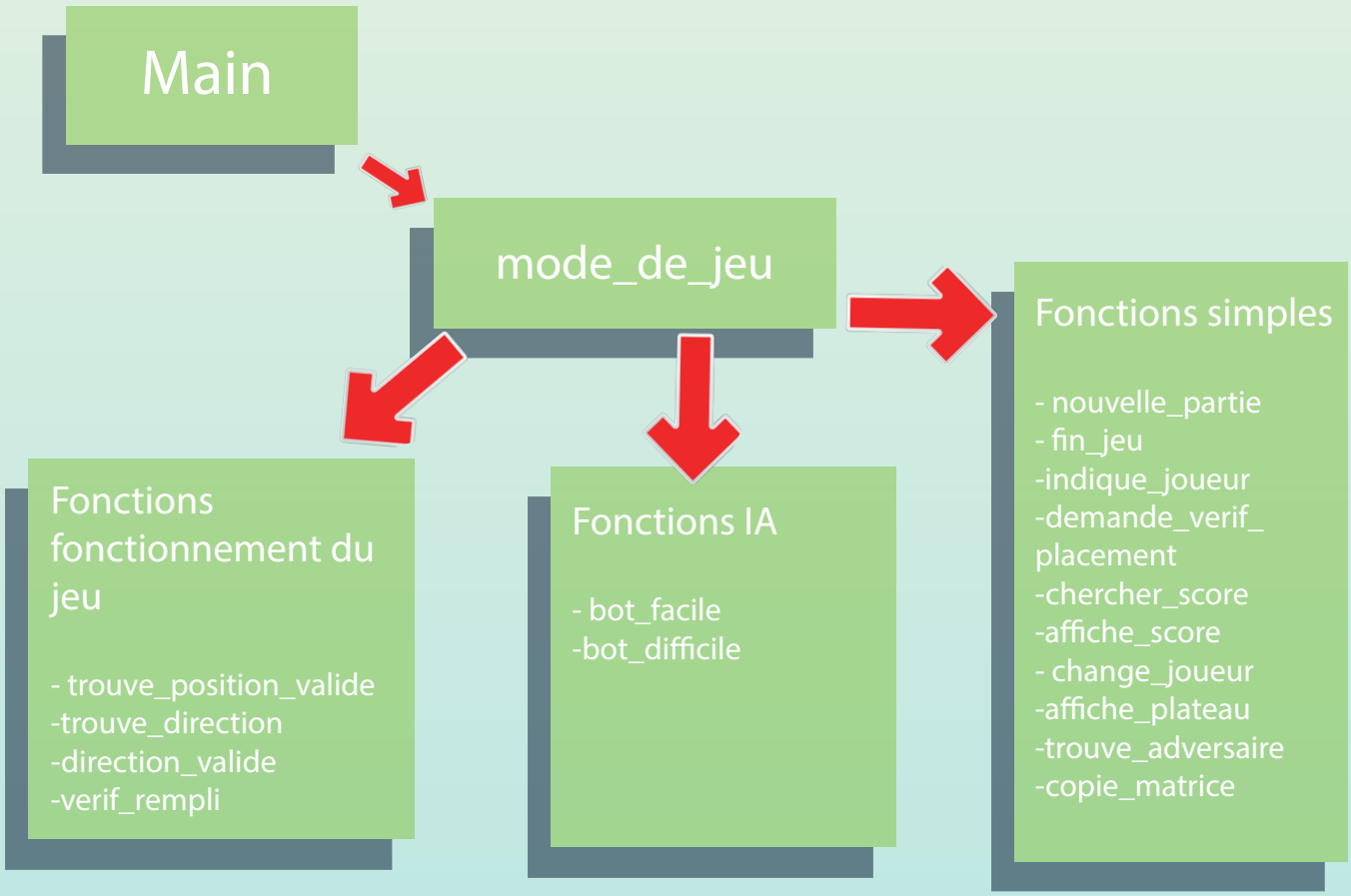
Introduction

Le but de ce projet est de programmer le jeu Othello en programme C. D'abord pour deux joueurs humains puis avec la possibilité de jouer contre l'ordinateur.

Les fonctions principales sont les fonctions qui servent à faire fonctionner et à appliquer les règles du jeu ainsi que les deux IA. Le reste des fonctions sont seulement des outils pour faciliter la lecture où qui sont souvent utilisées. Leurs algorithmes est facilement compréhensible en lisant les commentaires directement dans le programme.

Le joueur doit placer un pion en respectant les règles. Il faut donc vérifier à chaque fois qu'il joue si le placement du pion est valide.

Premièrement on crée une fonction qui cherche tous les placements valides (c'est à dire qui respectent les règles du jeu).



I / Fonction règles

a) Fonction : trouve_position_valide

Elle prend en paramètre le joueur (caractère b ou n pour blanc ou noir), la matrice jeu (dans laquelle est stockée le plateau de jeu), la matrice jeuvalide et un tableau DIRECTIONSVALIDES.

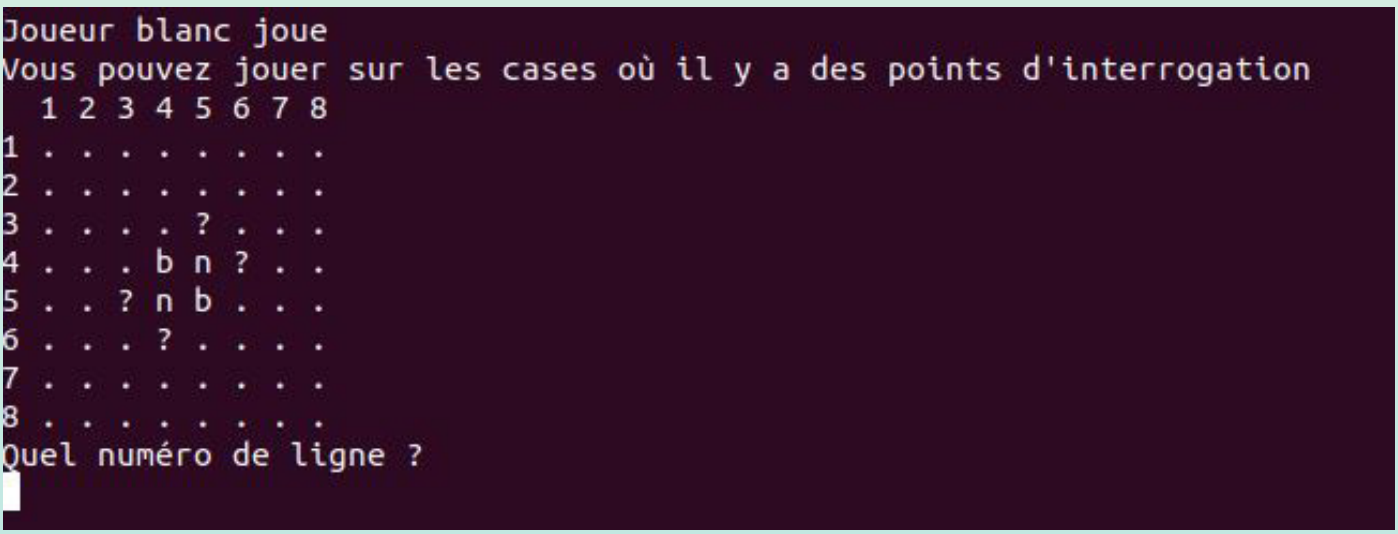
Premièrement on copie la matrice jeu dans la matrice jeuvalide (avec la fonction simple copie_matrice) pour ne pas altérer le plateau de jeu avec nos futures affectations.

Ensuite, on parcours la matrice jeu et à chaque fois qu'on croise l'adversaire du joueur on vérifie toutes les directions autour.

Une fois toutes les cases vérifiées nous avons le plateau jeuvalide qui répertorie toutes les positions valides sur lesquelles peut jouer le joueur actuel.

Pour vérifier toutes les positions autour d'un adversaire on fait appel à la fonction direction_valide.

En utilisant affiche_plateau(jeuvalide) on obtient le plateau suivant. Les points d'interrogations représentent les cases où le joueur peut jouer.



b) Fonction : trouve_direction

Cette fonction va trouver les opérateurs à appliquer à une ligne et à une colonne d'une matrice pour parcourir dans une direction. Elle va être appelée par direction_valide et verif_rempli.

Exemple : Pour parcourir la matrice à gauche depuis une case à l'emplacement (ligne,colonne)

jeu[ligne+i*pligne][ligne+i*pcolonne] …
i étant le nombre de case de déplacement souhaité
pligne et pcolonne fixant les directions dans la matrice

Pour un déplacement de 1 case à gauche →i=1, pcolonne = -1, pligne = 0.

Donc pour aller à gauche on se déplace juste de colonne en colonne.

Donc cette fonction prend en paramètre l'adresse de pcolonne et l'adresse de pligne (car on ne peut pas faire un return avec deux paramètres) et la direction désirée.

Les directions sont définies avec des #define pour avoir des directions visuelles mais avoir la possibilité de pouvoir travailler avec des chiffres, pour les switch notamment (car les switchs ne fonctionnent pas avec des chaînes de caractères donc nous n'aurions pas pu indiquer les directions avec des chaînes de caractères).

```
#define GAUCHE 1
#define DROITE 2
#define BAS 3
#define HAUT 4
#define DIAGHAUTDROITE 5
#define DIAGHAUTGAUCHE 6
#define DIAGBASDROITE 7
#define DIAGBASGAUCHE 8
```

c) Fonction : direction_valide

On commence avec un for de 0 à 7 (pour nos 8 directions).

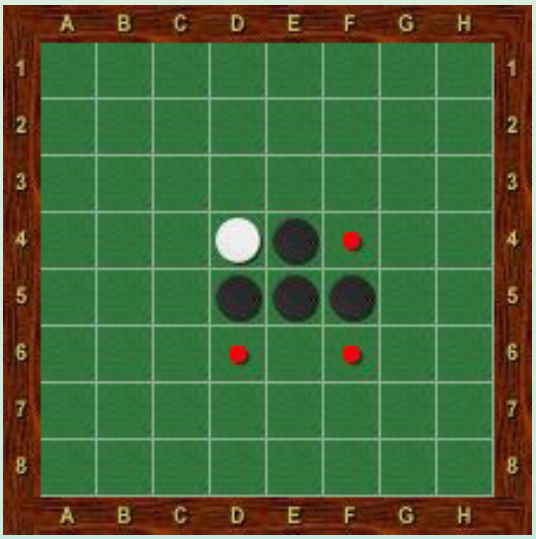
Donc on utilise d'abord notre fonction trouve_direction pour trouver la valeur de nos opérateurs. L'astuce ici est d'avoir un tableau TOUTESDIRECTIONS :

```
int TOUTESDIRECTIONS[8] = {GAUCHE,DROITE,BAS,HAUT,
DIAGHAUTDROITE,DIAGHAUTGAUCHE,DIAGBASDROITE,DIAGBASGAUCHE};
```

Donc à chaque fois qu'on incrémente le compteur dans le for, on change de direction et on refait le test.

Le test :

Si, dans une direction, sur la case actuelle ou suivante il y a l'adversaire ou le joueur (suivant si on appelle la fonction depuis trouve_position_valide ou verif_rempli) alors on regarde si il existe une direction pour laquelle on rencontre un joueur. Pour faire simple on vérifie si il existe une direction qui, avec le joueur, peut encadrer un ou plusieurs pions de l'adversaire. Si c'est le cas, on place un '?' sur chaque case qui rempli cette condition dans la matrice jeuvalide.



On prendra l'exemple de l'adversaire sur la case E5 pour joueur=blanc.
En effet dans la direction gauche on rencontre un joueur donc si la case est vide à droite (F4) on pose un '?' (ici le point rouge) car le placement est valide.

On retiendra également les directions qui ont données une position valide (car cela va nous servir dans verif_rempli quand on voudra remplir le vrai plateau de jeu).

d) Fonction : verif_rempli

Maintenant qu'on connaît tous les bons placements, on lance verif_rempli quand le joueur a choisi une case valide (verifié par la fonction simple demande_verif_placement). La fonction fait appel à nouveau à direction_valide. Cette fois le but ne sera pas de remplir jeuvalide avec des '?' mais d'obtenir les directions valides suivant ce placement.

Une fois qu'on a toutes les directions valides, on rempli la case actuelle avec le joueur et on remplace tous les adversaires jusqu'à ce qu'on atteigne un joueur à nouveau. On le fera alors dans toutes les directions valides.

Voilà le principe de l'algorithme et les fonctions qui gèrent le fonctionnement du jeu.

II / Les IA

Les IA « faciles » et « difficiles » permettent au joueur de jouer contre l'ordinateur. Les nominations faciles et difficiles font référence au niveau du jeu et non à la complexité du code.

a) Bot facile

L'IA facile place un pion valide de façon aléatoire sur le plateau de jeu.

La fonction bot_facile s'occupe de ce mode de jeu.

Après avoir appelé la fonction trouve_position_valide, on appelle bot_facile. Une fois dans bot_facile, on compte le nombre de points d'interrogations dans jeuvalide (on rappelle que jeuvalide est généré par l'appel de la fonction trouve_position_valide appelée au préalable).

Ensuite on génère un nombre aléatoire entre 1 et le nombre de placements valides comptés juste avant.

Enfin, on place le pion sur le n-ième placement valide sur le plateau (n faisant référence au nombre aléatoire généré juste avant).

Exemple : On compte 4 points d'interrogations dans jeuvalide (il y a donc 4 placements valides).
On génère un nombre aléatoire en 1 et 4. La fonction rand nous génère 3.
On place le pion du bot sur le 3ème point d'interrogation dans la matrice jeu.

A la fin, nous affectons les valeurs de ligne et de colonne pour la case en question passés par adresse dans la fonction.

Après l'appel de bot_facile on appellera verif_rempli pour remplir les cases correspondantes au placement « choisi » par bot_facile.

Le time complexity de ce programme est $O(n^2) = O(81)$ car il y a 2 boucle for imbriquées. Il y a deux fois deux boucles for, toutefois $O(2n^2)=O(n^2)= O(81)$.

a) Bot difficile

L'IA difficile place un pion valide de sorte à ce que le pion placé rapporte le plus de points possible.

La fonction bot_difficile s'occupe de ce mode de jeu.

Après avoir appelé la fonction trouve_position_valide, on appelle bot_difficile.

On parcours la matrice et à chaque fois qu'on croise un '?' dans jeuvalide on retient l'ancienne valeur du score (la première fois c'est 0), on copie la matrice jeu dans jeutest, on utilise verif_rempli sur le placement en question et on rempli jeutest pour ne pas altérer le plateau de jeu, on regarde le score généré.

Si le score généré est supérieur à l'ancien score, on retient le placement (donc la ligne et la colonne).

À la fin le placement retenu est celui qui rapporte le plus de points donc on affecte les valeurs de ligne et de colonne passés par adresse en paramètre de la fonction.

Après l'appel de bot_difficile on appellera verif_rempli pour remplir les cases correspondantes au placement « choisi » par bot_difficile.

Le time complexity de ce programme est $O(n^2)= O(81)$ car il y a 2 boucle for imbriquées.

III / Quelques détails

La fonction main sert seulement à demander le mode de jeu que veut choisir l'utilisateur : mode humain ou mode computer. Si il choisi le mode computer il doit ensuite choisir le mode facile ou difficile.

Ensuite on fait appel à la fonction mode_de_jeu qui s'occupera de lancer l'algorithme adéquat au mode de jeu choisi par l'utilisateur. C'est donc la fonction mode_de_jeu qui va faire appel aux fonctions principales.

Conclusion

Notre programme fait face à des limites dans l'intelligence artificielle difficile.

En effet ce bot choisira toujours la première meilleure case qu'il a trouvé. Si il y a 3 placements qui permettent de gagner le maximum de points alors il choisira le premier car c'est un système comparatif. Le bot devient donc vite prévisible après quelques parties. Pour l'améliorer il faudrait retenir les cases qui donnent le plus de points et en choisir une au hasard. Nous avons préféré garder le programme simple tout en sachant qu'il faudrait l'améliorer.

Le programme fait aussi face à des problèmes de visibilité.

En effet celui ci est indenté et commenté mais la logique de programmation derrière celui-ci n'est pas simple à comprendre et très peu claire. Certaines fonctions sont « bricolées » afin de pouvoir être réutilisées dans d'autres avec des paramètres qui gênent pour la compréhension. Les fonctions ne sont donc pas facilement réutilisables d'un programme à un autre.

Enfin, l'importance n'a pas été donnée à l'affichage mais le jeu est un minimum compréhensible.