Projet de POOIG

Ce projet est basé sur le jeu `Les Colons de Catane` dont vous pouvez trouver les règles complètes sur : `https://www.regledujeu.fr/catane/`. Il est disponible en application android sous le nom `Catan Universe`, ou sur un navigateur de PC `https://catanuniverse.com/fr/game/`

Vous avez probablement déjà passé du temps en soirée ou en ligne sur des jeux de ce type. L'intérêt ici est de vous rendre compte du travail à fournir pour en réaliser les bases. Bien entendu il ne s'agit pas de produire un résultat final de qualité professionnelle comme ceux précédemment cités, mais d'aborder le problème de la conception en compartimentant bien les différents aspects, quitte à les améliorer ensuite.

I) 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 1/11 minuit).
- La note de soutenance pourra être individualisée, chacun doit donc maîtriser l'ensemble du travail présenté, y compris la partie développée par votre 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.

II) Présentation superficielle du jeu

Sur l'image ci-dessous vous pouvez observer un plateau du jeu. On y distingue différentes tuiles hexagonales, qui représentent des terrains de 5 natures possibles (ainsi qu'un désert) :

- des champs de blé,
- des forêts,
- des paturages,
- des rochers,
- de l'argile.

Les joueurs sont repérés par leurs couleurs, ici il n'y en a que 3 : le bleu, le blanc et le rouge, mais il peut y en avoir davantage selon les scénarios. Ils prennent possession d'endroits stratégiques : des villes peuvent être construites sur les sommets des tuiles, et des routes sur les arêtes.

Chaque joueur, à son tour, lancera deux dés. Leur total désignera certaines tuiles : c'est pour cela que des numéros entre 2 et 12 sont disposés sur chacune d'entre elles. Lorsqu'un total sort, chaque joueur possédant une ville sur un sommet d'une tuile correspondante reçoit la ressource associée : du blé, du bois, un mouton, une pierre ou de l'argile. Elles lui permettront de se développer, car les constructions se payent avec des combinaisons de ces ressources.

Ces premiers éléments en tête vous pouvez à présent aborder les règles plus générales sur : `https://www.regledujeu.fr/catane/`

III) Cahier des charges

La bonne utilisation des connaissances vues en cours 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 qui sera inévitablement fortement pénalisé.

Cahier des charges minimal

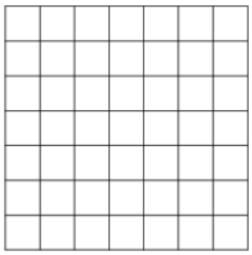
Au minimum, voilà ce que votre code devra supporter :

1. Un environnement de jeu, réalisant l'accueil de l'utilisateur, et permettant de procéder au *paramétrage du jeu*. Il doit permettre :
   — de choisir le nombre de joueurs (3 ou 4) ;
   — de sélectionner "humain" ou "IA" pour chaque joueur.
2. La création d'un plateau constitué de tuiles **carrées** (voir 1a). Pour simplifier l'affichage, nous n'utiliserons **pas** les hexagones.
3. L'implémentation de toutes les règles du jeu – à l'exception du *négoce* (c'est-à-dire la possibilité pour les joueurs d'échanger des ressources). Voir la section IV)e) pour un rappel.
4. Votre programme demandera aux joueurs "humains" quelles sont leurs décisions (via la *vue*), et générera les décisions des joueurs "IA". Ces décisions n'ont pas à être intelligentes. L'IA devra simplement être capable de générer des coups légaux dans chaque situation (même si ceux-ci sont purement aléatoires) sans se "bloquer".
5. Pour illustrer la séparation de la *vue* et du *modèle* le jeu doit pouvoir être joué soit en mode graphique, soit en mode texte (les deux options doivent être implémentées). La vue textuelle n'a pas vocation a être compliquée, elle pourra être purement descriptive. Voir la section IV)d) pour un rappel.

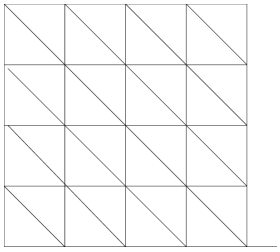
Fonctionnalités avancées

Les fonctionnalités suivantes sont indépendantes les unes des autres, et ne sont pas obligatoires. Vous pouvez en implémenter certaines si vous avez le temps, ce dont nous tiendrons compte dans la notation. Même si vous ne les implémentez pas, il est intéressant de réfléchir à où et comment modifier votre code pour les intégrer.

1. Ajouter la possibilité lors du paramétrage du jeu de créer un plateau avec des tuiles triangulaires (voir 1b).
2. 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.



(a) Plateau carré



(b) Plateau triangulaire

Figure 1 – Nous n'utiliserons pas les hexagones pour ce projet. L'avantage est que sur ces deux visualisations, il est très facile de repérer les sommets et les arêtes, ce qui facilitera d'autant vos calculs et votre disposition graphique sans changer les règles.

3. Implémenter le négoce. Dans le jeu réel, les échanges jouent un rôle important !
4. Concevoir une IA plus avancée (meilleure que le hasard). Expliquez alors votre démarche dans le rapport. Si vous avez plusieurs versions d'IA en tête, rien ne vous empêche de les implémenter toutes, et de les proposer à l'utilisateur lors du paramétrage du jeu.
5. Proposer des variantes des règles. L'utilisateur devra pouvoir les activer / désactiver lors du paramétrage du jeu. Voici quelques idées :
   — Améliorer les tuiles aux probabilités les plus faibles (pour équilibrer le jeu), en laissant les joueurs choisir la ressource qu'ils reçoivent parmi plusieurs propositions par exemple.
   — Donner la possibilité de récupérer une route construite pour la placer ailleurs, pour un coût que vous déciderez.
   — Imaginer de nouvelles cartes de développement. Par exemple, une carte qui permet de relancer les dés s'ils ne conviennent pas, ou de rejouer immédiatement...

IV) 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 !

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. Par exemple, vous pouvez laisser le brigand, les ports et les cartes développement de côté dans un premier temps, et les ajouter ensuite (l'un après l'autre). Il peut être utile de planifer à l'avance l'ordre dans laquelle vous implémenterez les différents éléments. 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. Ecrivez 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.

e) Règles à implémenter

Voilà un résumé pour vous permettre de ne rien oublier. Il faut qu'un joueur soit capable de :
- construire des routes, villes, cités ;
- consulter ses ressources ;
- gérer le personnage du voleur en cas de 7 aux dés ;
- acheter, stocker, utiliser les cartes spéciales de développement ;
- échanger des ressources via les ports.

Le score se calcule simplement à partir du nombre de ville et de cités construites. Vous remarquerez cependant qu'il y a aussi des bonus particuliers :
- le tirage d'une certaine carte de développement ;
- le fait d'être celui qui a joué le plus de chevaliers ;
- le fait d'être le joueur qui a la route la plus longue (ce problème est un peu plus délicat et intéressant algorithmiquement).

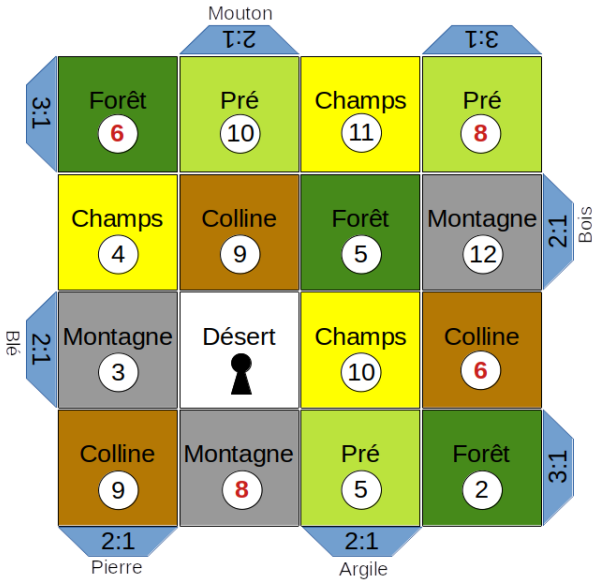


Figure 2 – Vous êtes libre de la façon dont vous initialiserez le plateau (taille et arrangement). Vous pouvez même laisser le choix à l'utilisateur lors du paramétrage du jeu. Ceci est une proposition de configuration de départ dans le cas d'un plateau carré 4x4 que vous pouvez utiliser.

V) 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. Le correcteur ne doit avoir que deux choses à faire pour tester votre travail [1] :
  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.

1. 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.

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.