UFR ST - Besançon- L1 ST - S2

ALGO. ET PROG. OBJET

Projet TP 2022 - Un jeu de combat en mode démo
TP NOTÉ à rendre sur Moodle en Java

IMPORTANT.
Ce TP est à réaliser *en binôme* et sera évalué par une note. Il s'agit d'implanter un jeu de combat en grille, entre des personnages. Les affichages se feront sur la console : pas de graphisme. Le jeu est prévu pour se dérouler automatiquement en mode démo.
Le projet sera à déposer sur Moodle, au plus tard le **vendredi 20 mai**. Il est à coder et à rendre sous la forme d'un ensemble de fichiers en java. Une programmation par objets est exigée, mettant strictement en œuvre le concept d'encapsulation.

Les plagiats de devoirs seront systématiquement analysés et seront sanctionnés, tant pour les plagieurs que pour les plagiés.

1 Présentation rapide

Ce jeu consiste à créer deux équipes de personnages qui vont s'affronter au hasard de leurs rencontres sur une grille. Les personnages possèdent des caractéristiques dont les valeurs définissent leurs capacités lors des combats.

Les personnages se déplacent (quand c'est possible) d'une case à chaque tour, alternativement pour chaque équipe. Quand un personnage arrive sur une case qui contient un personnage de l'équipe adverse, il le combat. Il prendra sa place si le personnage combattu meurt, sinon il devra se déplacer sur une autre case voisine. Les règles précises de déplacement et de combat sont détaillées dans la suite du sujet, partie 2.

Le jeu est affiché sur la console (pas de graphisme), et se déroule en mode démo : les personnages évoluent tout seuls, au hasard. Le jeu prend fin quand l'une des équipes a perdu tous ses personnages. Il peut aussi arriver en principe que le jeu ne s'arrête pas, si les personnages restants ne peuvent pas se combattre l'un l'autre.

En option, il est possible de prévoir un mode interactif si vous le jugez pertinent. Vous pouvez également rajouter des règles pour assurer que le jeu terminera dans tous les cas.

2 Règles de fonctionnement détaillées

2.1 Personnages, combats et équipes

Un personnage est caractérisé par trois types de capacité : l'attaque, la défense et la santé. Chacune de ces capacités est elle-même composée de 2 capacités de base. Pour l'attaque il s'agit de l'habileté et de la force. Pour la défense il s'agit de l'armure et de l'esquive. Pour la santé il s'agit de la vie et de la résurrection. Les capacités de base sont définies au moyen de valeurs entières initialement strictement positives.

Les personnages sont amenés à se rencontrer en combat, pour une attaque de l'un contre l'autre. Une attaque du personnage A sur le personnage B est gérée de la façon suivante :

1. Si l'habileté de A est strictement supérieure à l'esquive de B moins le résultat du tirage au sort d'un nombre entier entre 1 et 6 alors l'attaque porte sinon l'attaque ne porte pas.
2. Si l'attaque porte on établit les dégâts en calculant la somme de la force de A et du tirage au sort d'un nombre entier entre 1 et 6, le tout moins l'armure de B.
3. Si la valeur de dégâts obtenue est strictement positive on retranche cette valeur à la vie de B. Dans le cas contraire, l'attaque n'a pas fait de dégât et la vie de B reste inchangée. Si la vie de B devient inférieure à 0, alors elle est ramenée à 0. On considère qu'un personnage dont la vie est (inférieure ou) égale à zéro est mort.

Les personnages sont répartis en deux équipes qui s'affrontent, et dont la taille initiale est identique (par exemple 5 personnages par équipe). Les personnages morts au cours d'un combat ne peuvent plus participer aux combats.

2.2 Grille et déplacements

Les personnages se déplacent sur une grille rectangulaire, d'une case à la fois, selon une direction qui leur est propre et qui peut être horizontale (1 case à droite ou à gauche), verticale (1 case en haut ou en bas) ou diagonale (1 case en haut ou en bas ET une case à droite ou à gauche). Chaque personnage calcule, avant de s'y déplacer, la prochaine case qu'il est supposé occuper par rapport à sa position actuelle et sa direction. La direction d'un personnage est amenée à changer au cours du temps en fonction du fait qu'il rencontre un bord, ou un autre personnage.

Un tour de jeu. Un tour de jeu consiste à faire se déplacer tous les personnages (vivants), alternativement d'une équipe puis de l'autre. L'alternance doit continuer à être respectée même lorsqu'une équipe a moins de personnages vivants que l'autre. Au cours d'un tour, les rencontres avec les bords de la grille ou avec d'autres personnages sont gérées comme indiqué ci-dessous.

Rencontre avec un bord. Quand il rencontre un bord, un personnage rebondit comme au billard : il se déplace d'une case dans la direction symétrique par rapport à la perpendiculaire du bord rencontré.

Rencontre avec un autre personnage. Quand un personnage A rencontre, dans la prochaine case qu'il est supposé occuper, un personnage B, les règles sont les suivantes.

1. Si le personnage B est de l'équipe adverse alors A attaque B.
   (a) Si A tue B alors B est retiré de la grille et A prend sa place, sans changer de direction.
   (b) Si B est toujours vivant après l'attaque, alors A se déplace sur une case voisine libre choisie aléatoirement, et sa direction devient celle correspondant à ce déplacement. Dans le cas où aucune case voisine n'est libre, A reste sur place mais sa direction change aléatoirement.
2. Si le personnage B est de la même équipe que A, alors A se déplace comme décrit ci-dessus après une attaque qui n'aurait pas tué son adversaire.

Fin du jeu. Le jeu prend fin dans le cas où une équipe n'a plus aucun personnage vivant. Il se peut que cette situation n'arrive jamais, auquel cas le jeu tourne jusqu'à ce qu'il soit interrompu « à la main ».

3 Visuels du jeu

Vous devrez afficher sur la console la grille ainsi que les personnages qui l'occupent. Les personnages d'une équipe pourront être visualisés par une lettre, distincte pour chaque personnage, et pouvant être par exemple en minuscule pour une équipe et en majuscule pour l'autre.

Voici un exemple possible de visualisation d'une grille avec cinq personnages d'une équipe (symboles de 'a' à 'e') et cinq personnages de l'autre équipe (symboles de 'A' à 'E').

```
+- - - - - - - - - - +
|D . . . . . . . . . |
|. . . . . c . . . . |
|. . . d . . . . . . |
|. . . . . . . . . . |
|. . . . . . . . . . |
|. . . . . . . . a . |
|. . . . . . A . . . |
|. . . . E . . . . . |
|. . . . . . . . e . |
|. . . . B C . b . . |
+- - - - - - - - - - +
```

Vous « enregistrerez » également lors d'un combat des messages indiquant qui attaque qui, en donnant les valeurs des capacités des deux personnages impliqués, et le résultat de l'attaque. Voici des exemples de messages (non directement consécutifs) qui pourraient être enregistrés :

```
[c: (H:1,F:3),(A:0,E:18),(V:6,R:13)] attaque [D: (H:0,F:17),(A:13,E:2),(V:16,R:6)]
  L'attaque ne porte pas

[B: (H:9,F:3),(A:19,E:17),(V:8,R:7)] attaque [e: (H:0,F:16),(A:18,E:8),(V:7,R:13)]
  L'attaque n'a pas fait de dégâts

[d: (H:18,F:2),(A:15,E:15),(V:6,R:1)] attaque [F: (H:6,F:6),(A:0,E:12),(V:10,R:19)]
  Après l'attaque [A: (H:6,F:6),(A:0,E:12),(V:4,R:19)]

[a: (H:14,F:14),(A:17,E:2),(V:4,R:19)] attaque [B: (H:9,F:3),(A:19,E:17),(V:1,R:7)]
  Après l'attaque [B: (H:9,F:3),(A:19,E:17),(V:0,R:7)] est mort
```

Pour « enregistrer » les messages, vous pouvez vous contenter de les afficher sur la console au moyen de Ecran.afficher() ou encore System.out.println(). Si vous ne souhaitez que ces messages s'intercalent avec les affichages de la grille, vous pouvez alternativement les stocker dans une ArrayList<String> (cf. CM du lundi 2 mai 2022, et annexe en fin de sujet), pour un affichage ultérieur.

4 Dimensionnement et initialisation du jeu

Vous êtes libres d'initialiser les paramètres du jeu (dimensions de la grille, capacités des personnages, positions et directions initiales, taille des équipes) selon votre convenance. Ces paramètres peuvent être soit :

— choisis au hasard,
— demandés à l'utilisateur,
— fixés en dur dans le code,
— (autre ?)

et la façon de les initialiser peut varier selon les paramètres.

5 Consignes de développement

Le projet est à développer en java, version 1.8 exclusivement. Les conventions de nommage (cf. Moodle) sur la casse des noms employés doit être respectée.

Une programmation *objet* est exigée. Cela signifie que les concepts clés du sujet devront être codés par des classes, telles que par exemple `Grille`, `Position`, `Equipe`, etc. A vous de choisir quelles sont les classes que vous jugez pertinentes pour le projet. Vous pourrez échanger avec votre enseignant sur la pertinence de telle ou telle classe.

Chaque classe est enregistrée dans son propre fichier java et le programme principal est codé dans une classe intitulée `Main`.

Les noms des classes, attributs, méthodes et variables doivent être soigneusement choisis et bien expliciter leur signification. Par exemple, si une méthode teste si un personnage est vivant, appelez-là par exemple `boolean estVivant()` (ou `isAlive()` si vous préférez l'anglais), plutôt que `vivant()` ou `statut()`, ou pire `test()`...

Votre code doit être commenté, et chaque méthode doit être spécifiée, soit en javadoc, soit de manière plus informelle comme dans les exemples sur Raphaello. Le nom de chacun des auteurs doit être indiqué en commentaire en haut de chaque classe.

Vous devez mettre en place l'encapsulation : les attributs sont *systématiquement privés*. Du coup vous avez à mettre en place des *getters* chaque fois que nécessaire. Les *setters* par contre ne sont pas systématiques, et ne devraient être utilisés que lorsque vous ne pouvez pas faire autrement.

Recommandations sur l'ordre des développements. Il est préférable de rendre un projet qui marche, même s'il est incomplet, plutôt qu'un projet où vous avez essayé de coder toutes les fonctionnalités mais qui au final ne marche pas.

Autrement dit ne vous attaquez pas à toutes les difficultés en même temps, mais rajoutez les petit à petit. Le conseil est de toujours disposer d'un projet qui compile et qui marche, que vous gardez soigneusement enregistré en l'état, pendant que vous ajoutez une nouvelle fonctionnalité dans la version en développement.

L'ordre dans lequel vous ajoutez les fonctionnalités est laissé à votre appréciation. N'hésitez pas à échanger avec votre enseignant. Voici quelques exemples possibles.

Exemple 1 : priorité au visuel. Définir la grille et l'afficher. Placer dessus des personnages. Les faire se déplacer (sans combattre). Gérer les collisions avec les bords et les autres personnages par simple changement de direction. Rajouter la gestion des cases voisines libres. Rajouter les combats. Etc.

Exemple 2 : priorité aux combats entre personnages. Définir les personnages indépendamment de la grille. Les faire se combattre entre équipes (un personnage affronte un autre tour à tour). Définir la grille et l'afficher. Rajouter aux personnages une position et une direction. Placer les personnages sur la grille et les faire se déplacer. Gérer les collisions dans le même ordre de priorité que ci-dessus. Etc.

6 Fonctionnalités non prioritaires

L'essentiel de l'évaluation portera sur l'implantation des fonctionnalités présentées ci-dessus. Vous pourrez compléter, une fois celles-ci termines, par les raffinements suivants.

Résurrection. Lorsqu'un personnage meurt (i.e. sa vie devient inférieure ou égale à 0) au cours d'un combat, il est susceptible de ressusciter si sa résurrection est suffisante :

1. si on peut prendre tout ou partie de la valeur de résurrection pour ramener la vie à une valeur minimum de 1, alors on rajoute à la vie une quantité aléatoire suffisante pour maintenir le personnage en vie, mais inférieure ou égale à la valeur de résurrection. Cette dernière est diminuée de la quantité prélevée pour le maintien en vie ;
2. si au contraire la résurrection n'est pas possible (car la quantité de résurrection est insuffisante) alors la vie de B est ramenée à 0.

Repos entre deux combats. Pour éviter qu'un personnage A qui vient d'attaquer un personnage B ne se retrouve tout de suite attaqué par ce personnage B, on peut laisser passer un tour entre deux attaques impliquant les mêmes personnages.

En option. Interactivité pour choisir le personnage à déplacer, fin du jeu garantie, ...

7 Travail à rendre

Vous déposerez sur Moodle une archive, exclusivement au format ZIP ou TGZ, portant le nom des deux binômes. Cette archive contiendra l'ensemble des fichiers Java que vous aurez développés et nécessaires à la compilation et l'exécution de votre projet, ainsi qu'un petit rapport (PDF, MD ou TXT).

Un seul des deux binômes rend le travail, aussi il est nécessaire de bien faire figurer les deux noms dans le travail déposé.

Dans le petit rapport qui accompagne le projet, vous indiquerez
— les fonctionnalités du sujet qui ont été implantées ;
— le nom de chacune des classes rendues ainsi qu'un court descriptif de leur utilité dans le projet ;
— les choix que vous avez fait pour l'initialisation des paramètres du jeu ;
— la description précise (en français) des règles (changement de direction, choix de la prochaine position, ...) que vous avez implantées pour gérer les déplacements, en particulier lors des collisions avec les bords, ou avec les autres personnages ;
— les éventuels ajouts que vous auriez fait par rapport au sujet initial ;
— les éventuelles petites libertés que vous auriez prises dans l'interprétation des règles du jeu (à condition de bien les justifier) ;
— ...

De manière plus générale, vous pouvez en plus inclure dans le rapport tout ce pensez utile de porter à la connaissance de votre enseignant.

8 ANNEXE. Utilisation d'une **ArrayList**

Pour simplifier, on peut considérer qu'une `ArrayList` est comme un tableau dont la taille s'ajuste automatiquement au contenu. On garde un accès indexé aux éléments mais on n'a pas la syntaxe à base de doubles crochets '`[ ]`'.

Il faut ajouter `import java.util.ArrayList;` en tête des fichiers de classe qui utilisent une ArrayList.

Déclaration et création. On indique entre chevrons le type des éléments à stocker dans l'ArrayList, et l'instanciation se fait par `new`. Par exemple, pour déclarer et instancier une ArrayList nommée `messages` et contenant des chaînes de caractères :

```
ArrayList<String> messages = new ArrayList<String>();
```

Et pour une ArrayList nommée `voisines` contenant des objets de type `Position` :

```
ArrayList<Position> voisines = new ArrayList<Position>();
```

Ajout, retrait, accès aux éléments. Une méthode nommée `add()` permet d'ajouter un élément dans une ArrayList. L'élément se retrouve ajouté à la suite des autres. Exemples :

```
messages.add("Nouveau Message");
```

```
voisines.add(new Position(2, 4));
```

Si on veut insérer à un indice i particulier (les décalages se font automatiquement si besoin) :

```
messages.add(i, "Nouveau Message");
```

```
voisines.add(i, new Position(2, 4));
```

Pour accéder à un élément d'indice i :

```
String msg = messages.get(i);
```

Pour retirer un élément situé à l'indice i (les décalages se font automatiquement si besoin) :

```
messages.remove(i);
```

Enfin, pour connaître le nombre d'éléments présents dans l'ArrayList :

```
int nbMessages = messages.size();
```

Consultez l'API de ArrayList pour plus de détails : `https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html`