Projet ADA : Jeu Puissance 4

ISSC 2A
Argenson Benjamin et Gardelle Camille

Ce projet a pour but de construire un jeux à deux joueurs qui peuvent être des humains ou des ordinateurs. Le package permettant de jouer est générique, et dans notre cas nous l'avons utilisé pour implémenter un jeu de Puissance 4. Les ordinateurs utilisent un algorithme MinMax pour déterminer quel coup jouer : cette algorithme permet de choisir un niveau de jeu.

Sommaire:

I. Exécution et Manipulation du Puissance 4
II. Exemples d'exécution
III. Les différents Packages
IV. Les principaux algorithmes

I. Exécution et Manipulation du Puissance 4

Pour exécuter le jeu de Puissance 4, ouvrez un terminal dans la racine du projet et tapez

```
# gnatmake main2joueurs.adb
# ./main2joueurs
```

Notre jeu de puissance 4 se joue sur une grille NbrCol=4 colonnes, NbrLig=3 lignes. Pour gagner une partie, il faut réussir à aligner N=3 pions.

Il y a trois modes différents de jeu :

- le mode deux joueurs humains : les joueurs utilisent le clavier pour jouer
- le mode joueur humain contre ordinateur : le joueur humain utilise l'entrée clavier, et l'ordinateur est de niveau Niv=2
- le mode deux ordinateurs : les deux ordinateurs ont respectivement un niveau Niv1=4 et Niv2=6, et jouent l'un contre l'autre

On peut modifier manuellement tous ces paramètres dans le fichier main2joueurs.adb :

ligne 15 package MyPuissance4 is new Puissance4 (NbrCol, NbrLi, N);
ligne 27 Niv,
ligne 40 Niv1,
ligne 54 Niv2,

Il suffit de remplacer la valeur définie par la valeur souhaitée, qui doit être un entier naturel.

II. Exemples d'exécution

Nous avons testé les différents modes de jeu :

- *mode joueurs* : on peut jouer sans difficulté
- *mode ordinateur contre joueur* : plus on augmente le niveau de l'ordinateur, plus il devient difficile de le battre. A titre d'exemple si en tant que joueur on entre la série de coups (1,3,4,2,2) on gagne si Niv<4 et on perd sinon.
- *mode ordinateurs* : les deux ordinateurs jouent tous seuls et on constate que c'est celui de plus haut niveau qui l'emporte

III. Les différents Packages

Dans un premier un temps, nous avons construit le package Partie et le package Puissance4 sans la fonction d'évaluation et la liste de coups afin de simuler une partie entre deux joueurs humains.
Nous avons au maximum essayer de respecter les structures de données qui étaient suggérer dans le sujet et dans le code source initial, ainsi les fonctions implémentées dans ces deux packages, sont celles suggérées par l'énoncé.

Package Puissance4

Il a tout d'abord fallu définir les types *Etat* et *Coup.*
Un coup est un entier naturel compris entre 1 et la largeur du plateaux: il correspond au numéro de la colonne dans laquelle le joueur veut placer son pion.
Le type Etat est une structure comportant quatres attributs:
- un tableau de caractères à deux dimensions (hauteur x largeur), représentant la "grille" du puissance 4, chaque case correspond à l'emplacement d'un pion. Une case contient soit 'X', correspondant à un pion joué par le joueur 1, soit 'O' correspondant à un pion joué par le joueur 2, soit ' ' en cas de case vide.
-un tableau a une dimension (1 x largeur), dont chaque case comptabilise le nombre de pions joués dans la colonne correspondante. Cela permet de vérifier très simplement si une colonne est pleine et donc de vérifier si le coup joué par un joueur est possible.
-un entier comptabilisant le nombre de pions joués par le joueur 1
-un entier comptabilisant le nombre de pions joués par le joueur 2

Les deux entiers permettent tout simplement de savoir à quel joueur c'est au tour de jouer. En effet si, un des deux entier est supérieur à l'autre alors c'est au tour du joueur associé à l'entier le plus petit de jouer. En cas d'égalité, c'est par défaut au tour du premier joueur de placer son pion.

Une fois les différents types définis on a pu implémenter les différentes fonctions permettant de caractériser le jeux Puissance 4 (Initialiser, Est_Gagnant, Est_Nul, Afficher, Afficher_Coup, Demande_Coup_Joueur1 et Demande_Coup_Joueur2). Le rôle de ces fonctions est le même que celui suggéré par l'énoncé et le code source .ads.

A noter que la fonction Initialiser remplit le tableaux à deux dimensions associés à l'état de case vide (contenant donc ' ').
La fonction **Est_Nul** accède au nombre de pions par colonne pour déterminer rapidement (avec un faible coût) si toutes les cases sont remplies.

La fonction **Afficher** parcours tout simplement le tableaux à deux dimensions et affiche la grille à l'écran.

La fonction **Jouer** définie met à jour la variable Etat associé au jeux en fonction du coup qui est passé en paramètre.

Enfin, la fonction **Demande_Coup_Joueur**, demande et retourne un entier naturel (de type Coup) tapé au clavier par le joueur. Si l'entier ne correspond pas à un Coup, alors une exception du type "Constraint_Erreur" est levée et est rattrapée de telle façon que la fonction demande au joueur de rentrer une nouvelle valeur jusqu'à ce qu'elle corresponde à un coup. De même, si l'entier correspond à une colonne pleine alors une erreur du type "CoupImpossible" est levée et rattrapée de la même façon.

La fonction **Eval** a été rajoutée par la suite pour que l'ordinateur soit en mesure de déterminer si un état lui est favorable ou non. Cette fonction est utilisée par l'algorithme MinMax pour choisir le coup le plus favorable à l'ordinateur.
La fonction **Coups_possibles** permet de fournir à l'Algorithme MinMax la liste des coups suivants possibles (qui varie selon les règles du jeu), afin d'évaluer lequel est le plus favorable à l'ordinateur.

Package Partie (générique)
Le package Partie, ne comprend qu'une seule fonction "**Joue_Partie**", permettant de simuler une partie (de Puissance 4 dans notre cas). Le package prend en paramètre les différentes fonctions du package Puissance 4 et demande successivement aux deux joueurs quel coup il souhaite et jouer et ceux jusqu'à obtenir un état nul ou gagnant.

Package Moteur_Jeu (générique)
Le package Moteur_Jeu est générique pour un jeu à deux joueurs et permet à un ordinateur de jouer. Il possède deux fonctions Choix_Coup et Eval_Min_Max.

Choix_Coup retourne le coup à jouer le plus favorable à l'ordinateur selon l'algorithme MinMax.

Eval_Min_Max estime combien un coup est favorable en simulant sur une certaine profondeur les états de jeu auxquels il conduit. Elle réalise l'évaluation de ces états grâce à la fonction Eval, qui est un paramètre de la généricité.

Package Participants
Ce package définit le type *joueur* et contient une fonction **Adversaire** qui permet de manipuler le type joueur.

Package Liste_Generique (générique)
Dans ce package, on a implémenté différentes fonctions permettant de manipuler des listes chaînes de n'importe quel type d'éléments puisque le type des éléments manipulés est un paramètres du package. La particularité de cette implémentation de liste chaînée est que le parcours des éléments se fait via un itérateur qui est un pointeur sur une cellule. Ainsi, on parcourt aisément les listes grâce aux fonctions Suivant (qui fait pointer l'itérateur sur la prochaine cellule) et A_Suivant qui vérifie si il reste un élément à parcourir.

IV. Les principaux algorithmes

Algorithme Eval

La fonction Eval retourne un entier qui représente à quel point un état E est favorable à un joueur J si on le considère statiquement.

Nous avons choisi d'implémenter une fonction statique simple : la fonction va compter le nombre de lignes de pions qui peuvent être complétés (avec une case vide à une extrémitée).

Pour chaque longueur L de ligne possible jusqu'à N-1, Eval est augmentée de L*L*nombre_ligne_de_taille_L si les pions qui constituent la ligne sont ceux de J. De la même façon, Eval est décrémentée si les pions sont ceux de l'adversaire de J.

Si E correspond à un état gagnant pour J, Eval renvoie N*N, qui est une valeur maximale. Si E correspond à un état perdant pour J, on renvoie -N*N. Et si E est une égalité, on renvoie 0.

Nous avons choisi de mettre au carré la longueur L car si une ligne est deux fois plus grande qu'une autre, alors elle est beaucoup plus facile à compléter que la première.

Algorithme de MinMax

L'algorithme de MinMax prend en paramètre un état E, un coup C, une profondeur P et un joueur J et retourne un entier.

Le principe de l'algorithme est le suivant : si un joueur est capable d'évaluer les états qui sont accessibles par une série de coups, alors il jouera le coup qui l'avantage le plus. On calcule ainsi récursivement tous les états accessibles à partir de E, jusqu'à la profondeur P. Puis on évalue ces états grâce à Eval, du point de vue de P. On recherche alors parmis les coups possibles lequel est le plus susceptible d'être joué, selon que ce soit à J ou à son adversaire de jouer.

Une fois qu'on a la valeur du coup le plus probable, on retourne son évaluation (c'est à dire la valeur de l'état auquel il mène) .

Conclusion

Ce TP nous a permis d'approfondir les différentes notions que nous avons vu en cours d'algorithmique dans le langage de programmation ada.

Il nous a également permis de découvrir le principe de fonctionnement d'une intelligence artificielle à l'aide d'un exemple concret et de l'algorithme MinMax.