Projet jeu POO Puissance 4

Introduction:

J’ai fait l’implémentation du jeu puissance 4. Nous verrons dans différentes parties les choix architecturaux de notre implémentation et des explications de chaque classe.

Notre programme est composé de 4 classes .java dans le fichier src et nous verrons aussi leur rôle de chacun.

Pour compiler : javac -d build src/*.java

Pour executer : java -cp build src.Prog_princ

Créations des différents classe.

Classe Cellule:

Dans les instructions on nous indique qu’une cellule ou un jeton possède une couleur mais aussi 4 voisins (haut, bas gauche et droite). Dans ce cas, il est évident de créer avec des énumérations des types voisins et couleurs.

Attribuer une couleur est simple c’est la stockée dans une variables de types couleur car c’est une seule valeur.

Comment alors attribuer à la cellule des voisins, vu qu’il sont nombreux ? On peut stocker les voisins plus spécifiquement leur référence dans des variables ou plus simplement dans un Enummap c’est ce que nous avions fait.

Après avoir fait cela il est important de noter que nous implémentons des méthodes qui attribue et retourne une couleur d’une cellule.

Mais aussi le but du jeu c’est de terminer le jeu si il y a un alignement de 4 couleur alors au lieu de regarder toute la matrice nous regardons lorsqu’un jeton est ajouté si il y a un alignement de quatre couleurs depuis ce jeton en utilisant la récurrence bien évidemment car une cellule connait la référence vers une autre cela réduit le champs de vérifications en une matrice plus petites en quelque sorte et augmente la performance.

Classe Matrice

Comme nous avons créé une classe cellule, le langage de java nous permet de créer un tableau de 2 dimensions de type cellule. De ce fait, chaque cellule de la matrice aura les mêmes propriétés de la classe cellule. Nous ajoutons également un autre tableau d'entier qui nous permet de savoir combien de case libre il y a dans chaque colonne. Quand nous construisons nous attribuons pas directement les voisins a chaque cellule car cela va mettre certain voisin a null alors qu'ils existent. Ainsi nous les attribuons donc en deuxième temps pendant la construction de la classe Matrice.

Après cela, si nous supposons que le jeu est terminé et si le joueur essaye d'ajouter des jetons dans des colonnes qui sont remplies, nous devons afficher message et attendre à ce qu'il mettent dans une colonne non remplie. Mais aussi il y a un autre cas à gérer c'est quand l'utilisateur met un jeton dans une colonne qui n'existe pas alors dans ce cas aussi nous devons afficher un message et attendre à ce qu'il mettent dans la bonne colonne.

L'affichage de la matrice ce fait comme un affichage d'un tableau a deux dimensions de type entier sauf dans notre cas si la couleur de la cellule est bleu nous affichons un jetons bleu (Unicode jetons bleu) si c'est rouge un jetons rouge est si c'est blanc qui veut dire qu'il y a une case libre on affiche un carrer tout en utilisons les symboles unicode.

Prog_princ

Dans le programme principale nous demandons à l'utilisateur s' il veut commencer un nouveau jeu, dans ce cas nous initialisant les variables avec les entrées des joueurs, comme les nom du joueur 1 et 2 et aussi la couleur choisie de jeton choisie par les joueurs. Sinon on demande aussi si les joueurs veulent continuer un jeu déjà sauvegardés ainsi s' il y a déjà un jeu sauvegarder nous le chargeons sinon en recommence du début. En fonction de ces deux choix nous affichons la matrice est nous commençons le jeu, le joueur rentre une colonne auquel il veut ajouter nous vérifions à ce que cette colonne existe et a ce qu'elle soit non remplie grâce à notre méthode MyException. Ensuite nous affichons la matrice et nous regardons si le jeu est terminé, si quelqu'un a gagné , si c'était un match nul ou si le jeu n'est pas terminé en fonction de ces cas nous affichons un message correspondant si le jeu est terminé.

Pendant le jeu nous donnons la possibilité à l’utilisateur de sauvegarder le jeu en entrant 999 .

Classe File_game

La classe File_game va avoir deux méthodes principales, une qui va sauvegarder et une qui va charger le jeu. Le constructeur nous crée un fichier pour initialiser les variables de la classe.
Regardons ensemble la partie qui va sauvegarder le jeu.
Cette méthode en premier temps reçoit le programme principal ensuite elle initialise un file reader puis un buffer reader, on peut sauvegarder avec un file reader mais le faire avec un buffer reader est plus efficace (source : Oracle). Quand nous sauvegardons le jeu dans le fichier créé, nous écrivons en premier temps le nom du joueur 1 ensuite dans une autre ligne le nom du joueur 2 , puis dans une autre ligne la couleur en entier (si c’est 1 player1 = bleu et player2 = rouge ou sinon player1= rouge et player2 = bleu) choisi par le joueur auquel est son tour actuelle dans le jeu. Dans une autre ligne on ajoute le tour du joueur en entier (si c’est 0 alors c’est le tour du player 1 sinon le tour du player 2). Une fois cela fait, à la ligne suivante en vas parcourir toute la matrice contenue dans le programme principale récupérer en paramètre. On vas mettre dans le fichier un 0 si la couleur de la cellule est blanc, un 1 si la couleur est bleu et un 2 si elle est rouge. Dans cette partie nous n’avons aucun cas à traiter.
La méthode qui charge le jeu va lire dans un fichier existant, tout d’abord il est astucieux de regarder si le fichier existe ou s'il est vide, sinon nous affichons un message et nous nous recommençons le jeu du début.

Si par contre le fichier existe et il n'est pas vide, nous rappelons que nous recevons le programme principale en paramètre nous pouvons donc instancier touts les variables comme le nom du player 1 et 2, le tour du joueur, sa couleur et la matrice en parcourant tout le fichier jusqu'à la fin est quand on trouve 0, 1 et 2 on instancie chaque cellule de la matrice correspondent avec une couleur.