Projet de POOIG

I) Introduction

Le projet de POOIG est un exercice dans lequel vous devez faire la démonstration que vous avez passé le cap de la simple programmation telle qu'elle était faite en L1. Les éléments orientés objet que nous avons vus ce semestre (l'héritage, les interfaces, la généricité, les modèles MVC, les exceptions, les énumérations ...) sont autant d'éléments conceptuels qui doivent venir enrichir votre style et vous permettre d'écrire des programmes plus clairs, avec une décomposition qui invite à une réutilisation simple lorsqu'il vous faut résoudre des problèmes conceptuellement proches.

Pour ce projet, nous allons vous décrire deux jeux *"conceptuellement proches"*. Vous devrez exploiter leurs points communs pour réaliser des prototypes pour les deux jeux, en réutilisant le maximum d'éléments. [1]

Le premier jeu est relativement simple, il s'agit d'une extension d'un jeu de dominos. Son étude vous permettra de mettre les choses en place. Le second jeu servira à démontrer la qualité de vos efforts de conception puisque pour l'essentiel ce qui a été fait pour le premier jeu sera réutilisable. Avant de commencer, lisez les deux règles de jeu, cela vous permettra d'avoir en tête ce qu'il faudra factoriser tôt ou tard. [2]

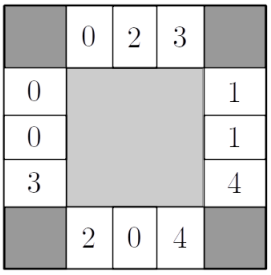
II) Généralités

- Le projet est à faire en binôme. Les monômes ne seront pas acceptés sauf pour des questions de parité. Vous pouvez éventuellement vous associer à quelqu'un qui n'est pas de votre groupe de TD (utilisez le forum dédié sur Moodle pour entrer en contact). Il n'y aura absolument pas de trinômes, et il est entendu que si des travaux se ressemblent trop nous prendrons les sanctions qui s'imposent.
- Il vous faut déclarer la constitution de votre binôme avant les vacances de la Toussaint (sur Moodle - section projet, jusqu'au 30/10 minuit).
- La note de soutenance pourra être individualisée, chacun doit donc maîtriser l'ensemble du travail présenté, y compris la partie développée par son camarade.
- La soutenance aura lieu durant la période des examens, et votre travail sera à rendre quelques jours avant. Nous vous donnerons les dates exactes lorsque les réservations seront confirmées.

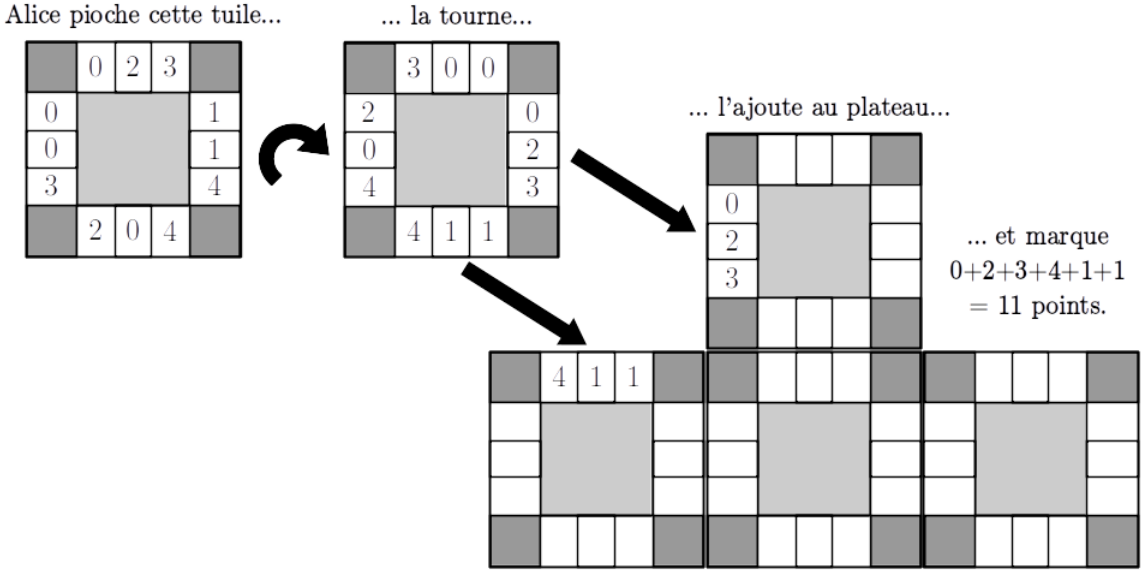
III) Dominos carrés

Plusieurs joueurs sont assis autour d'une table, des tuiles sont à leur disposition dans un sac opaque mis en commun. Les tuiles sont des carrés où chaque coté porte trois chiffres.

1. ... réutiliser n'est pas copier-coller ! On parle d'utiliser la *même* classe dans les 2 contextes.
2. Un code ne s'écrit pas d'une seule traite, vous serez amenés à le réorganiser.



Au départ, une tuile (prise au hasard) est posée face visible. Chaque joueur à son tour va en piocher une dans le sac, et la déposer sur la table, à coté des autres (avec l'orientation de son choix), pourvu qu'il trouve une correspondance bord à bord avec celles qui y sont déjà. S'il ne le peut pas il la défausse (sans la remettre dans le sac) et passe son tour. Lorsqu'un joueur pose une nouvelle tuile, il marque alors un certain nombre de points : le total des chiffres en contact avec ceux des tuiles voisines.



Le jeu se poursuit tant que le sac n'est pas vide. Les joueurs peuvent aussi abandonner.

IV) Présentation superficielle de Carcassonne

Les règles complètes sont disponibles sur :

`https://www.jeuxavolonte.asso.fr/regles/carcassonne.pdf`.

De la même façon que pour les dominos carrés, le jeu se joue à plusieurs joueurs placés en cercle autour d'une table. Chacun à leur tour, les joueurs piochent une tuile dans un sac puis la posent sur la table. La correspondance ne se fait plus avec des chiffres, mais avec des éléments graphiques : villes, champs, routes, etc. L'image ci-dessous vous donne une idée du paysage qui peut se former.

Chaque joueur, après avoir placé sa tuile peut ensuite déposer un pion (*partisan*) de sa couleur sur un élément du paysage de cette tuile. En nombre limité, les partisans permettront à leur propriétaire de marquer des points de victoire plus tard. Par exemple, un partisan posé sur :

— une ville, rapporte des points fonction de la surface de la ville ;
— une route, rapporte des points en fonction de la longueur de la route ;
— une abbaye, rapporte des points en fonction du nombre de tuiles adjacentes, etc.

Ce jeu est bien plus complet que les dominos, mais *conceptuellement proche*.

V) Cahier des charges

On rappelle que la bonne utilisation des notions de Programmation Orientée Objet constituera une part importante de l'évaluation. Même s'il était fonctionnel, un code qui ne comporterait qu'une seule classe serait un cas extrême fortement pénalisé. Essayez d'illustrer au maximum les aspects vus en cours.

Cahier des charges minimal

Au minimum, voilà ce que votre code devra réaliser :

1. Un environnement de jeu, réalisant l'accueil de l'utilisateur, et permettant de procéder au *paramétrage du jeu* :
   — choisir le jeu (dominos ou Carcassonne) ;
   — choisir le nombre de joueurs ;
   — sélectionner "humain" ou "IA" pour chaque joueur [3].
2. L'implémentation de toutes les règles du jeu du domino.
3. L'implémentation partielle des règles du jeu de Carcassonne. Plus précisément, votre programme devra gérer la partie de la pose de la première tuile jusqu'à ce que le sac soit vide, en s'assurant que les règles sont respectées, avec 2 exceptions majeures :

3. Votre programme demandera aux joueurs "humains" quelles sont leurs décisions, et générera automatiquement les décisions des joueurs "IA". Ceux ci **n'ont pas à être très intelligents**, il nous suffira que l'IA soit toujours capable de faire un choix.

— il n'est pas demandé de vérifier les contraintes de placement des pions : un joueur peut toujours placer un pion sur un élément de la tuile qu'il vient de jouer (pourvu qu'il lui en reste).
— il n'est pas demandé de compter les points.

4. Un mode de jeu "texte" pour les dominos (pouvant se jouer entièrement dans le terminal). Il n'a pas vocation a être très ergonomique, nous vous le demandons pour être certains que votre travail ne soit pas bloqué par le traitement de problèmes liés à l'interface graphiques.
5. Un mode de jeu "graphique" pour les dominos et pour Carcassonne utilisant `awt` et `swing` (et uniquement ces librairies [4]).

Fonctionnalités avancées

Ces fonctionnalités avancées viennent compléter le cahier des charges minimal. Elles sont **indépendantes** les unes des autres. Elles ne sont **pas obligatoires** mais seront valorisées.

Même si vous ne les implémentez pas, il est utile de réfléchir aux questions qu'elles posent, et d'anticiper les modifications qu'elles provoqueraient dans votre code. En général, suivre des ambitions généralistes aide à produire un code *"bien conçu"*, c'est pourquoi nous ouvrons ici votre horizon.

1. **Carcassonne 100%.** Implémenter les règles manquantes de Carcassonne : contraintes de placements des pions et calcul du score. (Pour cela, vous aurez besoin d'explorer récursivement les tuiles.)
2. **Sauvegardes.** Permettre de sauvegarder des parties, et de les charger lors du paramétrage du jeu. (En java les sauvegardes se font assez simplement en faisant en sorte que vos objets implémentent l'interface `Serialisable`. Référez vous à la documentation de cette interface.)
3. **HAL 9000.** Concevoir une IA meilleure que le hasard, pour un des deux jeux ou les deux. Expliquez votre démarche dans le rapport.
4. **Hexa-Carcassonne.** Ces jeux sont naturellement les mêmes si l'on utilise des tuiles hexagonales plutôt que carrées. Pouvez vous commencer à structurer de tels jeux ? (L'implémentation n'est pas demandée. Vous pouvez vous contenter de décrire votre approche en détail dans le rapport.)

VI) Le cas des tuiles

Une attention particulière est à porter à la modélisation des tuiles.

— Il serait maladroit d'utiliser des chiffres incompréhensibles pour décrire les éléments du paysage de Carcassonne, préférez des objets ou au moins des énumérations.
— Il n'est pas nécessaire de considérer exactement les tuiles du jeu Carcassonne original. Vous pouvez générer vous-même des tuiles (pourvu qu'elles soient **variées**, et que chaque élément du paysage apparaisse sur au moins quelques tuiles). En particulier, la structure des routes peut être simplifiée : on accepte que les routes soient toujours connectées par un centre conventionnel.
— Envisagez différentes possibilités de modélisation. Par exemple, vous pouvez diviser les tuiles en plusieurs zones, et indiquer pour chaque zone de quel élément de paysage il s'agit – ou bien conserver une liste avec les différentes composantes connexes. Vous pouvez

4. Pas de `gradle` ou `maven`, pas de `javafx`.

considérer que les routes sont de même nature que les villes et les champs – ou bien les traiter à part. Et ainsi de suite...

Dans tous les cas, prenez le temps de bien réfléchir à ce que vous voulez faire avec les tuiles.

VII) Conseils préalables à l'implémentation

a) Sauvegarde

Sauvegardez régulièrement votre travail. Chaque fois que vous envisagez une modification importante, conservez bien la version antérieure. Ces précautions permettent d'éviter des catastrophes ! Vous pouvez utiliser chez vous git/eclipse **mais conformez vous strictement aux contraintes de rendus (pas de gradle ou maven, pas de javafx)**.

b) Décomposition du code

Pour pouvoir maîtriser la complexité de votre travail, il vous faut absolument le découper en objets et méthodes qui joueront des rôles bien délimités. Vous avez une grande liberté dans la façon dont vous ferez votre développement, mais veillez à distinguer les choses conceptuellement. Vous remarquerez qu'alors chaque petite variation ou évolution pourra se faire facilement et localement. De plus, cela est essentiel pour pouvoir vous répartir le travail.

Il est toujours préférable d'avoir plusieurs petites méthodes à écrire plutôt qu'une seule grande méthode qui ferait un travail compliqué à déchiffrer. Si vous avez quelque part dans votre code un bloc qui fait plus d'une vingtaine de lignes, alors il est quasi-certain que vous devriez vous relire pour introduire une phase intermédiaire.

c) Développement progressif

"Premature optimization is the root of all evil". Commencez par élaborer une version jouable **minimale** (mais fonctionnelle) du jeu avant de vous lancer dans les fonctionnalités plus compliquées comme les aspects graphiques. Vous pouvez décrire votre organisation dans le rapport.

Veillez à tester rigoureusement votre code après chaque ajout conséquent, et *avant* de passer à l'étape suivante. Écrivez pour cela toutes les fonctions d'affichage nécessaires. Cela peut sembler fastidieux au premier abord, mais vous y gagnerez sur le long-terme en passant moins de temps à corriger les erreurs. Vous pouvez documenter vos tests dans le rapport.

d) Vue et Modèle

Il est important de bien séparer la vue du modèle. On rappelle que la *vue* désigne l'ensemble des aspects liés au rendu graphique, et le *modèle* regroupe l'ensemble des concepts qui sont sous-jacents, vraiment propres au jeu, et qui sont largement indépendants de la vue.

Ainsi, si vous souhaitez faire évoluer votre programme pour en changer la présentation, par exemple pour l'adapter à l'écran d'un téléphone ou d'une tablette, alors l'essentiel du jeu sera tout de même préservé. Les changements à faire relèvent tous de ce qu'on appelle *la vue*. De la même façon, si on souhaite changer un peu les règles, ajouter des variantes, celles ci concernent essentiellement *le modèle*.

Dans votre cas, de toutes façons, vous allez dans un premier temps présenter les choses en mode texte puisque les aspects graphiques seront abordés progressivement en cours de semestre. Puis, lorsque vous serez plus avancé dans votre réflexion, que vous aurez davantage de connaissances sur les interfaces graphiques, et que vous aurez mieux cerné ce que vous souhaitez apporter

au projet, alors il sera temps de redéfinir les vues. Vous pouvez **prévoir cela en définissant assez tôt une interface ou une classes abstraites pour les vues attendues**, il suffira ensuite de l'instancier par des sous-classes plus ou moins évoluées (textuelle ou graphique). L'association entre les modèles et les vues se fera en introduisant un champs typé `VueGenerale` dans le modèle de votre jeu, et réciproquement si besoin. Pour rendre compte d'une modification, graphique ou textuelle, le jeu s'adressera à sa vue. La vue se chargera aussi de transmettre les actions choisies par le joueur au modèle.

VIII) Aspects pratiques de l'évaluation

Rendu

Les travaux sont à rendre sur `Moodle` sous la forme d'une archive nommée *`nom1-nom2`* selon la constitution de votre binôme, et qui s'extraira dans un répertoire *`nom1-nom2`*. Elle devra contenir :

- les sources et ressources (images ...) de votre programme ; ne polluez pas votre dépôts avec des fichiers `.class` inutiles ...
- un fichier nommé `README` qui indique comment on se sert de votre programme (compilation, exécution et utilisation) ; faites en sorte que son utilisation soit la plus simple possible. Ne pensez pas que nous utilisons le même environnement de développement que celui que vous avez choisi. **Important :** vous n'utiliserez pas javafx mais uniquement swing et les choses vues dans ce cours ci. Pour simplifier la portabilité vous n'utiliserez pas non plus ni maven ni gradle, même si la majorité d'entre vous les ont utilisés en conduite de projet. Ici, tout doit être compilable sous java 11, et "à la main".
  Le correcteur ne doit avoir que deux choses à faire pour tester votre travail [5] :
  1. décompresser votre dépôt dans une console unix ;
  2. lire votre fichier README et y trouver la ligne de commande qui lance le programme.
- un rapport au format *PDF* d'au moins cinq pages **rédigées** expliquant les parties du cahier des charges qui ont été traitées, les problèmes connus, et les pistes d'extensions que vous n'auriez pas encore implémentées. Il devra contenir impérativement une représentation graphique du modèle des classes (le plus simple est qu'elle soit manuscrite, puis scannée). Le rapport n'est en aucun cas une impression de votre code !
- toute chose utile pour rendre la soutenance fluide.

Soutenance

La soutenance devra pouvoir se dérouler sur une machine du script à partir des sources que vous avez déposées. Si tout se passe bien elle se déroulera devant un ou deux enseignants dans un mélange de questions et de tests. Il pourra vous être demandé de modifier votre code pour répondre à une question spécifique.

Idéalement, vous pouvez proposer des "tutoriaux" (accessibles lors du paramétrage du jeu), ou de parties sauvegardées (si vous avez implémenté cette fonctionnalité) pour mettre le joueur directement dans des situations précises, afin de gagner du temps lors de la soutenance. Cela n'est pas obligatoire.

5. Testez vous même votre rendu sur une autre machine ou pour le moins dans un dossier séparé de celui où vous avez travaillé. Si nous n'arrivons pas à exécuter votre code vous serez évidement fortement pénalisé.