# Projet de C++

## Introduction

Ce projet est un exercice dans lequel vous devez faire la démonstration que vous savez utiliser en c++ les éléments que nous avons vus ce semestre (la construction/destruction/copie, l'héritage, la généricité, les exceptions, les énumérations, la généricité, ...) Ils doivent venir enrichir votre style et vous permettre d'écrire des programmes clairs, avec une analyse qui invite à une réutilisation simple lorsqu'il vous faut résoudre des problèmes conceptuellement proches.

Pour ce projet, nous allons vous décrire des jeux *"conceptuellement proches"*. Vous devrez exploiter leurs points communs pour réaliser des prototypes qui réutilisent 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, très proche, ajoute une condition de gain non triviale, il servira à démontrer la qualité de vos efforts de conception puisque pour l'essentiel ce qui a été fait pour le premier jeu devrait être réutilisé. Le troisième jeu est un cas plus complet. Les variations que vous aurez identifiées d'une jeu à l'autre vous guideront dans vos choix d'abstraction. Avant de commencer, lisez les règles, cela vous permettra d'avoir en tête ce qu'il faudra factoriser tôt ou tard. [2]

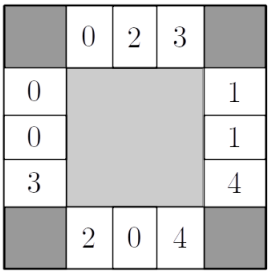
## Généralités

- Le projet est à faire en binôme. Les monômes ne seront pas acceptés sauf pour des questions de parité dans les groupes de TD ou des cas très exceptionnels. Un forum Moodle sera ouvert pour mettre en contact ceux qui cherchent un partenaire. 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, sur moodle, la constitution de votre binôme après les vacances de la Toussaint. Ceci nous permettra de nous assurer que le travail a bien commencé.
- 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, début janvier, et votre travail sera à rendre quelques jours avant. Nous vous donnerons les dates exactes lorsque les réservations seront confirmées.

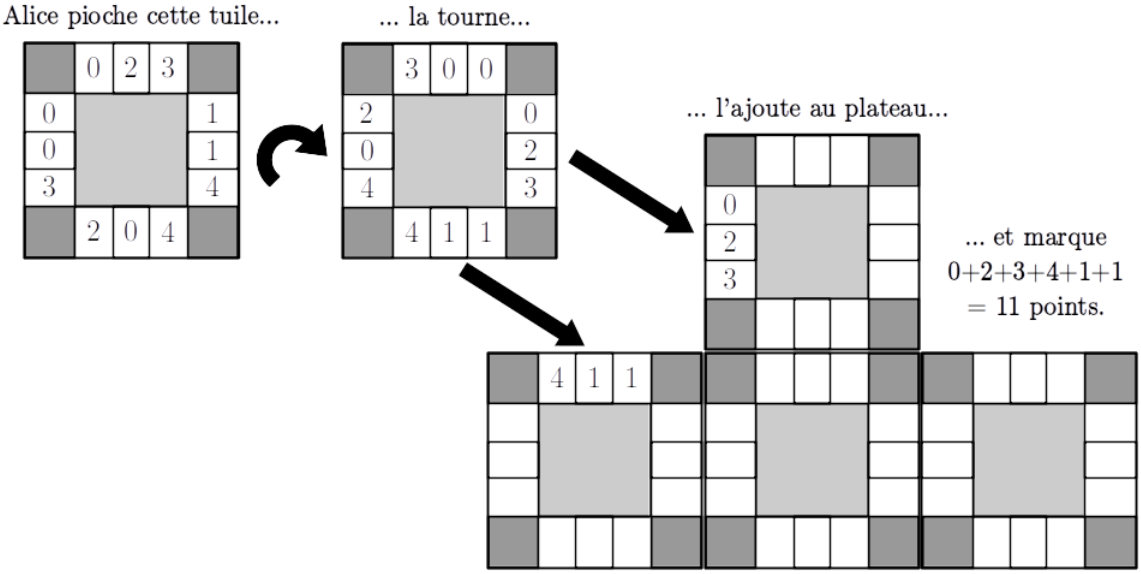
## Dominos carrés

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 veux utiliser les *même* classes de base dans le contexte de jeux différents
2. Rappel : 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. Puis c'est le tour de son voisin.




Le jeu se poursuit tant que le sac n'est pas vide.

## Trax

Il s'agit d'un jeu à deux joueurs, l'un défend la couleur blanche, et l'autre la noire. Les tuiles sont toutes identiques, et elles sont imprimées en recto/verso : [3] :

L'espace de jeu est limité à une surface de 8x8 (sans qu'on sache d'avance où se situeront ces bords). C'est le joueur blanc qui commence. Le tour d'un joueur consiste à prendre une tuile, la

3. `http://jeuxstrategieter.free.fr/Trax_complet.php`

placer bord à bord avec une autre en faisant se correspondre les couleurs (il est libre de choisir la face qu'il veut voir apparaître). Des situations intéressantes existent où le placement d'une tuile détermine sans ambiguïté ce que doit être une voisine. Dans ce cas on place encore toutes les tuiles contraintes avant de laisser la main au joueur suivant.

Les chemins que l'on construit (la croix est considérée traversante pour blanc) permettent de déclarer un vainqueur :

— lorsqu'une boucle est formée
— lorsqu'un chemin relie 2 bords opposés

## Carcassonne

Nous ne faisons qu'une présentation partielle ici, les règles complètes sont disponibles sur : `https://www.jeuxavolonte.asso.fr/regles/carcassonne.pdf`.

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 se fait 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 sa longueur, si elle mène bien quelque part ;
— une abbaye, rapporte des points en fonction du nombre de tuiles adjacentes, etc.

## 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 d'accueil (qui peut être textuel) permettant de procéder à une démonstration :
   — choix du jeu
   — paramétrage rapide
   — accès à des situations particulières
2. L'implémentation des jeux de dominos, Trax, et Carcassonne
3. Un affichage graphique du plateau en FSML. Vous pouvez utiliser la console pour les entrées clavier.

## Conseils habituels

### Sauvegarde

Sauvegardez régulièrement votre travail. Chaque fois que vous envisagez une modification importante, conservez bien la version antérieure afin d'éviter des catastrophes. (Utilisez git par exemple)

### Décomposition du code

Pour pouvoir maîtriser la complexité de votre travail, et être efficace en binôme, il vous faut absolument le décomposer en objets et méthodes qui joueront des rôles que vous aurez bien délimités.

On rappelle qu'il est toujours préférable d'avoir plusieurs petites méthodes plutôt qu'une seule 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.

### Développement progressif

Faire trop de généralisation dès le départ peut vous conduire en pratique à des impasses. Visez un premier objectif raisonnable [4]. Vous retravaillerez ensuite votre code.

### Vue et Modèle

Ne mélangez pas trop les parties du code qui correspondent à ce qui est propre au jeu, et celles qui correspondent à l'affichage. Séparer autant que possible le modèle et sa vue.

## Aspects pratiques de l'évaluation

### Rendu

Les travaux seront à rendre sur `Moodle` sous la forme d'une archive **binomeNum.tar** (Utilisez impérativement ce format) [5] . Elle s'extraira dans un répertoire *nom1-nom2*, et contiendra :

— les sources et ressources (images ...) de votre programme ;

4. faisable en une quinzaine de jours
5. `tar cvf binomeNum.tar fichiers...` crée l'archive mon_fichier.tar

— un Makefile
— un `README` qui indique succinctement ce qu'est ce projet, et comment effectuer la prise en main (compilation, exécution et utilisation). Le correcteur ne doit avoir que deux choses à faire[6] :
  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* **rédigé** explicitant :
  — ce qui a été traité,
  — les aspects les plus significatifs,
  — les quelques problèmes connus,
  — l'état des extensions que vous n'auriez pas eu le temps d'implémenter.
  — un ou plusieurs diagrammes UML avec un niveau de détail suffisant pour comprendre votre architecture.
— toute chose utile pour rendre la soutenance fluide.

## Soutenance

Elle se déroulera devant un ou deux enseignants dans un mélange de questions et de tests. Vous commencerez par une démonstration afin que nous ayons rapidement une vue d'ensemble ; nous étudierons votre architecture en nous basant sur votre diagramme ; vous nous guiderez dans votre code lorsque nous chercherons à vérifier la cohérence entre vos explications et votre réalisation. Il pourra vous être demandé d'écrire quelques lignes de code. Votre style sera scruté.

Vous pouvez préparer des supports pour répondre rapidement à des questions que vous anticiperiez. Dans tous les cas il faudra privilégier les échanges avec le jury.

6. Testez vous même votre rendu sur une autre machine