Module ITC313 - Informatique

Partie C / C++

TP11 + TP12 Projet de synthèse

Benoît Darties - benoit.darties@u-bourgogne.fr
Université de Bourgogne

Année universitaire 2016-2017

Exercice 1 : *Realisation d'un jeu de Puissance 4*

Les questions de cet exercice doivent vous permettre de mettre en oeuvre les concepts que vous avez vu sur la programmation objet dans un premier programme de taille conséquente. Nous utiliserons pour cela la réalisation d'un jeu de Puissance 4, dont toute l'algorithmique vous a déjà été communiquée. Il convient donc ici simplement de mettre en oeuvre des algorithmes précédemment définis, en utilisant la notion d'objet. L'intérêt repose sur le fait que tout élément physique ou abstrait va pouvoir être modélisé avec une classe

Dans cet exercice, nous mettrons en place un jeu de puissance 4 basé sur une implémentation avec des classes C++. Nous utiliserons les classes suivantes :

1. La classe Plateau
2. La classe Jeton
3. La classe Joueur
4. La classe Partie

Chaque classe possède une déclaration présente dans un fichier de même nom, et d'extension `.h`. La définition des différentes méthodes de chaque classe se situe dans un fichier de même nom, et d'extension `.cpp`. Pour réaliser les différentes méthodes de ces classes, on s'aidera du TD d'algorithmique relatif à la mise en place du jeu Puissance4. Cependant, au lieu de travailler avec une matrice de caractères comme dans les TD d'algorithmique, nous utiliserons une matrice de pointeurs sur des jetons. Lorsque la case est occupée, le pointeur pointera sur un Jeton auquel est associé une couleur. Lorsque la case est libre, le pointeur pointera sur `NULL`. On définira également un fichier `Makefile` permettant de compiler facilement le projet.

1. Ecrivez la déclaration et la définition de la classe `Jeton` dans deux fichiers `Jeton.h` et `Jeton.cpp`. Cette classe contient les éléments suivants :
   (a) un attribut privé `couleur` (un caractère prenant comme valeur `'J'` si le jeton est jaune ou `'R'` si le jeton est rouge ;
   (b) un constructeur prenant un paramètre de type caractère qui initialise l'attribut `couleur` avec le paramètre donné. Il vérifiera l'intégrité de ce paramètre, et attribuera à l'attribut une valeur par défaut si la valeur du paramètre est erronée.
   (c) un accesseur en lecture sur l'attribut `couleur`
2. Ecrivez la déclaration et la définition de la classe `Plateau` dans deux fichiers `Plateau.h` et `Plateau.cpp`. Cette classe contient les éléments suivants :

   (a) un attribut `grille` de type matrice de pointeurs sur des jetons, de dimensions 7×6 ;
   (b) un constructeur sans paramètres qui initialise l'ensemble des cellules de `grille` avec la valeur `NULL` ;
   (c) une méthode `ajouterJeton()` dont le premier paramètre est un caractère désignant une couleur de jeton ('R' ou 'J') et le second un indice de colonne compris entre 0 et 6 inclus. Cette méthode crée un jeton de couleur correspondante, et l'ajoute à la grille de jeu dans la colonne correspondante. L'ajout se fait en recherchant la cellule de la colonne donnée pour laquelle la valeur contenue est égale à `NULL` et l'indice de ligne est le plus petit possible. Il suffit alors d'affecter la valeur de cette cellule avec l'adresse mémoire du jeton correspondant. Réfléchir à la zone mémoire dans laquelle doit être créé le jeton. La méthode retourne 1 si l'ajout s'est correctement passé, et 0 en cas d'impossibilité d'ajout (indices hors domaine, colonne pleine ...) ;
   (d) une méthode `grillePleine()` qui renvoie vrai si et seulement la grille est pleine, c'est à dire que toutes les cellules de la grille ont été affectées avec l'adresse mémoire d'un objet de type `Jeton`.
   (e) une méthode `grilleGagnante()` qui renvoie vrai si et seulement si la grille est gagnante, c'est à dire qu'il existe au moins 4 jetons de la même couleur alignés horizontalement, verticalement, ou en diagonale sur la grille.

3. Ecrivez une classe `Joueur` contenant les éléments suivants :
   (a) deux attributs décrivant le nom et le prénom d'un joueur ;
   (b) un constructeur initialisant ces deux attributs avec deux paramètres donnés.
4. Ecrivez enfin une classe `Partie` dans deux fichiers `Partie.h` et `Partie.cpp` permettant de gérer une partie de Puissance 4. Cette classe contient les éléments suivants :
   (a) Deux attributs privés `joueur1` et `joueur2` de type pointeur vers des objets `Joueur` ;
   (b) Un attribut privé de type `Plateau` ;
   (c) un constructeur dont les deux paramètres sont des objets de type `Joueur`, et qui initialise les deux attributs `joueur1` et `joueur2` avec les adresses de ces paramètres.
   (d) Une méthode publique `lancerPartie()` qui simule le déroulement d'une partie. Le joueur 1 est associé à la couleur rouge, l'autre à la couleur jaune. Tant que la grille n'est pas gagnante ou remplie, cette méthode demande successivement à chaque joueur de rentrer un indice de colonne dans lequel ajouter un jeton de couleur associée au joueur. Si l'ajout est invalide (indice hors limite, colonne pleine), la saisie est redemandée au même joueur. La méthode renvoie un pointeur vers le joueur gagnant, ou `NULL` en cas d'égalité.
5. Redirigez les opérateur de sortie `«` des différentes classes de sorte que l'on puisse afficher les propriétés d'un joueur, la grille de jeu ainsi que le déroulement de la partie, au moyen de l'instruction `cout`. Modifiez enfin la méthode `lancerPartie()` de sorte à afficher le résultat du plateau après chaque coup valide joué.
6. Terminez ce programme en créant une fonction `main()` dans un fichier `Puissance4.cpp` qui demande à l'utilisateur de rentrer deux noms et prénoms de joueurs, puis initialise une partie et affiche le nom du joueur gagnant.

L'objectif de ce TP est de définir un jeu de puissance 4 entièrement écrit en C. Dans un soucis de lisibilité et de facilité de rédaction, l'ensemble des fonctions et procédures nécessaires à la mise en place de ce jeu de Puissance 4 est présenté progressivement. Une partie facultative consistera à ajouter de l'intelligence artificielle à ce jeu. Vous disposez des séances de TP pour réaliser ce projet.

Vous avez jusqu'au 25 juin 23h59 pour envoyer votre code source ainsi qu'un compte rendu au format pdf uniquement correspondant au développement de votre TP à l'adresse mail benoit.darties@u-bourgogne.fr avec comme sujet `"[Geipi 1A] projet"`

1 Rappel des règles du jeu

Le but du jeu est d'aligner 4 pions sur une grille comptant 7 colonnes et 6 rangées. Chaque joueur dispose de 21 pions d'une couleur (par convention, en général jaune ou rouge). Tour à tour les deux joueurs placent un pion dans la colonne de leur choix, le pion coulisse alors jusqu'à la position la plus basse possible dans ladite colonne suite à quoi c'est à l'adversaire de jouer. Le vainqueur est le joueur qui réalise le premier un alignement (horizontal, vertical ou diagonal) d'au moins quatre pions de sa couleur. Si alors que toutes les cases de la grille de jeu sont remplies aucun des deux joueurs n'a réalisé un tel alignement, la partie est déclarée nulle.

2 Modélisation

2.1 Modélisation des jetons et des joueurs

La couleur d'un jeton peut être représentée par un caractère : 'J' pour un jeton de couleur jaune, et 'R' pour un jeton de couleur rouge. Un joueur étant associé à un jeton, on peut également considérer que chaque joueur peut être représenté par un caractère 'J' ou 'R'. Par convention on dira que le joueur qui commence est celui avec les jetons 'jaunes'

2.2 Modélisation de la grille

Pour modéliser la grille de jeu, nous utiliserons une matrice M de 7 colonnes par 6 lignes. Les colonnes sont indicées de 0 à 6, les cellules d'indice de colonne 0 représentant la colonne la plus à gauche, et celles d'indice 6 la colonne la plus à droite. Les lignes sont indicées de 0 à 5. Les cellules d'indice de ligne 0 représente la ligne la plus basse du plateau de jeu, tandis que les cellules d'indice 5 la ligne la plus haute.

Chaque cellule peut prendre l'une des valeurs suivantes :

— . : on considère alors que la case correspondante n'est occupée par aucun jeton.

— J : on considère alors que la case correspondante est occupée par un jeton jaune

— R : on considère alors que la case correspondante est occupée par un jeton rouge

Par soucis de simplicité, on considèrera durant tout le TP qu'à chaque instant, la grille sur laquelle nous travaillerons possède une configuration possible, c'est à dire que si une cellule est occupée par un jeton, alors toutes les cellules de même indice de colonne et d'indice de ligne inférieur sont également occupées par un jeton. Par ailleurs, si une grille est gagnante, la partie s'arrête. Il ne peut donc pas y avoir plus d'une couleur qui possède au moins quatre pions alignés. Un exemple d'une telle grille est présenté ci après.

```
. . . . . . .
. . . . . . .
. . . . J . .
J . . R J . .
R R R J J R .
R J J J R R J
```

Exemple de grille de jeu

1. Quelle entité et zone de mémoire devriez-vous utiliser pour stocker cette grille ?
2. Ecrivez une fonction *nouvelleGrille()* qui ne prend aucun paramètre, réserve une zone mémoire correspondant à une matrice de dimensions 7 par 6, remplit l'ensemble de ses cellules avec la valeur ., et renvoie un pointeur sur l'espace mémoire stockant cette grille.
3. Ecrivez une procédure *affiche()* qui prend en paramètre une matrice de dimensions 7 par 6, et affiche la grille de jeu dans laquelle les pions du joueur 1 sont représentés par le caractère 'J', ceux du joueur 2 par le caractère 'R', et l'absence de pion par le caractère '.' , en rajoutant les colonnes et les bords inférieurs de la grille. Un exemple correspondant à l'affichage d'une matrice est présentée ci après.

```
| . | . | . | . | . | . | . |
| . | . | . | . | . | . | . |
| . | . | . | . | R | . | . |
| R | . | . | R | J | . | . |
| J | J | J | R | J | R | . |
| R | R | J | J | J | R | J |
-----------------------------
```

4. Testez vos deux fonctions

3 Evaluation de la grille

Nous allons définir les fonctions permettant de déterminer si une grille est gagnante pour l'un ou l'autre des joueurs. Par simplicité, on considèrera seulement les tests pour une configuration normale de jeu. A chaque étape, on pourra réutiliser au besoin les fonctions précédemment définies.

1. Ecrivez une fonction *estRemplie()* qui renvoie vrai si et seulement si toutes les cases de la grille sont occupées par un jeton.
2. Ecrivez une fonction *gagnantLigne()* qui renvoie :
   — 1 s'il existe un alignement horizontal d'au moins 4 cellules adjacentes de valeur J
   — 2 s'il existe un alignement horizontal d'au moins 4 cellules adjacentes de valeur R
   — 0 dans tout autre cas
3. Ecrivez une fonction *gagnantDiagonale()* qui renvoie :
   — 1 s'il existe un alignement en diagonale d'au moins 4 cellules adjacentes de valeur J
   — 2 s'il existe un alignement diagonale d'au moins 4 cellules adjacentes de valeur R
   — 0 dans tout autre cas
4. Ecrivez une fonction *gagnant()* qui renvoie :
   — 1 s'il existe un alignement d'au moins 4 cellules adjacentes de valeur J
   — 2 s'il existe un alignement d'au moins 4 cellules adjacentes de valeur R
   — 0 dans tout autre cas
5. Ecrivez une fonction *matchNul()* qui renvoie vrai si est seulement si la grille est remplie et qu'aucun joueur n'a gagné.

4 Actions des joueurs

1. Ecrire une fonction `colonneRemplie()` qui prend comme paramètre la grille de jeu et un indice de colonne, et renvoie vrai si et seulement si la colonne est remplie de jetons, c'est à dire que le joueur ne peut poser de jetons dans cette colonne.
2. Ecrire une fonction `poserJeton()` qui prend comme paramètres la grille de jeu, un indice de colonne , et un numéro de joueur. Si la colonne n'est pas remplie, cette fonction glisse un jeton du joueur correspondant dans la colonne indiquée et renvoie 0, et -1 autrement.

5 Interface de jeu

1. Ecrire une fonction *Puissance4()* qui décrit le déroulement d'une partie de puissance 4, en incluant les opérations suivantes :
   — Initialise la grille de jeu ;
   — demande tour à tour à chaque joueur de saisir un numéro de colonne entre 1 et 7 ;
   — ajoute un jeton du joueur correspondant dans la colonne indiquée si celle ci n'est pas vide, ou demande de saisir une autre colonne autrement ;
   — continue à demander à tour de rôle quel coup jouer jusqu'à ce qu'un joueur gagne la partie ou que la grille soit remplie ;
   — affiche le vainqueur de la partie.

6 Mise en place de l'intelligence artificielle

Nous mettrons en place 3 types d'intelligences artificielles :

1. La première intelligence artificielle se contentera de jouer au hasard un coup valide
2. La seconde intelligence artificielle essayera de toujours trouver le meilleur coup à jouer.
3. La troisième intelligence artificielle sera d'un niveau intermédiaire, qu'il sera possible de calibrer en fonction d'un niveau de difficulté.

6.1 mise en place d'une intelligence artificielle sur la base d'un modèle aléatoire

Ajouter les élements à votre programme pour qu'un joueur puisse choisir de jouer soit contre un être humain, soit contre l'ordinateur. Lorsque l'ordinateur est choisi, définissez la fonction `IARandomChoisirCoup()` qui prend en paramètre une grille, et renvoie une position de colonne aléatoire. Cette position correspond au coup que jouera l'ordinateur. Intégrez cette fonction dans la boucle de jeu lorsque l'ordinateur est sélectionné comme joueur.

6.2 mise en place d'une intelligence artificielle avancée

La définition d'une intelligence artificielle via la méthode min-max fera l'objet d'une présentation en cours afin que vous puissiez les incorporer à votre code. La mise en place de cette intelligence artificielle passe par les éléments suivants :

— Définition d'une fonction `evaluate` qui prend en paramètre un plateau de jeu, et renvoie un entier. Cette fonction va évaluer une grille et renvoyer un entier entre -1000 et +1000. Elle renverra -1000 si la grille est gagnante pour le joueur 1, ou +1000 si elle est gagnante pour le joueur 2. Dans les autres configurations, il vous faudra déterminer par vous même si la grille est plus favorable au joueur 1, auquel cas la valeur à retourner doit être d'autant plus négative que la grille est favorable au joueur 1, ou plus favorable au joueur 2 (la valeur à renvoyer est alors positive). Par exemple, si 3 jetons uniquement du joueur 1 sont alignés, cette configuration est favorable au joueur 1 s'il y a la place de mettre un 4eme jeton (s'il y a une cellule de libre de part et d'autre, la configuration est encore plus favorable...). A vous de déterminer les règles pour évaluer une grille.

— utiliser la stratégie min-max en créant un arbre de décision (de profondeur bornée) : pour une grille donnée G, on repère les colonnes dans lesquelles le joueur suivant peut jouer. Pour chacune de ces possibilités, on génère un noeud fils dans l'arbre. Pour chacun de ces fils, on réitère l'opération, en prenant en compte cette fois l'autre joueur. Et ainsi de suite jusqu'à atteindre la taille de l'arbre que l'on souhaite. Sur les noeuds fils, on utilise enfin la fonction `evaluate` pour savoir si la configuration est préférable au joueur 1 ou au joueur 2. Enfin on fait remonter les valeurs des fils à leur parent immédiat. Le parent va alors être affecté d'une valeur minimale de ses fils, ou maximale, en fonction de sa parité dans la profondeur de l'arbre (cad si son niveau correspond au joueur 1, ou au joueur 2). L'opération est alors réitérée jusqu'à la racine de l'arbre

— L'objectif est d'avoir, au niveau de la racine de l'arbre, qui correspond à la grille G, un ensemble de choix de colonnes possibles, chacun étant associé à une valeur, qui aura été calculée à partir de chaque sous-arbre. Il suffira alors de choisir le coup avec la valeur la plus adéquate pour le joueur représenté par l'ordinateur.

6.3 mise en place d'une intelligence artificielle de niveau intermédiaire

A partir d'une IA optimale, deux stratégies sont possibles pour faire varier le niveau :

1. ajouter une dose d'aléatoire pour faire varier le niveau, en sélectionnant les stratégies selon une probabilité donnée
2. Ne pas explorer tout l'arbre mais s'arrêter à certains niveaux de profondeur

Ces deux stratégies feront l'objet d'une présentation en cours afin que vous puissiez les incorporer à votre code.

7 Définition d'une interface graphique (optionnel)

A partir des éléments qui vous auront été communiqués en cours, définissez une interface graphique pour votre jeu.

Solutions TD

7.1 Some Solutions