Projet de Java
Puissance 4

Rémi Synave

Préliminaires

Durant ce projet, vous aurez à développer un puissance 4 en mode graphique pour un ou deux joueurs :
`https://fr.wikipedia.org/wiki/Puissance_4`

Pour réaliser ce projet, il est conseillé de suivre le sujet. Toutefois, si vous vous sentez assez à l'aise avec le langage, vous pouvez réaliser ce projet comme bon vous semble (sauf pour la dernière partie du sujet).

Ce projet est découpé en 3 parties :
— Création du moteur du jeu.
— Création de l'interface graphique du jeu.
— Création d'une intelligence artificielle.

Contexte de travail

Ce projet a pour but de :
— Faire de l'algorithmique.
— Revoir les bases de java.
— Utiliser des bibliothèques.
— S'initier à l'Intelligence Artificielle.

Il est conseillé de travailler sous linux. Durant ce projet, vous aurez à utiliser une bibliothèque graphique : MG2D. Les instructions d'installation de la bibliothèque sont fournies pour linux mais peuvent facilement être adpatées pour Windows (ajout/modification des variables d'environnement).
Si vous travaillez sous windows, merci d'enregistrer les fichiers contenant du code dans l'encodage utf-8 ou de ne simplement pas utiliser d'accent particulier.

Préliminaire

La bibliothèque MG2D peut être téléchargée sur Youcan. Elle vous permettra de simplifier le développement de l'interface graphique. Avant de commencer le développement de l'interface graphique du projet, n'hésitez pas à l'installer et la prendre en main en faisant le TP de prise en main contenu dans l'archive. Vous pouvez également jeter un oeil aux démos. OneMission a été créé par un étudiant de SupTechnology !

Puissance 4, le jeu, les règles

L'objectif de ce projet est de développer un puissance 4 avec interface graphique et une intelligence artificielle. La figure 2 vous montre l'interface graphique minimale que vous devrez coder pour la fin du projet. Lors de la première partie de ce sujet, vous allez développer le moteur du jeu. La seconde partie sera libre et consacrée à la création de l'interface graphique. Finalement, la dernière partie vous donnera des idées pour créer votre propre Intelligence Artificielle et l'intégrer au jeu.

FIGURE 1 – Jeu IRL.

Les règles de ce jeu sont très simples. Le jeu se compose d'une grille de 7 colonnes sur 6 lignes. La grille est verticale et les joueurs glissent, tour à tour, un pion de leur couleur dans une colonne. Le joueur gagne lorsqu'il a réussi à aligner 4 pions de sa couleur. L'alignement peut être horizontal, vertical ou même diagonal.

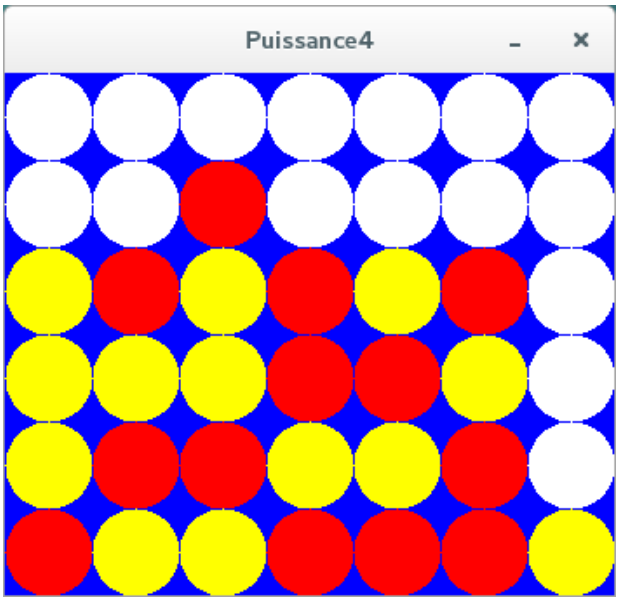


FIGURE 2 – Interface graphique minimale du jeu à la fin du projet.

Vous pouvez le tester en ligne à cette adresse :
`https://connect4.gamesolver.org/fr/`
Il est normal que vous perdiez sans arrêt. l'intelligence articielle y est très poussée.

Trève de bavargade ! Codage ! L'énoncé suivant est assez détaillé pour que vous puissiez arriver à un résultat minimal sans trop de peine.

Attention ! ! ! !
Vous allez développer une application graphique. Avant de vous lancer dans le développement, il est important de connaitre le système de coordonnées de la bibliothèque qui sera utilisée. En effet, avec MG2D, lorsque vous allez créer une fenêtre graphique, l'origine de la fenêtre (0,0) se trouvera **en bas** à gauche. Or, dans le terminal, lorsque vous afficherez votre grille, la case origine (0,0) se trouvera **en haut** à gauche. Si vous ne gérez pas cela, dans le terminal, la grille d'affichera donc à l'envers. Les pions vont remonter. Rien de grave mais il vaudrait mieux afficher la grille dans le bon sens si possible.

Partie 1 - Le moteur du jeu

La classe `Case`

Cette classe permettra de stocker l'état d'une case du plateau de jeu.
La classe doit contenir les attributs et méthodes suivants :

— Un attribut `contenu` permettant de savoir ce que contient la case (numéro du joueur la tenant ou 0 si la case est vide).

— Un constructeur par défaut qui affectera 0 au contenu.
— Un constructeur prenant un contenu spécifique en paramètre. Avant d'affecter le paramètre à l'attribut, vous le testerez pour vérifier qu'il vaut bien la valeur d'une case vide ou d'un joueur (0, 1 ou 2). Dans le cas contraire, un 0 sera affecté au contenu.
— Un constructeur par copie.
— Les accesseurs : getter et setter pour le contenu (paramètre du stter à tester).
— La méthode
`public String toString()`
Cette méthode retournera un espace pour une case vide, sinon le numéro du joueur la contrôlant.

La classe `Ligne`

Cette classe permettra de stocker une ligne de `Case`. Créez cette classe comportant :
— Un attribut de type tableau de `Case`.
— Un constructeur par défaut créant un tableau à une case dont le contenu sera fixé à 0.
— Un constructeur prenant en paramètre la taille d'une ligne qui sera remplie de 0.
— Un constructeur par copie.
— Une méthode `getX(int i)` retournant la i^{eme} `Case` de la ligne (l'objet contenu dans la i^{eme} case de la ligne - les indices commencent à 0).
— Une méthode `setX(int i, int valeur)` permettant d'affecter la `valeur` à la $i$ème case de la ligne. Vous ferez attention à la cohérence de la `valeur` par rapport au jeu. Est ce bien un numéro de joueur ou une valeur de case vide ? (0, 1 ou 2)
— Une méthode `setLigne(String valInit)` où `valInit` est une chaîne de caractères contenant des chiffres représentant les joueurs ou les cases vides. Cette méthode initialise la ligne avec ces valeurs. Attention à la longueur de `valInit` qui doit correspondre à celle de la ligne et au contenu qui doit être cohérent. Si une des contraintes n'est pas correcte, vous afficherez au moins un message d'erreur.
— Une méthode `getTaille()` qui retourne la taille de la ligne.
— La méthode `toString()`. Cette méthode retournera une chaîne de caractères symbolisant une ligne de cases.

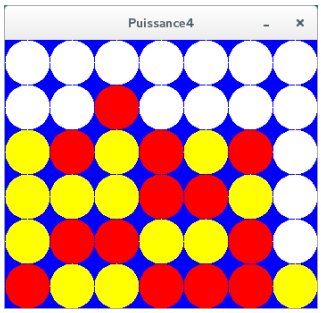
La classe `Plateau`

Cette classe permet de stocker un plateau complet de jeu qui est composé d'un tableau de `Ligne`. Créez cette classe comportant :
— Un attribut de type tableau de `Ligne`.
— Un constructeur par défaut créant un plateau d'une seule case (une case par une case) qui sera remplie par 0.
— Un constructeur prenant en paramètre un nombre de lignes et un nombre de colonnes. Le plateau sera rempli de 0.

— Un constructeur par copie.
— Un constructeur prenant en paramètre une chaine de caractères ayant la structure suivante :

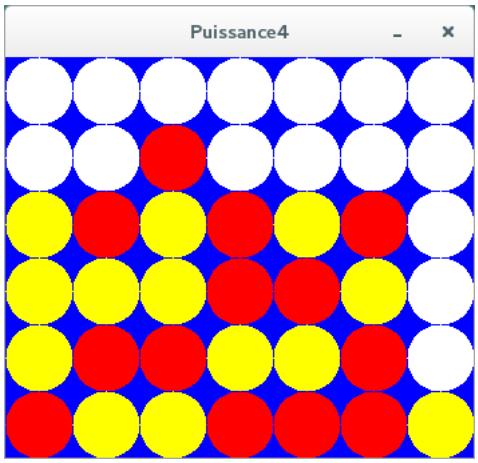
  — Le nombre de lignes sur le plateau
  — suivi du caractère 'x'
  — suivi du nombre de colonnes
  — suivi du caractère '-'
  — suivi de l'ensemble des valeurs sur les lignes. La première ligne suivi de la seconde, de la troisième etc. Les lignes sont remplies de 0 pour une case vide sinon du numéro du joueur.

  Exemple avec le plateau suivant :



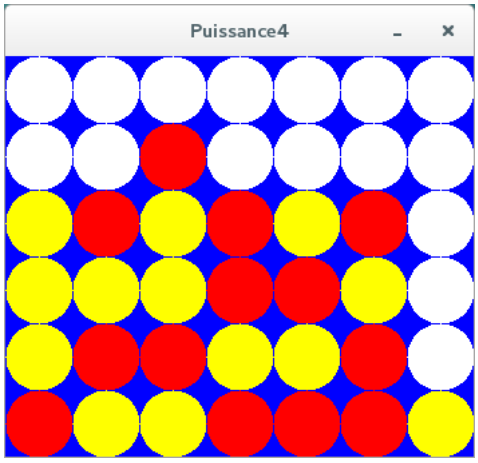
  La chaine de caractères correspondante sera la suivante (0 pour case vide, 1 pour jaune, 2 pour rouge) :
  `6x7-2112221122112011221012121200020000000000`
  6 lignes, 7 colonnes, 2112221 correspond à la première ligne (celle du bas dans l'interface graphique), 1221120 correspond à la seconde ligne, etc.
— Une méthode `getXY(int x, int y)` retournant la case se trouvant aux coordonnées (x, y). On considère que la case $(0, 0)$ se trouve en bas à gauche. x correspondant au numéro de la colonne et y au numéro de la ligne.
— Une méthode `colonnePleine(int col)` permettant de savoir une colonne est pleine ou si un pion peut y être déposé.
— Une méthode `ajoutPion(int numColonne, int numJoueur)` permettant l'ajout d'un pion dans une colonne.
— Une méthode `String toString()` retournant le plateau sous forme d'une chaine de caractères. Attention, la transformation en chaine de caractères doit se faire à l'envers pour que les lignes soient affichés dans le bon ordre (voir figure 3 et remarque au début du sujet).

— Une méthode `String toStringPourIA()` retournant une chaine de caractères au format décrit ci-dessus pour le constructeur (voir figure 4).




```
 ---------
|         |
|   2     |
|121212   |
|111221   |
|122112   |
|2112221  |
 ---------
```

FIGURE 3 – Illustration du retour de la méthode `toString()`. A gauche, un exemple de plateau. À droite, ce que doit retourner la méthode `toString()` - jaune est le joueur 1, rouge le 2. Attention ! Par défaut, la grille sera à l'envers.




```
6x7-2112221122112011221012121200020000000000
```

FIGURE 4 – Illustration du retour de la méthode `toStringPourIA()`. A gauche, un exemple de plateau. À droite, ce que doit retourner la méthode `toStringPourIA()` - jaune est le joueur 1, rouge le 2.

La méthode de fin de partie

Pour ce jeu, il va falloir détecter si un joueur a gagné. Ainsi, à chaque fois qu'un joueur aura placé un pion, il faudra vérifier s'il a gagné ou non. Pour ce faire, il y a plusieurs solutions possibles. Toutefois, une solution relativement simple est proposée ici.

La solution consiste à ajouter une méthode `public boolean gagne()` dans la classe `Ligne`. Cette méthode doit retourner `true` si 4 points de même couleur sont positionnés les uns à côté des autres (peu importe la couleur).

Cette méthode peut ensuite être utilisée dans l'algorithme général suivant :

1. Tester si le pion joué permet de créer un alignement horizontal de 4 pions est alors facile. Il suffit d'appeler la méthode `gagne` sur la ligne où le pion a été placé.
2. Tester si le pion joué permet de créer un alignement vertical de 4 pions est simple également. Il suffit de créer un nouvel objet temporaire de type `Ligne` ayant pour taille la hauteur du plateau, et d'y mettre comme valeur la valeur contenu dans la colonne. Il suffit ensuite d'appeler la méthode `gagne` sur cette ligne temporaire.
3. Idem pour les alignements diagonaux.

Si l'une des méthodes renvoie `true` alors le pion placé termine la partie.

Attention à la taille de la grille ! Il ne faut pas en sortir lors des différents tests.

Attention ! Il y a un cas particulier lorsque personne n'a gagné et que le plateau est complet.

La classe `Jeu`

Cette classe doit contenir un programme principal permettant la création d'un plateau de jeu et d'y jouer en mode terminal. Il vous faudra demander aux utilisateurs d'entrer la colonne dans laquelle ils veulent glisser un pion et afficher la nouvelle grille.

Les deux sections suivantes peuvent être traitées indépendamment et dans l'ordre que vous voulez, voire même en parallèle.

Partie 2 - L'interface graphique

Pour l'interface graphique, vous créerez une nouvelle classe `JeuGraphique`. Ce programme va afficher le plateau et attendra qu'un clic soit fait sur une colonne pour y déposer un pion du joueur suivant.

Vous êtes libre de créer l'interface graphique que vous voulez. Le plus simple est de créer une fenêtre contenant :

— un rectangle bleu couvrant la fenêtre entière

— des cercles blancs régulièrement répartis dont la couleur changera.

A chaque pion joué, le plateau est parcouru et la couleur des cercles est mise à jour.

Partie 3 - Intelligence Artificielle

Pour cette partie, merci de bien respecter les consignes au niveau du nom de la classe, du nom de la méthode, des paramètres et du type de retour !

Créez une classe que vous nommerez `IAVotreNom`. Par exemple `IASynave` pour moi. Ajoutez y la méthode suivante :

```
public static int meilleurCoup(String plateau, int joueur)
```

à votre classe `Plateau` permettant de calculer le meilleur coup possible. Le plateau sera passé en paramètre sous la forme d'une chaine de caractères. Pour l'appel à cette méthode, cette chaine de caractères peut être générée par le retour de la méthode `toStringPourIA()`. Elle peut ensuite être utilisé dans la méthode pour créer un `Plateau` temporaire. Ce plateau facilitera la prise de décision grâce à toutes les méthodes accessibles. Le second paramètre et le numéro du joueur qui doit placer son pion.

Cette méthode retournera le numéro de la colonne où le joueur devrait jouer.

Plusieurs niveaux d'IA sont possibles :

- 1er niveau : IA qui va choisir une colonne aléatoirement (en fonction des colonnes libres).
- 2eme niveau : IA qui va vérifier si il peut gagner et jouer le bon coup. Sinon aléatoire.
- 3eme niveau : IA qui va vérifier si il peut gagner et jouer le coup. Sinon vérifier s'il peut aligner 3 pions et jouer le coup. Sinon aléatoire.

Pour une intelligence artificielle avancée, il est possible, pour un plateau donné, d'évaluer les "chances" de gagner d'un joueur. Il s'agit donc ici de "noter" une grille du point de vue d'un joueur.
Lorsque cette méthode est disponible, l'IA peut alors tester toutes les possibilités et évaluer la meilleure future position. (algo glouton)

Pour une IA encore plus poussée, il est possible de simuler plusieurs coups de suite et de choisir la meilleure position en fonction des évaluations des futures grilles possibles. (algo min-max)