Objectif du projet : Création d'une application en Java offrant la possibilité de jouer à deux jeux, le jeu de Nim ou le jeu Puissance 4.

Le projet est à réaliser par trinôme. Il suivra un développement itératif et incrémental. Les itérations seront de deux semaines, les butées temporelles correspondront aux séances de TP au cours desquelles vous devrez faire une démonstration du fonctionnement de votre projet. Il est important d'avoir un projet opérationnel à présenter à chaque séance même s'il ne couvre pas toutes les fonctionnalités.

Le découpage en itérations sera le suivant :

Itération 1 : Développer une application permettant de jouer au jeu de Nim à deux joueurs.

Itération 2 : Développer une application permettant de jouer au jeu de puissance 4 à deux joueurs.

Itération 3 : Développer une application proposant deux jeux au choix, le jeu de Nim ou le jeu Puissance 4 à deux joueurs avec intégration de contraintes sur les deux jeux.

Itération 4 : Intégrer à l'application la possibilité de jouer à deux joueurs ou de jouer seul contre l'ordinateur en codant plusieurs stratégies pour l'ordinateur.

Ce sujet vous présente le détail de la quatrième itération.

Tout au long du projet, vous devrez toujours respecter une architecture MVC.

Itération 4 :

Vous devez développer une application Java permettant à deux joueurs humains de jouer au jeu de Nim ou au jeu de Puissance4 en mode console ou à un seul joueur contre l'ordinateur que nous nommerons « IA ».

Au démarrage de l'application, dans le `main`, on demandera au joueur à quel jeu il souhaite jouer, Nim ou Puissance4 et ensuite s'il souhaite jouer contre l'IA ou à deux joueurs humains. Dans le cas où le joueur joue contre l'IA, c'est toujours le joueur qui commence.

Pour le jeu de Nim, vous devrez coder deux stratégies de l'IA.

Stratégies pour le jeu de Nim :

L'IA utilisera une **stratégie gagnante** quand la partie se déroule sans contrainte sur le nombre maximum d'allumettes qu'on peut retirer et une stratégie **aléatoire** quand la partie se déroule avec contrainte sur le nombre maximum d'allumettes.

1 - Stratégie gagnante :

Cette stratégie permet à l'ordinateur d'identifier une situation gagnante, c'est à dire une situation qui lui permet d'être sûr de gagner, en la transformant en situation perdante pour l'adversaire.

Cette stratégie est basée sur la représentation binaire des entiers.

1 - Commençons par le cas où il n'y a pas de contrainte sur le nombre maximum d'allumettes autorisé.

Pour gagner, l'ordinateur doit laisser à l'adversaire une situation dans laquelle le OU exclusif du nombre d'allumettes de chacun des tas vaut 0. Appelons cette situation, situation paire.
L'ordinateur se retrouvera donc dans une situation gagnante si elle est impaire, c'est à dire que le OU exclusif du nombre d'allumettes de chacun des tas est différent de 0. Dans ce cas, il pourra toujours enlever des allumettes pour se ramener à une situation paire qui laissera l'adversaire en situation perdante.
De son côté, l'adversaire, en retirant des allumettes dans un des tas, transformera toujours cette situation paire en situation impaire permettant ainsi à l'ordinateur de retrouver une situation gagnante.

Quand c'est au tour de l'IA de jouer, il faut donc calculer le OU exclusif du nombre d'allumettes de chacun des tas. On obtient un entier nommé `resultatXor`.

- S'il est égal à 0, l'ordinateur n'est pas dans une situation gagnante. Il va donc jouer un coup quelconque. On fixe qu'il va retirer une allumette sur le premier tas contenant des allumettes.
- S'il est différent de 0, il est dans une situation gagnante.
  Dans ce cas, pour chaque tas, il doit rechercher le nombre d'allumettes qu'il devrait laisser dans ce tas pour que le joueur se retrouve en situation perdante. On sait qu'au moins un tas lui permet de le faire. Dès qu'il l'a trouvé, il retourne le coup correspondant.
  Pour calculer le nombre d'allumettes `nb` qu'il doit laisser dans le tas, il fera le OU exclusif du nombre d'allumettes du tas avec `resultatXor`. Si `nb` est stritement inférieur au nombre d'allumettes du tas, on peut en déduire le nombre d'allumettes à retirer pour laisser `nb` allumettes dans le tas.
  Vous trouverez ci-dessous un exemple pour illustrer l'identification d'une situation gagnante pour l'IA et le calcul du coup à jouer.

  Supposons qu'on ait défini une partie avec 3 tas et que le joueur 1 ait pris une allumette dans le deuxième tas (coup : 2 1).

  ```
    |
   | |
  | | | | |
  ```

  Calculons le OU exclusif du nombre d'allumettes de chacun des tas :
  resultatXor = 1 ⊕ 10 ⊕ 101 = 110
  resultatXor **≠ 0**.

Donc l'IA est dans une situation gagnante. Il peut donc jouer de manière à laisser une situation perdante pour le joueur 1.
Pour chacun des tas, on calcule le nombre d'allumettes à laisser pour que le joueur se retrouve en situation perdante.
Tas 1 : 1 ⊕ 110 = 111 Il faudrait laisser 6 allumettes dans ce tas. Il n'y en a qu'une . Impossible
Tas 2 : 10 ⊕ 110 = 100 Il faudrait laisser 4 allumettes dans ce tas. Il n'y en a que 2 . Impossible
Tas 3 : 101 ⊕ 110 = 11 Il faudrait laisser 3 allumettes dans ce tas. Il y en a 5. Donc l'IA va retirer 2 allumettes dans le tas 3. Il joue 3 2.

```
  |
 | |
| | |
```

Calculons le OU exclusif du nombre d'allumettes de chacun des tas :
resultatXor = 1 ⊕ 10 ⊕ 11 = 0
Le joueur 1 est en situation perdante. Quelque soit son coup, il laissera une situation gagnante à l'IA.

Remarque : le OU exclusif en Java se fait avec le symbole ^. La convertion des entiers en représentation binaire est automatique. On manipule directement les entiers sans avoir besoin de passer par leur représnetation binaire.

2 - Stratégie aléatoire :

Dans ce cas, on commence par calculer tous les coups possibles que l'IA peut jouer.

Une fois qu'on a calculé tous les coups possibles, il faut en choisir un au hasard.

Pour cela vous pourrez utiliser la classe `Random`. Le constructeur sans paramètre de cette classe permet de créer un générateur de nombres aléatoires. La méthode `public int nextInt (int bound)` retourne un entier choisi aléatoirement entre `0` (inclus) et `bound` (exclus).

Stratégie pour le jeu Puissance4 :

L'IA utilisera la stratégie suivante :

L'IA a les jetons jaunes. Si l'IA peut faire une rotation et qu'une rotation permet de gagner, l'IA la joue. Sinon, pour chaque case vide correspondant à un coup possible, l'IA calcule combien au max elle peut aligner de jetons en mettant un jeton de couleur jaune et combien l'adversaire peut au max aligner en mettant un jeton de couleur rouge.

On attribut un score à chaque coup possible de la manière suivante :

7 si l'IA peut aligner 4 sinon

6 si l'adversaire peut aligner 4 sinon

5 si l'IA peut aligner 3 sinon

4 si l'adversaire peut aligner 3 sinon

3 si l'IA peut aligner 2 sinon

2 si l'adversaire peut aligner 2

1 sinon

L'IA associe donc à chaque score compris entre 1 et 7 une liste de coups possibles. Elle parcourt ensuite ces coups possibles en commençant par celui ayant le score le plus élevé. Pour chaque coup X, elle teste si en le jouant l'adversaire pourrait gagner en faisant de suite une rotation. Si ce n'est pas possible alors l'IA joue le coup X sinon elle refait la même chose avec le coup ayant le second meilleur score et ainsi de suite.

Bonus : Si vous avez du temps, vous pouvez ajouter une stratégie supplémentaire pour le jeu de Puissance 4. Vous aurez normalement choisi un design pattern qui vous permettra de le faire facilement.

Vous pouvez imaginer une autre stratégie ou coder la stratégie optimisée ci-dessous.

L'IA utilisera une **stratégie simple** quand la partie se déroule avec la possibilité de jouer avec des rotations et **l'autre une stratégie** quand la partie se déroule sans rotation.

Idée de seconde stratégie: stratégie optimisée

Cette stratégie est une optimisation de la stratégie simple.

L'optimisation est basée sur le fait qu'il est inutile de jouer une coup qui permet d'aligner 2 ou 3 jetons si on sait déjà qu'on ne pourra pas en aligner 4 car les positions sont déjà occupées par l'adversaire.

Donc avant d'ajouter un coup à un score de 5 ou 3, on vérifiera qu'il est possible d'en aligner 4.

Vous ajouterez en plus un tirage aléatoire de l'ordre dans lequel seront considérés les coups de même score dans la stratégie précédente.

Travail à faire :

Première partie

1. Compléter votre diagramme de classes de conception de l'itération3 de manière à intégrer la possibilité de joueur contre l'IA. Essayer de proposer une conception qui permette facilement d'ajouter de nouvelles stratégies de l'IA pour chacun des jeux.

2. Coder et tester votre application.

Vous déposerez sur CELENE dans le dépôt correspondant à l'itération 4 un dossier contenant le diagramme de classe de conception et le code de l'application.