DIOP Khady, TEBEBU Esrom et WANG Boyan
L3 MIAGE
Groupe n°1

Puissance4

Le 18 novembre 2022

Dans le cadre du projet en semaines d'études et développement, nous nous sommes amenés à développer un simple jeu de Puissance4. Notre programme a qu'une seule classe composé en deux parties : une pour les méthodes et une autre pour l'affichage, i.e. le *main*. Notre classe utilise *Serializable* qui est utile pour la fonction de sauvegarde.

1. Les constantes

Tout d'abord, nous avons défini les constantes que nous allons utiliser tout au long du programme : un *Scanner* pour les *inputs* ; des *int* pour vide, joueur_1, joueur_2.

2. Les méthodes

2.1 Les méthodes utilisées tout au long du programme

Les méthodes utilisées tout au long du programme sont la fonction d'initialisation qui retourne *void* après avoir initialiser un tableau d'entier à deux dimensions (i.e. rempli chaque cellule par la constante vide), de sauvegarde qui retourne un *void*, et une qui permet de convertir—visuellement—l'*array* à deux dimension en grille en retournant un *void*. N.B. on a fait le choix que lorsque nous allons de droit à gauche sur la grille, nous allons de 0 à 6 sur l'*array*[ligne] et lorsque nous allons du haut vers le bas dans la grille, nous allons de 0 à 5 sur l'*array*. Pour la fonction de sauvegarde, nous avons utilisé l'*exception handling* en important au début du programme *IOException* et *FileNotFoundException*.

Pour permettre au jeu de se dérouler en vérifiant si le coup joué par un utilisateur humain est valide, nous avons créer une fonction qui retourne une booléenne. Tout d'abord, la fonction vérifie que le numéro de colonne choisie est valide, i.e. il est entre 0 et 6, puis si elle n'est pas pleine. Dans le cas contraire, on retourne *false*. Sinon, nous allons commencer du bas de la colonne jusqu'en haut pour trouver une cellule vide. Lorsqu'on trouve une cellule vide, on la remplace par le « jeton » du joueur et le programme retourne *true*. Sinon, il retourne *false*.

Ensuite, nous avons une fonction de comptage qui retourne un entier et qui permet de « scanner » la matrice dans quatre directions en comptant si le même « jeton » réapparaît plus d'une fois de suite : horizontalement, diagonalement de bas en haut, diagonalement de haut vers le haut allant de gauche à droite ; et verticalement allant de haut vers le bas. Et, en utilisant cette fonction qui compte, nous allons déterminer si un joueur à gagner—i.e. on a compter le même « jeton » plus de 3 fois—mais nous devons faire attention que nous laissons pas la fonction de comptage dépasser les bords du tableau à deux dimensions (ou la matrice 6 x 7) en imposant des conditions sur la colonne et la ligne. Cette fonction qui détermine s'il y a un gagnant retourne une booléenne.

Puis, nous avons une fonction qui vérifie si la matrice est pleine—et par conséquent c'est un match nul—en retournant une booléenne.

Enfin, nous avons une fonction qui simule le lancement d'une pièce équilibrée en utilisant *Math.random( )* et en retournant un entier qui nous permettra de déterminer qui commence.

2.2 La méthode utilisée pour le mode multijoueur

Pour le mode multijoueur, on a tout simplement créer une fonction qui retourne *void*. Cette méthode simplement reçoit un input des joueurs via la constante *Scanner* que nous avons définis au début (cf. 1.) et tant que la colonne choisie par le joueur est invalide—que nous vérifions avec la méthode (que nous avons défini au 2.1) permettant au jeu de se dérouler correctement—nous allons demander au joueur de choisir de nouveau sa colonne. Lorsque la colonne choisie est validée, cela veut dire que le « jeton » du joueur se placera dans la colonne qu'il a choisi dans le *main* du multijoueur.

2.3 Les méthodes utilisées pour jouer contre l'ordinateur

2.3.1 La méthode utilisée pour jouer contre l'ordinateur quelque soit le niveau

Pour alterner les tours entre joueur_1 et joueur_2, nous avons fait la décision que le joueur_2 est toujours l'ordinateur. De même que la méthode pour multijoueur (cf. 2.2) nous faisons une boucle tant que pour remplacer une cellule vide dans la colonne choisie par un le « jeton » de l'ordinateur tout en vérifiant que la colonne n'est pas pleine.

2.3.2 La méthode utilisée pour jouer contre un ordinateur facile

À ce niveau, nous n'écrivons pas un algorithme utilisant une heuristique pour gagner. Tout simplement, l'ordinateur choisie une colonne aléatoirement—pourvue qu'elle n'est pas pleine— dans le main de l'ordinateur facile.

2.3.3 Les méthodes utilisées pour jouer contre un ordinateur moyen

Nous utilisons d'abord une fonction qui détermine qu'elle colonne choisir en retournant un entier compris entre 0 et 6. Nous suivons les heuristiques suivantes : en premier, l'ordinateur va tenter de remplir la première ligne de la colonne du centre si elle est vide ; ensuite, l'ordinateur va privilégier les colonnes du milieu, i.e. de 2 à 4 ; puis, si les angles sont vides, l'ordinateur va tenter de remplir les angles ; et, enfin, l'ordinateur choisira les coins, i.e. 2 et 5.

Ensuite, nous utilisons une fonction qui retourne *void* qui fait une boucle tant que le coup joué par l'ordinateur n'est pas valide. Par la suite, nous allons passer cette fonction dans le *main* pour l'ordinateur moyen.

2.3.4 Les méthodes utilisées pour jouer contre un ordinateur difficile

De même qu'avec l'ordinateur de niveau moyen, nous construisons d'abord une fonction qui retourne un entier pour choisir la colonne. Les heuristiques que nous utilisons est la suivante : tout d'abord si la cellule à la ligne 5 et à la colonne 3 est vide, nous la remplaçons par le « jeton » de l'ordinateur ; ensuite, l'ordinateur vérifie qu'il y a une victoire imminente avec la fonction de comptage (cf. 2.1) pour voir si il y a trois de ces jetons de suites et une des cellules adjacentes est vide, si c'est le cas, il choisie cette colonne ; avec la méthode, l'ordinateur veille que le joueur n'a pas une victoire imminente également, et si c'est le cas, l'ordinateur bloque la cellule vide adjacente ; puis l'ordinateur de niveau difficile utilise les mêmes heuristiques que l'ordinateur de niveau moyen (cf. 2.3.3).

Puis, nous utilisant une fonction qui retourne *void* qui fait une boucle tant que le coup joué par l'ordinateur est invalide. Par la suite, nous allons passer cette fonction dans le *main* pour l'ordinateur de niveau difficile.

3. Les *main*s

Pour simplifier le *main* principal, nous avons crées des *mains* pour un jeu à multijoueur ou contre ordinateur—facile, moyen et difficile—. Tous ces *mains* utilisent *IOException, FileNotFoundException* et *ClassNotFoundException* et utilisent la méthode de sauvegarde pour arrêter le jeu à tout moment si l'utilisateur entre 7687 et donne le choix de sauvegarder en nommant le fichier. Et avant de commencer un jeu, donne l'option d'accéder à un jeu déjà enregistré.

3.1 Le *main* pour le multijoueur

Le *main* du multijoueur permet de jouer dans tous les cas où nous allons chercher un *input* humain, donc il sera également utilisé dans les mains pour les jeux contre ordinateur. En premier, nous déclarons, initialisons et affichons la matrice. Ensuite, nous faisons un boucle tant qu'il n'y a pas de vainqueur et la grille n'est pas pleine. Dans cette boucle, nous vérifions à qui est le tour, et après qu'un joueur fait un coup valide, nous affichons la grille et alternons entre les joueurs. S'il y a un vainqueur, nous annonçons qui a gagné. N.B. si le tour actuel est celui d'un joueur, ce joueur a perdu. S'il n'y a pas de vainqueur et la matrice est pleine, nous concluons que c'est un match nul.

3.2 Le *main* pour l'ordinateur facile, moyen et difficile

Nous avons la même structure que le *main* pour le multijoueur sauf que nous déterminons au début que si c'est le joueur_1, on utilise la méthode pour multijoueur ; sinon, on utilise la méthode pour jouer contre ordinateur facile, moyen ou difficile.

4. Le *main* principal

Le *main* principal demande à l'utilisateur, à l'aide de *Scanner*, s'il veut jouer contre un autre joueur ou contre l'ordinateur. Si le jouer décide de jouer contre l'ordinateur, le programme demande de choisir le niveau. Si le joueur à choisi le mode multijoueur, on utilise le *main* pour le multijoueur (cf. 3.1) sinon un des *mains* pour jouer contre l'ordinateur à différents niveaux (cf. 3.2).