Projet d'Algorithmique : « Jeu des guerriers »

1. Présentation des jeux

1.1 Jeu de base

Vous allez programmer un jeu qui se joue avec un plateau, des pions et un dé.
Les pions sont des guerriers d'où le nom du projet : « Jeu des guerriers » !

Le jeu demande une configuration de départ, car les nombres de cases, de joueurs, de tours, de guerriers et de points de vie peuvent être choisis.

Dans l'exemple ci-dessous, le plateau compte 12 cases, il y 2 joueurs qui ont chacun 3 guerriers et la partie va commencer.

Les règles du jeu de base sont les suivantes :

1) Le nombre total de guerriers ne peut pas dépasser le nombre de cases
2) Tous les joueurs ont le même nombre de guerriers au début de la partie.
3) Tous les guerriers ont le même nombre de points de vie au début de la partie.
4) Au départ, on place un guerrier par case en commençant par la case 1. On met d'abord le premier guerrier de chaque joueur en commençant par celui du premier joueur, et ainsi de suite jusqu'à ce que tous les guerriers soient placés.
5) C'est le joueur 1 qui commence la partie.

6) Un joueur ne peut déplacer qu'un seul guerrier par tour d'un nombre de case déterminé par le jet d'un dé à 6 faces non pipé.
7) Un joueur ne peut pas jouer avec le guerrier d'un autre joueur. S'il essaie de jouer avec le guerrier d'un autre joueur, il perd son tour de jeu.
8) Si un joueur clique sur case où il n'y a pas de guerrier, il perd son tour de jeu.
9) Il ne peut jamais y avoir plus d'un guerrier par case :
   Si le guerrier qui est joué arrive sur une case déjà occupée par un autre guerrier, un assaut s'engage entre les 2 guerriers, même s'ils sont du même camp.
   Au terme du combat, chaque guerrier aura perdu des points de vie. Il ne faut pas simuler un duel avec plusieurs frappes, le nombre de points de vie perdu par chaque guerrier est déterminé par un jet d'un dé à 6 faces.
   Il y a quatre situations possibles :
   a) Les deux guerriers n'ont plus de points de vie et sont éliminés du jeu
   b) L'un des guerriers n'a plus de point de vie, il est éliminé et c'est l'autre qui reste sur la case.
   c) Les deux guerriers sont en vie et le guerrier qui est joué a fait perdre plus de points de vie que le guerrier attaqué. Alors le guerrier attaqué reste sur la case qu'il occupait et l'autre avance jusqu'à la première case disponible. La recherche commençant à la case après celle du guerrier attaqué.
   d) Les deux guerriers sont en vie et le guerrier qui est joué a fait perdre le même nombre ou moins de points de vie que le guerrier attaqué. Alors les 2 guerriers restent sur leur case respective.
10) Un guerrier a fait un tour quand il repasse sur la case 1.
11) Un joueur gagne soit quand un de ses guerriers a fait le nombre de tours demandés au départ soit s'il est le dernier joueur qui a encore des guerriers en jeu.
12) Le jeu s'arrête s'il y a un gagnant ou s'il n'y a plus de guerriers en jeu.

1.2 Jeu simplifié

Nous vous demandons de commencer par implémenter le jeu de base simplifié pour vous familiariser avec les classes et l'interface graphique.
Pour ce jeu, il ne faut pas afficher de classements.
Ce jeu ne sera ni testé ni coté !

Les règles du jeu simplifié sont les suivantes :
1) Il y a 2 joueurs au début de la partie.
2) Chaque joueur a 3 guerriers au début de la partie.
3) Au départ, on place un guerrier par case en commençant par la case 1. On met d'abord le premier guerrier de chaque joueur en commençant par celui du premier joueur, et ainsi de suite jusqu'à ce que tous les guerriers soient placés.
4) C'est le joueur 1 qui commence la partie.
5) Un joueur ne peut déplacer qu'un seul guerrier par tour d'un nombre de case déterminé par le jet d'un dé à 6 faces non pipé.

6) Un joueur ne peut pas jouer avec le guerrier d'un autre joueur. S'il essaie de jouer avec le guerrier d'un autre joueur, il perd son tour de jeu.
7) Si un joueur clique sur case où il n'y a pas de guerrier, il perd son tour de jeu.
8) Il ne peut jamais y avoir plus d'un guerrier par case :
   Si le guerrier qui est joué arrive sur une case déjà occupée par un autre guerrier, il prend sa place. Le guerrier qui y était est retiré du jeu.
9) Un joueur gagne soit quand un de ses guerriers est repassé par la case de départ, soit s'il est le dernier joueur qui a encore des guerriers en jeu.
10) Le jeu s'arrête s'il y a un gagnant.

1.3 Jeu amélioré

Une fois le jeu de base fonctionnel, nous vous demanderons un troisième jeu.
Pour ce jeu, vous modifierez l'une ou l'autre règle du jeu de base et vous en ajouterez d'autres.
Soyez créatif. Etonnez-vous et étonnez les professeurs.
Ce jeu ne sera pas coté mais les plus beaux jeux se verront récompensés par des points bonus.

2. Présentation des classes

Téléchargez le répertoire `JeuGuerriers` dans votre workspace. Dans le sous-répertoire `doc` vous trouverez le fichier index.html qui vous permettra de naviguer dans la javadoc des classes qui vous sont fournies.

2.1 Interface Graphique

Nous avons réuni dans une interface graphique toutes des méthodes qui seront utiles pour l'implémentation du jeu. Celle-ci est implémentée par la classe **`PlateauDeJeu.`**
Voici une représentation en UML de cette classe.

PlateauDeJeu
+PlateauDeJeu(nombreDeCases:int,nombreDeJoueurs:int,nombreDePionsParJoueur:int,grille:GrilleJeu) +afficherGuerrier(guerriers:Guerrier[]): void +afficherResultatDe(i:int) +afficherJoueur(joueur:Joueur): void +afficherGagnant(jou:Joueur): void +afficherFinDuJeu(): void +afficherInformation(texte:String): void +afficherInformation2(texte:String): void +actualiser(grille:GrilleJeu): void +jouer(): int

La classe **`PlateauDeJeu`** va permettre de visualiser le jeu.
Elle permet, grâce au constructeur, d'afficher le plateau de départ.
Pour modifier celui-ci, il faudra appeler la méthode `actualiser(GrilleJeu grille)`.
Pour jouer, le joueur va cliquer sur la case contenant le guerrier qu'il veut déplacer.
La méthode `jouer()` renvoie un entier qui correspond au numéro de la case sur laquelle celui-ci a cliqué.

Il existe plusieurs méthodes d'affichage. Celles-ci vont permettre d'afficher le joueur qui va jouer, le résultat du dé, le classement intermédiaire, le joueur gagnant et la fin du jeu.
Vous pouvez afficher vos propres messages grâce aux méthodes `afficherInformation(String texte)` et `afficherInformation2(String texte)`. Ces messages permettent de commenter le jeu. Voici quelques exemples : « Les 2 guerriers sont morts ! », « Plus que x tours ! », …

N'hésitez pas à consulter la Javadoc !

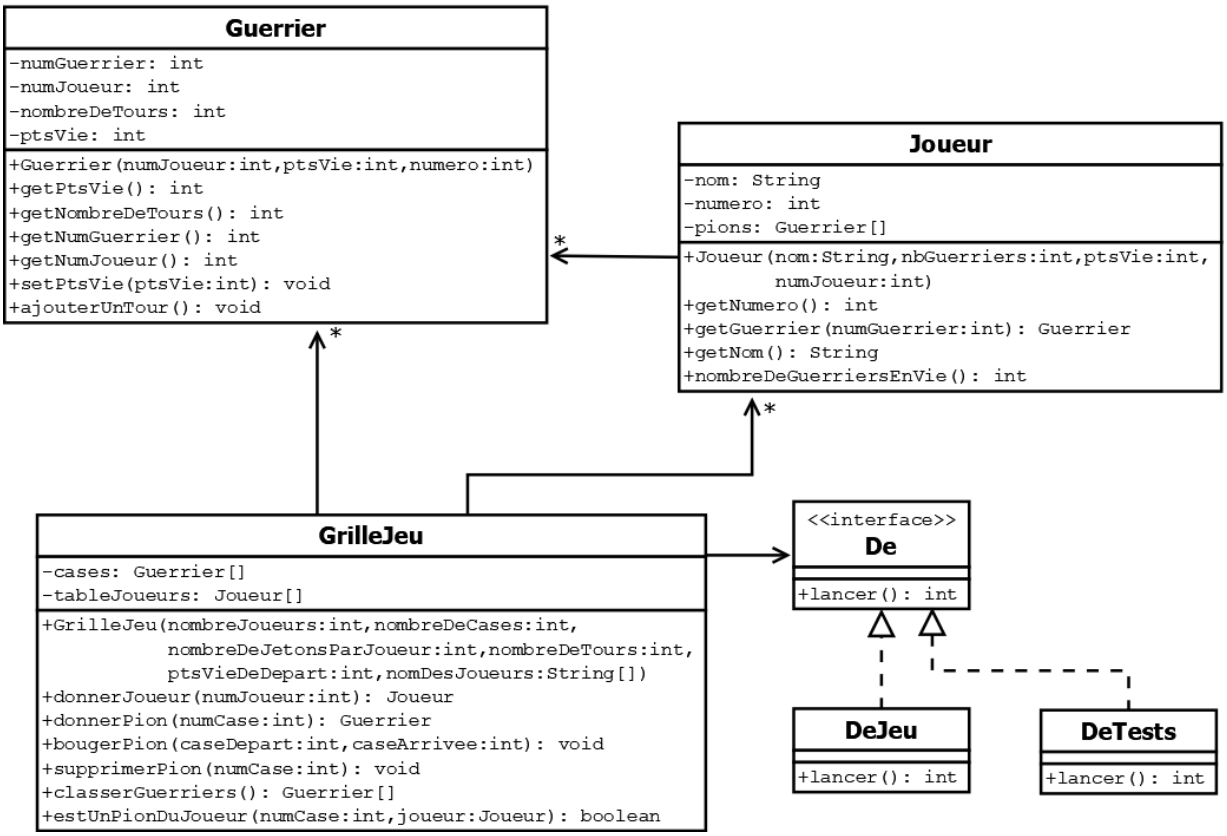
Les guerriers d'un joueur sont tous représentés par une même image. Ils portent des numéros différents.

La numérotation des cases, des joueurs et des guerriers d'un même joueur débute toujours à 1.

Attention :
La classe demande que le nombre de cases soit **pair**.
De plus elle permet un maximum 8 joueurs et de 10 guerriers par joueurs.

2.2 Les classes pour gérer l'état du jeu

Pour vous guider, nous fournissons l'UML suivant. Les méthodes et attributs présents sur cet UML doivent se retrouver dans vos classes.

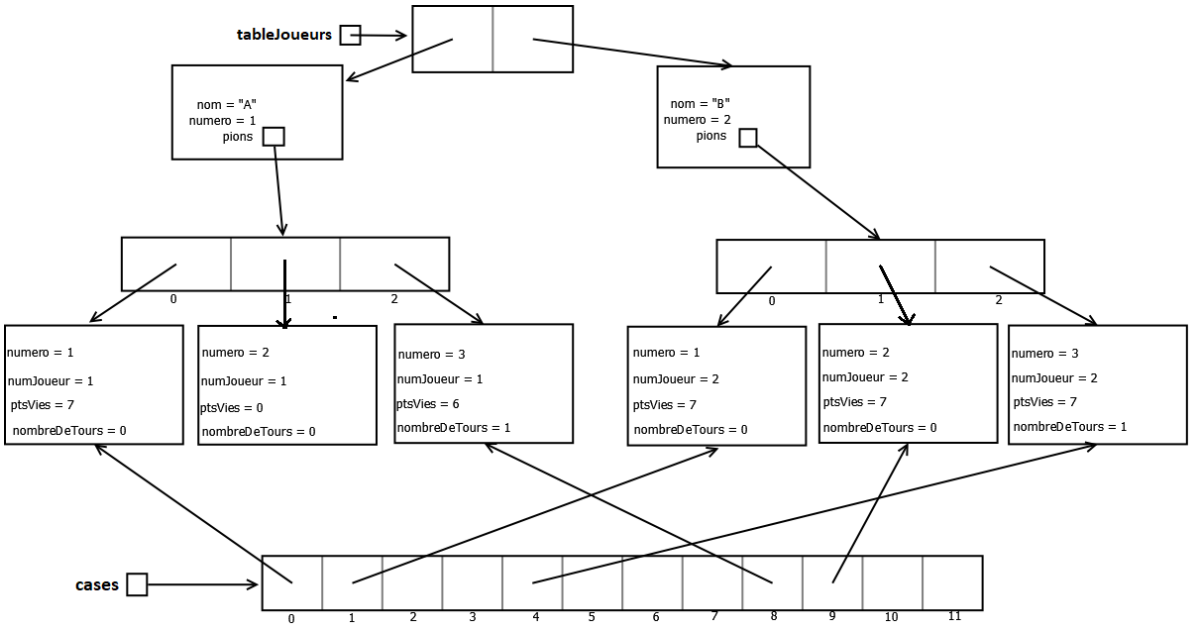


Voici quelques explications sur ces classes :

1) La classe `Guerrier` modélise un pion du jeu avec ses caractéristiques.
2) La classe `Joueur` modélise un joueur du jeu. Etant donné qu'un joueur possède plusieurs guerriers, cette classe a comme attribut une table de `Guerrier`.
3) La classe `GrilleJeu` représente l'état du jeu. Comme attributs, elle a notamment une table de `Joueur` qui contiendra les joueurs et une table de `Guerrier` qui représentera les cases du jeu.

Voici un exemple d'état du jeu pour une partie à 2 joueurs ayant chacun 3 guerriers.

Et voici la représentation en mémoire de ce jeu

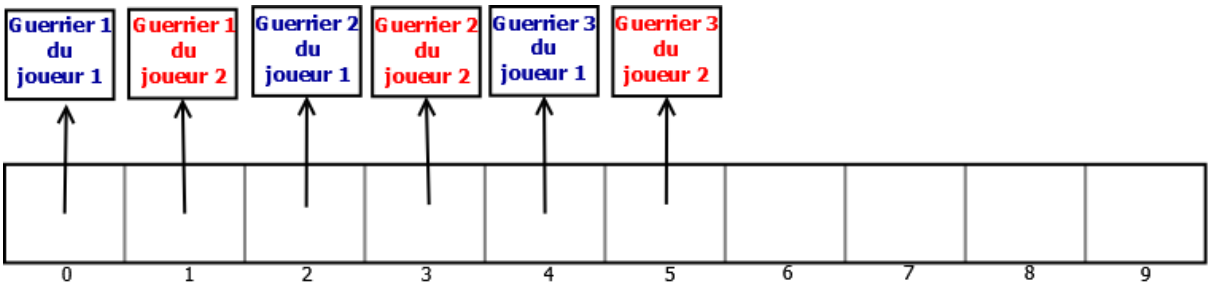


3. Travail à réaliser

Le déroulement du jeu se fera dans la classe `JeuGuerriers` qui contiendra une méthode `main()`. Cependant, toute la gestion de l'état du jeu devra se faire dans la classe `GrilleJeu` !
On vous demande donc de compléter les différentes classes en suivant les consignes suivantes.

3.1 Initialisation

Au commencement d'une partie, les guerriers doivent être disposés comme suit :

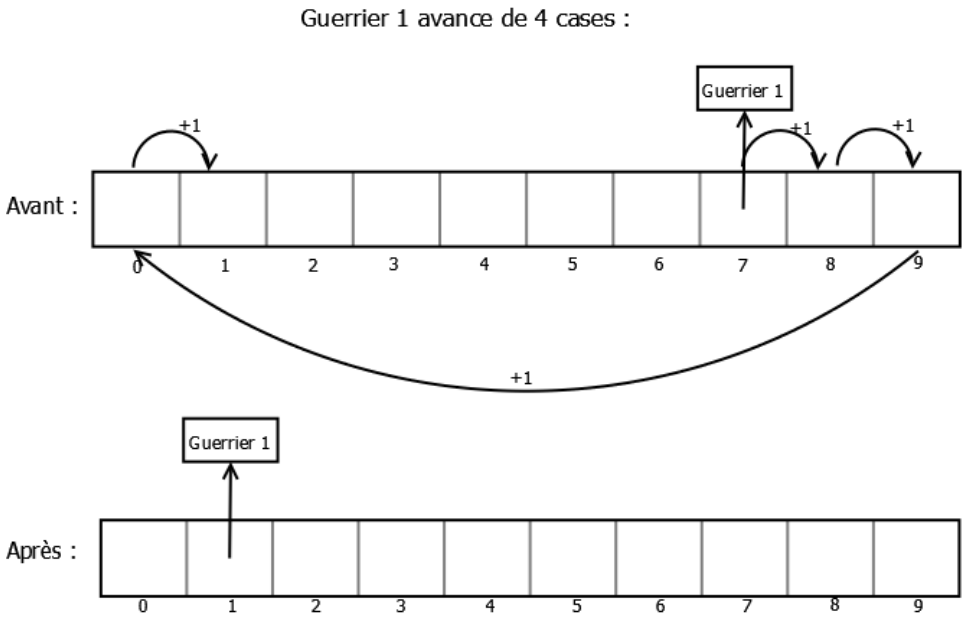


Il faut mettre d'abord le premier guerrier de chaque joueur en commençant par celui du premier joueur, puis mettre le second guerrier de chaque guerrier et ainsi de suite jusqu'à ce que tous les guerriers soient placés.

3.2 Gestion des tours

Etant donné que les guerriers doivent pouvoir faire plusieurs fois le tour du plateau, le tableau doit être circulaire. Cela signifie que quand un guerrier arrive au bout de la table, il doit revenir au début.

Voici un exemple où le guerrier se trouvant à l'indice 7 d'un tableau de 10 cases doit avancer de 4 cases :



On remarque que le guerrier au départ se trouvait à l'indice 7. La table étant de longueur 10, le guerrier se trouve à l'indice 1 après s'être déplacé de 4 cases, c'est-à-dire à l'indice (7+4) modulo 10.

3.3 Faire bouger un pion

Dès que le joueur a cliqué sur la case du guerrier qu'il veut faire avancer, vous récupérerez le numéro de la case grâce à la méthode `jouer()` de la classe `PlateauDeJeu`.

Vous modifierez l'état du jeu en utilisant les méthodes `bougerPion(int numeroCase)` et `supprimerPion(int numeroCase)` de la classe `GrilleJeu`.

Pensez également à modifier la version graphique de l'état du jeu en utilisant la méthode `actualiser(GrilleJeu grille)`.

3.4 Tri des guerriers

Il vous est demandé d'implémenter la méthode `classerGuerrier()`. Celle-ci va classer les guerriers **encore en jeu** de la manière suivante :
Les guerriers doivent être classés d'abord en fonction du nombre de tours parcourus (celui qui a parcouru le plus de tours d'abord) et ensuite en fonction de sa place dans la course (celui à qui il reste le moins de cases pour finir le tour d'abord).
Pour faire cela, nous vous demandons de procéder de la manière suivante :

1) Créer un nouvelle table du type `Guerrier[]` dont la taille est le nombre de guerriers toujours en vie.
2) Parcourir le tableau des cases du jeu de la première à la dernière case. Pour chaque case :

a) Regarder si elle contient un guerrier.
b) Si la case contient un guerrier, l'insérer dans la nouvelle table avant tous les guerriers ayant parcouru le même nombre de tours que lui.

3) Renvoyer la nouvelle table triée.

3.5 Dernières consignes

- Votre implémentation doit fonctionner pour n'importe quel nombre de cases, de joueurs et de guerriers par joueur.
  Toutes ces données devront être entrées au clavier par l'utilisateur au début de la méthode `main()` dans la classe `JeuGuerriers`. Tant les numéros des joueurs que ceux des guerriers commencent à 1.
- Toute modification de l'état du jeu doit être faite dans la classe `GrilleJeu`.
  Vous allez donc devoir ajouter des méthodes et des attributs.
  Par exemples : `donnerPremiereCaseLibreApres(int numeroCase)`
  Les tables `tableJoueurs` et `cases` sont `private` !
  Vous ne pouvez pas ajouter des méthodes qui renvoient ces tables.
  La méthode `main` de la classe `JeuGuerriers` sert uniquement à la gestion des tours de jeu, à l'affichage et à l'appel de méthodes de la classe `GrilleJeu` pour modifier l'état du jeu.
- Pensez à faire une faire une découpe de votre code en plusieurs méthodes pour rendre celui-ci plus lisible.
- N'apportez pas de modifications à la classe `PlateauDeJeu`, même pour le jeu amélioré.

3.6 Tests

Vous ne devez pas écrire de classes de tests. Mais testez bien votre jeu en l'essayant ! Pour ce faire, nous vous conseillons d'une part d'utiliser un objet de classe `DeTests` pour simuler le dé.
En effet cette classe permet de générer les lancés de dé dont on connaitra à l'avance le résultat et cela facilitera vos tests.
Dans le cadre de ce projet, on ne vous demande pas de tester systématiquement tous les paramètres
En semaine 13, nous vous demanderons individuellement d'écrire un quatrième jeu.
Il s'agira d'apporter quelques modifications à votre jeu de base pour que celui-ci corresponde aux règles du nouveau jeu demandé.
Les professeurs testeront en votre présence ce jeu.
En cas de problèmes, les professeurs vous demanderont d'y apporter des corrections.
Vous devez être capable d'expliquer votre code car cela peut vous être demandé.

4. Déroulement

Les séances d'exercices en algorithmique des semaines 11 et 12 sont consacrées à la réalisation de ce projet.
La **présence** est **obligatoire**.
Le travail est à réaliser par **groupe de deux étudiants**.

5. Soumission

On vous demande **1 soumission via Moodle.** Vous remettrez **une seule** soumission pour le groupe.
Vous soumettrez le répertoire (Eclipse) compressé (.zip) `JeuGuerriers`.
Vos 2 noms doivent apparaître dans toutes les classes que vous avez implémentées.
Soumission : **dimanche 16/12** (22h)
Attention ! Ce sera cette soumission qui sera utilisée pour les tests ! Vous ne pourrez donc plus modifier votre projet une fois soumis.

On vous demande **1 dossier « papier ».**
Vous remettrez ce dossier au début de la séance de tests en semaine 13**.**
Vous mettrez sur la page de garde vos noms, prénoms et le numéro de votre série.

Contenu :

- Les règles de votre jeu amélioré
- Ce qui fonctionne ou pas dans vos jeux
- Une conclusion reprenant vos impressions sur votre projet et son déroulement

Les règles de votre jeu s'adressent à quelqu'un ne connaissant pas le jeu de base et n'y ayant donc jamais joué. Elles doivent donc être compréhensibles et sans ambiguïté.

6. Evaluation

Voici à titre indicatif les critères d'évaluation :

Le code est fonctionnel

F1 Le code est correct.

F2 Le jeu suit bien ses règles.

F3 Le jeu tient compte des cas rares mais cependant possibles.

Le code est clair et documenté

C1 Le code est auto-commenté grâce à ses noms d'identifiants.

C2 Une bonne découpe en méthodes a été faite de façon à clarifier le code.

C3 La *JavaDoc* est présente et intéressante.

C4 Les passages difficiles à comprendre sont commentés de façon pertinente.

C5 Le code n'est pas inutilement compliqué.

L'Application est conviviale

G1 L'interface graphique a bien été utilisée.

G2 Les affichages permettent aux joueurs de bien suivre le jeu.

G3 Des messages apparaissent dans des situations rares mais possibles. Par exemple, : « Le jeu est terminé car il ne reste plus qu'un seul joueur en jeu ».

G4 Il faut toujours féliciter le gagnant.

Voici une liste contenant quelques causes de sanction.
Selon la gravité, le projet pourrait se voir attribuer une note équivalente à 0 ou ne pas être corrigé !

S1 Le **plagiat** (N'oubliez pas de mettre les auteurs dans chaque classe)

S2 Le travail a été fait seul sans la permission préalable d'un professeur.

S3 L'étudiant n'a pas été **présent** à toutes les séances d'exercices consacrées au projet.

S4 L'étudiant ne s'est pas montré actif lors de ces séances.

S5 L'étudiant n'a pas été présent lors des tests.

S6 La soumission n'a pas été faite dans le délai imparti.

S7 Le dossier a été remis en retard.