Bernaud Benjamin **Rapport de Projet** *L2MI*

C

• Création d'un jeu en C : le Puissance 4.

Lors de la création du jeu, j'ai dû au préalable réfléchir aux structures que j'allais utiliser. En effet il a fallu que je représente la grille du puissance 4, j'ai donc choisi de le faire grâce à une matrice (ou plutôt un tableau de tableaux) constituée d'un tableau pour chaque ligne, j'ai également créé une structure pour la position dans la grille car cela était plus simple et m'a donc fait gagner du temps par la suite.

Les règles du jeu sont simples : c'est un puissance 4 ! Il suffit d'aligner 4 jetons sur une ligne, une colonne, ou bien en diagonal. Cependant, certaines spécificités se sont ajoutées. En effet, il est désormais possible d'effectuer une rotation de la grille sur la droite, sur la gauche, ou même de retourner la grille.

Au niveau de la structure du code, toutes les fonctions sont écrites dans le fichier « jeu.c » dont les prototypes se trouvent dans le « jeu.h ». Seul un fichier « main.c » contient une seule fonction (la fonction main) qui permet de lancer le jeu. J'ai également construit mon programme de sorte qu'il puisse s'adapter à un nombre variable de lignes et de colonnes. Cependant la grille doit rester carrée afin de préserver un fonctionnement correct du programme.

• Fonctionnement

Au démarrage du programme, il faut regarder s'il est exécuté sur un appareil Windows, auquel cas il faut charger la librairie *windows.h* afin de pouvoir utiliser des *Sleep*. Si on l'exécute sur un autre OS alors charger la librairie *windows.h* crée une erreur, de plus, il existe une fonction similaire *sleep*.

Au lancement du jeu, un menu apparait nous demandant le nombre de joueur(s) prenant part à la partie. C'est grâce à ce menu que nous allons distinguer les choix par la suite dans un switch. Il ne faut pas oublier d'initialiser la grille avec des cases vides. Ensuite si un seul joueur prend part à la partie, il faut lui demander s'il décide de jouer en tant que joueur n°1 ou bien n°2.

Si 2 joueurs s'affrontent, alors c'est simple : on affiche la grille, on demande au joueur de faire un choix puis on modifie la case, enfin c'est à l'autre joueur de choisir. La vérification de la grille s'effectue à chaque fois qu'un joueur modifie la grille.

Si nous avons qu'un seul joueur présent, alors c'est un peu plus corsé puisqu'il a fallu mettre en place une IA qui va choisir où jouer, l'IA peut elle aussi effectuer des rotations (elle a 1 chance sur 20 par défaut, mais modifiable). Dans ce cas-là on affiche la grille, on demande au joueur de choisir puis on modifie la case, enfin on peut faire jouer l'IA. Pour savoir si l'IA effectue une action ou joue un jeton, on tire un nombre au hasard entre 1 et 20, si c'est 1 alors elle effectue une action, puis on tire un nombre au hasard entre 1 et 3 pour savoir si elle effectue ***rotat_D()***, ***rotat_G()*** ou ***retourne_plateau()***.

Avec ces nouvelles règles à ajouter, j'ai dû créer plusieurs fonctions. Évidemment il a fallu coder les différentes rotations (***rotat_D()***, ***rotat_G()*** et ***retourne_plateau()***) , mais il a également fallu mettre en place une fonction imitant la gravité, c'est-à-dire faire tomber les jetons lorsque l'on effectue une rotation. Celle-ci fut assez intuitive et simple à mettre en place.

Finalement, si le joueur entre un numéro de colonne invalide, alors on affiche une erreur et en guise de « punition » il ne joue pas, c'est désormais au tour de son adversaire.

• Difficultés

Au cours de la réalisation du projet, quelques difficultés plus ou moins importantes me sont apparues. En effet, j'ai eu du mal à implémenter les nouvelles règles car je n'ai pas compris tout de suite comment elles marchaient, puis comment transformer ma matrice pour qu'elle corresponde aux mouvements.

J'ai également eu des problèmes quant aux choix des actions puisque j'ai dû lire la réponse en ***char*** puis le transformer en ***int***, cela me permettant de regarder dans la table ascii puisque chaque joueur peut rentrer un numéro de colonne entre 1 et ***NB_co*** (constante contenant le nombre de colonnes), mais il peut aussi écrire **D** ou **d** pour la rotation à droite, **G** ou **g** pour la rotation à gauche, **R** ou **r** pour retourner le plateau et enfin **Q** ou **q** pour mettre fin à la partie.

Au niveau de l'IA, il a fallu initialiser ma variable coup avant de lui donner une valeur, ce que je n'ai pas réussi à faire : j'ai tenté de lui donner la valeur **NULL** au départ puis de calculer le coup, cependant avec cette méthode l'IA ne répondait plus et mon jeu crashait. J'ai également essayé de l'initialiser à 0, sans succès. C'est le seul warning que le terminal m'affiche à la compilation.

• Conclusion

Avec plus de temps, j'aurais principalement aimé pouvoir construire un algorithme plus efficace pour l'IA afin qu'elle puisse gagner plus souvent, notamment grâce à un algorithme alpha-beta (note : je pense qu'un algorithme min-max aurait très largement suffit au vu de la complexité du jeu). De plus, j'aurais également aimé implémenter la sauvegarde d'une partie, en écrivant dans un fichier texte le nombre de joueurs prenant part au jeu (pour pouvoir sauvegarder une partie par mode de jeu), puis écrire la grille afin de la récupérer au prochain lancement.

Finalement, grâce à ce projet j'ai pu manipuler différents objets et structures pour mettre en application de manière plus concrète les cours et TP effectués au cours du semestre. J'ai trouvé mon code assez compacte (~300 lignes en tout) même s'il aurait sûrement pu l'être plus encore…