DEVOIR DE CONTROLE CONTINU

UNIVERSITÉ CAEN NORMANDIE

A RENDRE POUR LE LUNDI 30 NOVEMBRE 2020 (AVANT 20H)

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 devra être créé sous subversion 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, Joannes Falade, Yohann Jacquier et Yann Mathet devront être ajoutés comme **managers** du dépôt. Si ces points ne sont pas respectés, le devoir ne sera pas corrigé (note de 0).

Le 30 novembre, 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 oeuvre de certains design patterns, le principe de vos algorithmes non triviaux (mais pas directement le code), etc. Ce rapport pourra faire aux environs de 5 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 classes, pas de redondance, mise en oeuvre 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 d'assemblage de formes

Le but de ce devoir est de réaliser une application de jeu, dotée d'une interface graphique, qui consiste à assembler des formes de sorte qu'elles occupent le moins de place possible.

L'idée s'inspire du fameux Tetris, mais les modalités sont différentes :

- toutes les pièces sont exposées sur l'aire de jeu dès le début de la partie,
- à tout moment, le joueur peut choisir une pièce en cliquant dessus, et a alors la possibilité de la faire tourner ou de la déplacer,
- son but est de minimiser l'espace occupé par l'ensemble des pièces. Plus précisément, la fonction d'évaluation sera l'aire du plus petit rectangle (parallèle aux axes) recouvrant l'ensemble des pièces,
- lorsque le joueur considère avoir terminé (ou lorsque le nombre maximum d'actions autorisées est atteint), il clique sur un bouton est son score est alors calculé,
- à chaque partie, il est possible soit de demander à obtenir une nouvelle configuration de départ, soit charger une configuration déjà créée et sauvegardée. Dans ce dernier cas, le meilleur score ainsi que le nom du joueur correspondant seront sauvegardés.
- Une option permettra de faire jouer l'ordinateur. Réaliser un bon joueur robot nécessiterait des connaissances solides en IA, mais vous essaierez de faire mieux que le hasard.

Différents types de formes pourront être conçus, par exemple des "L", des "T", des peignes ou des rectangles. Chacune de ces formes pourra être générée avec des valeurs tirées aléatoirement (par exemple certains "L" pourront être assez allongés, d'autres plus ramassés, etc.).

La mise en place d'une configuration initiale de jeu pourra être paramétrée (avec des paramètres tels que le jeu ne soit pas trop dense et reste jouable), et bien sûr les pièces ne doivent pas se superposer.

À chaque fois que cela vous semblera opportun, vous essaierez de mettre en place des design-patterns. Le minimum requis est que l'application soit intégralement MVC

c'est-à-dire que l'interface graphique ne soit qu'une couche possible de jeu, mais que l'on puisse potentiellement jouer par exemple en ligne de commande sans interface graphique sans avoir à modifier le modèle. Voici quelques suggestions de patterns (non obligatoires et non limitatives) pour ce jeu :

- Strategy, pour la mise en place des stratégies possibles du joueur robot. Vous pourrez ainsi proposer une stratégie de base qui fait des actions aléatoires, puis un joueur un peu plus évolué.
- Strategy encore, pour la création de la configuration initiale d'une partie.
- Chain of Responsibility : pour vérifier qu'une action du joueur est possible, avec un maillon vérifiant qu'il n'y a pas chevauchement avec une autre pièces, et un maillon vérifiant que l'on ne sort pas du cadre du jeu.
- Factory, pour la création des pièces lors d'une nouvelle partie. Ainsi, une factory donnée pourrait proposer de nouveaux types de formes, chaque factory pourrait proposer une répartition de formes spécifiques, etc.