DM d'Algorithmique en Pascal

Anne Bouillard & Loris Marchal

facultatif, à rendre le 10 ou 11 mai si vous le souhaitez

Résumé

On souhaite écrire un programme qui permette de jouer au morpion sur un ordinateur. Dans un premier temps, on jouera à deux joueurs, puis on implémentera un joueur virtuel pour jouer contre l'ordinateur. Si vous constatez une erreur dans l'énoncé, ou quelque chose d'incompréhensible, merci d'écrire à `loris.marchal@ens-lyon.fr` ou à `anne.bouillard@ens-lyon.fr`. Vous pouvez travailler à plusieurs sur ce devoir, mais nous vous demandons de rendre chacun un code original.

Règles du jeu du morpion

Le jeu du morpion se présente habituellement sous la forme d'une grille à 3×3 cases, sur laquelle chacun des joueurs dépose alternativement un pion. Ici, par manque de couleur, on représentera les pions du joueur 1 par des croix, et ceux du joueurs 2 par des cercles. Le joueur qui arrive à aligner trois pions de sa couleur a gagné. Il se peut qu'aucun des deux joueurs ne gagne et que la partie s'arrête parce que la grille est complètement remplie.

(a) Le joueur 1 gagne

(b) Aucun des joueurs ne gagne

FIG. 1 – exemples de situations finales

Comme le jeu sur une grille 3×3 est d'un intérêt limité, on souhaite pouvoir étendre le morpion à de plus grande taille, en changeant également le nombre de pions à aligner pour gagner. L'objectif pourrait ainsi devenir : sur une grille 11×11, réussir à aligner 4 pions.

Définition de types

On propose les types suivants, à utiliser pour la suite du devoir :

```
CONST TAILLE_GRILLE     = 3;
      NB_DE_PIONS_A_ALIGNER = 3;

Type t_indice_grille = 1..TAILLE_GRILLE;
     t_pion      = 0..2;
     t_grille      = array[t_indice_grille,t_indice_grille] of t_pion;
```

La taille de la grille est une constante fixée à trois pour le début. La deuxième constante est le nombre de pions à aligner pour gagner.

Le contenu de la grille proprement dit (champ `cases`) est un tableau à deux dimensions. Pour accéder à la case situer sur la ligne *ligne* et la colonne *colonne*, il suffit d'écrire `cases[ligne][colonne]`.

1 Fonctions et procédures à écrire

1.1 Affichage

Écrivez une procédures qui permettent l'affichage d'une grille de morpion. L'entête de cette procédure est le suivant :

```
Procedure Affichage(grille : t_grille);
```

Vous devez obtenir un affichage plus ou moins proche de ceci :

```
+---+---+---+
| X | O |   |
+---+---+---+
|   | X | O |
+---+---+---+
|   | O |   |
+---+---+---+
```

1.2 Test de réussite

Commencez par écrire une fonction

```
Function Test_Ligne_Horiz(grille : t_grille; ligne, colonne, joueur : integer) : Boolean;
```

qui vérifie si une série horizontale de pions du joueur *joueur* commence à la case située à la colonne *colonne* sur la ligne *ligne*. Par exemple, sur la grille suivante, cette fonction devra renvoyer `true` pour $ligne = 2$, $colonne = 1$ et $joueur = 1$.

```
+---+---+---+
|   | O |   |
+---+---+---+
| X | X | X |
+---+---+---+
| O | O |   |
+---+---+---+
```

Sur le même modèle, écrivez les fonctions suivantes :

```
Function Test_Ligne_Vert(grille : t_grille; ligne, colonne,joueur : integer) : Boolean;
Function Test_Diag_Montante(grille : t_grille; ligne, colonne, joueur : integer) : Boolean;
Function Test_Diag_Descendante(grille : t_grille; ligne, colonne, joueur : integer) : Boolean;
```

Les deux dernières fonctions testent s'il existe une diagonale (montante ou descendante) appartenant au joueur *joueur* qui démarre à la case définie par les paramètres *ligne* et *colonne*.

Ces quatre fonctions doivent gérer proprement les débordements : si elle sont appelées d'une façon incorrecte, et que la ligne à tester déborde de la grille, ces fonctions doivent afficher un message d'erreur sur l'écran.

Enfin, en utilisant les fonctions définies ci-dessus, écrivez une fonction qui vérifie si un joueur a gagné (le numéro du joueur est passé en paramètre à la fonction) :

```
Function Test_Reussite(grille : t_grille; joueur : integer) : Boolean;
```

1.3 Ajout d'un pion

Écrivez une procédure qui permettent d'ajouter un pion, dont la position et la couleur (le numéro du joueur) sont passés en paramètre :

```
Procedure Ajout_Pion(Var grille : t_grille; ligne, colonne, joueur : integer);
```

1.4 Faire jouer un joueur

Écrivez une procédure qui demande à un joueur d'entrer la position d'un nouveau pion, qui vérifie qu'il est bien possible d'ajouter ce pion, puis l'ajoute (à l'aide de la fonction précédente).

```
Procedure Saisie_Pion(Var grille : t_grille; joueur : integer);
```

2 Programme

À l'aide des procédures et fonctions définies précédemment, écrivez un programme qui permet de jouer à deux joueurs au morpion.

3 Amélioration : jouer contre l'ordinateur

Dans cette section, nous vous présentons quelques pistes pour que les plus motivés d'entre vous puissent continuer, cependant **cette partie ne sera pas prise en compte dans la notation**.

Écrivez une fonction permettant l'ajout d'un pion sur la grille :

```
Function Ajout_Pion_Fonc(grille : t_grille; ligne, colonne, joueur :
integer) : t_grille;
```

La différence par rapport à la question 1.3 est qu'ici, le résultat est renvoyé comme **valeur de retour** de la fonction et non plus comme paramètre passé par adresse : ainsi l'argument `grille` passé à la fonction ne sera pas modifié.

On rajoute une constante :

```
CONST SCORE_MAX = 100;
```

On rajoute également un type :

```
Type t_position = Record
                    ligne : t_indice_grille;
                    colonne : t_indice_grille;
                  End;
```

Ce type pourra vous être utile pour stocker les différents coups à jouer.

Écrire une fonction récursive :

```
Function Teste_tous_les_coups_possibles(grille : t_grille; joueur : integer) : Integer;
```

Cette fonction doit renvoyer un "score" qui correspond à l'intérêt que le joueur a à jouer ce coup :

- si la fonction renvoie le score maximum `SCORE_MAX`, alors ce coup est gagnant (c'est-à-dire qu'il conduit nécessairement à une situation où l'on gagne)
- en revanche, si la fonction renvoie le `-SCORE_MAX`, alors ce coup est perdant.

Cette fonction devra :

1. énumérez tous les coups que le joueur peut faire (c'est-à-dire toutes les cases libres de la grille) ;

2. pour chacun de ces coups :
   - jouer ce coup à l'aide la fonction précédente `Ajout_Pion_Fonc`, ce qui donne une nouvelle grille
   - tester si on gagne avec ce nouveau coup : dans ce cas, renvoyer le score maximum `SCORE_MAX`
   - sinon :
     - appelez récursivement `Teste_tous_les_coups_possibles` sur cette nouvelle grille, avec le numéro de l'autre joueur : ceci revient à examiner tous les coups que l'autre joueur peut jouer si on a joué ce coup
     - calculer le maximum max des scores obtenus pour chacun de ces coups (on suppose que l'adversaire est malin et joue le meilleur coup possible)
     - renvoyer l'opposé de ce maximum : $-max$ (notre gain est l'opposé du gain de l'adversaire).

Enfin, à l'aide de cette fonction, écrivez une fonction :

```
Function Meilleur_coup(grille : t_grille; joueur : integer) :
```

qui renvoie le coup ayant obtenu le meilleur score parmi tous les coups possibles.

À l'aide des fonctions/procédures précédentes, écrivez un programme qui permet de jouer au morpion contre l'ordinateur.

Toutes les améliorations sont les bienvenues.