**DEVOIR DE CONTROLE CONTINU**

UNIVERSITÉ CAEN NORMANDIE

**A RENDRE POUR LE**

**LUNDI 2 DECEMBRE 2024 A 13H**

## 1. Organisation

Ce devoir de Contrôle Continu est à réaliser en groupes de 4 étudiants. En cas de nécessité, des groupes de 3 étudiants pourront être acceptés, mais en aucun cas des groupes de plus de 4.

Une partie du temps de travail de TP pourra y être consacrée, selon les indications de vos enseignants respectifs, mais du travail hors des heures de TP sera indispensable pour mener à bien ce devoir.

Un dépôt (svn ou git) devra être créé sur la forge pour chaque groupe (obligatoirement comme un sous-projet du projet "TP Génie Logiciel L3"). Les noms court et long du dépôt devront comporter les noms des quatre membres du groupe, selon l'exemple suivant :

- nom court "durand-dupont-smith-doe"
- nom long "Génie Logiciel Durand, Dupont, Smith, Doe".

D'autre part, Christophe Charrier, François Ledoyen, Guillaume Letellier, Amal Mahboubi et Yann Mathet devront être ajoutés comme **managers** du dépôt. Si ces points ne sont pas respectés, le devoir ne sera pas corrigé (note = 0). Attention : ne pas faire hériter les membres du projet lors de la création du dépôt.

Le 2 décembre à 13h, le code de chaque groupe sera extrait de son dépôt pour correction : un répertoire nommé *livraison* devra être présent à la racine et contenir :

- un répertoire *src* contenant le code commenté
- un répertoire *doc* contenant la Javadoc
- un répertoire *dist* contenant l'exécutable (un fichier .jar, et d'éventuelles ressources nécessaires à l'exécution)
- un fichier *build.xml* permettant de re-générer le contenu de dist via ANT.
- Et enfin un *répertoire* rapport contenant un mini rapport au format PDF contenant toute information utile à la compréhension de votre conception logicielle. Par exemple quelques diagrammes de classes précisant votre mise en œuvre de cer-

tains design patterns, le principe de vos algorithmes non triviaux (mais pas directement le code), etc. Ce rapport pourra faire aux environs de 3 à 7 pages bien illustrées.

## 2. Critères d'évaluation

En premier lieu, nous prendrons en compte la qualité de l'architecture logicielle et de votre code. Nous regarderons notamment si votre conception est :

- facile à comprendre (répartition en packages et en classes cohérente, noms de classes et de méthodes éloquents, commentaires dans le code lorsque c'est utile)
- facile à maintenir et à faire évoluer (pas de code spaghetti, pas d'inter-dépendance entre packages, pas de redondance, mise en œuvre de patterns permettant l'intégration de nouveaux éléments ou algorithmes, etc.)
- robuste (tests effectués)

Bien sûr, une bonne ergonomie, un design agréable, des options supplémentaires seront appréciés mais ne constitueront pas l'essentiel de la note. En particulier, une application qui ferait ce qui est demandé dans le sujet mais qui ne répondrait pas aux critères d'évaluation exposés ci-dessus n'obtiendrait assurément pas la moyenne.

## 3. Sujet : Jeu de stratégie au tour par tour

On souhaite réaliser une application de jeu de combat opposant entre 2 et n joueurs sur une grille en 2 dimensions, de taille paramétrable.
Un combattant dispose initialement d'une certaine énergie et d'armes de différents types, avec des munitions en nombre limité.

**Règles du jeu :**

À chaque tour de jeu, les joueurs jouent l'un après l'autre selon un ordre initialement défini, ou selon un ordre tiré aléatoirement à chaque tour.
Une action possible est :
- déplacement d'une case (4 directions possibles seulement)
- dépôt d'une mine ou d'une bombe sur l'une des 8 cases voisines
- utilisation d'un tir horizontal ou d'un tir vertical.
- déclenchement du bouclier, qui protège des tirs et bombes lors du prochain tour
- ne rien faire pour économiser son énergie

La grille peut contenir, lors de sa création, des murs (cases non utilisables par les combattants et infranchissables par les tirs), et des pastilles d'énergie que les combattants peuvent récupérer en se plaçant dessus.
Une mine explose lorsqu'un combattant se place sur la case qu'elle occupe.
Une bombe explose au bout d'un certain délai t (compte à rebours en nombre de tours de jeu), et impacte les combattants se trouvant sur l'une des 8 cases voisines, ou, comme une mine, si l'on se place dessus.
Les bombes et les mines peuvent avoir deux types de visibilité : visible de tous les combattants, ou visible seulement du combattant qui l'a déposée.
Un tir horizontal impacte les combattants se trouvant sur la même ligne horizontale, à un nombre de cases inférieur à la distance d (portée limitée). Idem pour un tir vertical (i.e. sur la même colonne).

Une explosion ou un tir impactant un combattant fait baisser son énergie d'une certaine valeur qui fera partie des paramètres du jeu.
Un déplacement et l'utilisation du bouclier ont eux aussi leurs coûts respectifs.
Un combattant perd le jeu lorsque son énergie est négative ou nulle. Il disparaît alors du jeu. Le gagnant est le dernier survivant.

**Conception** :

On utilisera les méthodes de conception et de développement vues en cours, avec pour objectif majeur la production d'un code propre, facile à lire, modulaire, réutilisable et facilitant de futures extensions.
En particulier, conformément à la mise en place d'une architecture MVC, la partie modèle du jeu devra être complètement indépendante de l'interface graphique (vue-contrôleur), et devrait pouvoir être utilisée selon d'autres modalités (en ligne de commande, via une application en réseau, etc.)
L'application doit être facilement paramétrable, au minimum via des constantes clairement identifiées dans une classe précise du code, ou mieux, via un fichier de paramétrage.
L'une des difficultés particulières de ce jeu est que certaines armes (bombes, mines) peuvent n'être visibles que pour les joueurs qui les ont déposées. En termes « objet », il faut donc faire en sorte que les joueurs aient un accès partiel au modèle complet du jeu. On évitera de dupliquer les modèle complet en modèles partiels (très inefficace). Suggestion : le pattern Proxy (à voir en cours) permettant de ne faire voir qu'une partie des données réelles. Il y aurait donc un proxy par joueur, configuré pour lui.
De la même façon, concernant les vues, on devra afficher autant de vues que de joueurs, chaque vue ne montrant que ce que le joueur en question peut voir (bien sûr, dans une application réelle, le jeu serait multipostes, et chaque joueur ne verrait que sa grille).
On pourra utiliser une Factory (factory method) pour créer différents types de combattants, certains avec beaucoup d'énergie mais peu d'armes, etc.
On pourra utiliser Strategy pour :
- définir la façon dont on remplit initialement la grille (murs, pastilles d'énergie, position initiale des joueurs)
- définir la stratégie d'un joueur robot (on pourra faire combattre des joueurs ayant des stratégies différentes sur un grand nombre de parties pour voir quelles sont les meilleures). On pourra créer une première stratégie aléatoire, puis créer au moins une stratégie un peu plus fine.
On pourra se limiter à une version de base ne faisant jouer que des robots (l'interface graphique ne propose plus alors comme seul contrôleur qu'un bouton « Suivant » qui fait passer au tour suivant).

**Suggestions concernant l'interface graphique :**

Une Frame principale pourrait montrer le jeu dans son intégralité, c'est-à-dire incluant les mines et bombes éventuellement invisibles pour la plupart des combattants. Cette frame pourrait tenir à jour une liste des combattants avec leur énergie et leurs armes respectives (ici encore, un pattern vu en cours pourra être utilisé conjointement à un composant JTable).
En plus de cette frame, il pourrait être intéressant de générer autant de frames qu'il y a de joueurs, chacune montrant seulement ce que peut voir un joueur particulier.