TD/TP 7
Facultatif (complément au TD/TP 6)

Maillage épisode 2 : L'attaque des cocottes mutantes

C'est très sérieux puisqu'il y sera question d'opérateurs de mutations morphologiques et colorimétriques sur des maillages clonés. Nous procéderons par irradiation stochastique directe des phénotypes des objets.

Objectifs, méthodes

L'objectif principal est d'abord d'avoir terminé le TD/TP 6. Une fois le TD/TP 6 terminé vous pouvez approfondir d'une part le développement du fil conducteur (maillage 2D triangulé) et d'autre part mettre en œuvre les conteneurs set et map dans un contexte où ils brillent par leur efficacité qui est celui du clonage total ou partiel de composites complexes. Je veux parler bien sûr du clonage de la classe Maillage.

Lors du cours 6 (un cours difficile conceptuellement et techniquement) j'ai indiqué que les objets de type-entités sont non copiables ou difficilement copiables. Le problème est d'une part conceptuel : que signifie copier un objet qui a vocation à être l'unique représentant d'une entité du modèle ? Les entités ne sont souvent pas « clonables », on ne clone pas une Personne, un Compte en Banque, un Véhicule d'entreprise... Le problème est aussi technique : comment cloner un objet qui se trouve relié à d'autres objets dans un **réseau** : est-ce qu'on se contente de copier les attributs de l'objet et de copier les liens mais pas les objets liés (copie superficielle) ? Souvent ça n'a pas de sens de copier les liens. Mettre les liens à nullptr ? Alors le clone devient inutilisables si il nécessite le contexte. Ou est-ce qu'on commence à copier les objets reliés (copie profonde) ? Mais alors les objets liés eux même vont vouloir se copier en copiant leurs relation etc... dans une réaction en chaîne l'ensemble du réseau se verra répliqué (comme un ADN se réplique : entièrement). Ceci se formalise par la notion de composante fortement connexe d'un graphe, notion qui sera abordée 2ème semestre en théorie des graphes avec toute la rigueur mathématique nécessaire.

En résumé quand un objet est lié au milieu d'un réseau, soit on ne copie pas cet objet (l'objet est déclaré non copiable) soit on copie le réseau en entier (tout le système). Dans ce dernier cas c'est le plus souvent le rôle d'une classe englobante de coordonner la copie. C'est précisément ce que je vous propose de faire sur ce TD/TP 7 avec la classe Maillage. Les objets Sommet et Triangle apporteront leur aide en proposant des méthodes de construction par copie spécifiques, mais cette copie ne fera sens que dans le contexte plus large du clonage de la classe composite complexe Maillage dans son ensemble : les objets Sommet et Triangle ne sont pas copiables « séparément ».

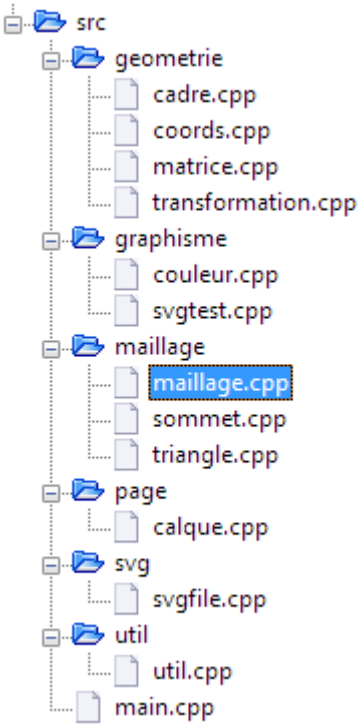
Ce TD/TP facultatif est d'un haut niveau. Ne vous découragez pas si vous n'y arrivez pas ou pas du 1er coup. Je donnerai les consignes sous forme de résultats à obtenir (captures d'écran) et quelques indications sur la façon de procéder. Un nouveau projet de départ est fourni... Ce nouveau projet de départ fournit de **nombreuses** fonctionnalités complémentaires prêtes à être utilisées, en particulier une classe Transformation incluant toutes les transformations affines du plan.

1. Télécharger le projet de départ, tester, comprendre la structure

https://fercoq.bitbucket.io/cpp/tdtp/tdtp7/maillage_v3_exo.zip

2. Intégrer le code que vous aviez fait dans la nouvelle architecture

Vous verrez que ce nouveau projet organise les fichiers d'une façon différente. Dès qu'un projet devient un peu grand on a facilement 50 ou 100 fichiers. On ne peut pas les laisser « en vrac » dans le même répertoire. Comme souvent en informatique on préfère ranger sous forme d'arborescence : on va mettre tous nos fichiers sources dans un sous-répertoire src/ puis dispatcher les sources dans des **répertoires par thème**. Il ne s'agit que d'une **proposition**. Voir capture ci-contre =>



Les fichiers .h ne sont pas montrés ici mais ils partagent les même répertoires avec les .cpp (coords.h et coords.cpp sont tous les 2 dans src/geometrie, voir avec l'explorateur)[1].

Vous ne trouverez pas les fichiers des répertoires src/maillage et src/page : c'est normal, c'est à vous de compléter ces codes ! En principe vous avez terminé le TD/TP précédent donc vous avez vos classes Maillage Sommet Triangle et leurs .h et .cpp correspondants. Copier ces fichier dans le répertoire src/maillage puis les ajouter au projet (dans Code::Blocks clic droit sur le projet puis Add files...) Remplacer le main fourni par celui que vous aviez. Le répertoire thématique page contiendra plus tard (peut-être) une classe Page. Dans ce TP on y mettra juste une classe Calque : un calque aura vocation à regrouper plusieurs maillage (et une page plusieurs calques).

Noter que les chemins d'accès aux includes doivent être modifiés. On utilise .. pour remonter d'un niveau relatif et /rep pour redescendre. Exemple pour include coords.h dans sommet.h on fera #include "../geometrie/coords.h". Modifier en conséquence tous vos include jusqu'à obtenir un projet qui compile et qui marche de la même façon que précédemment (en principe je n'ai pas introduit d'élément incompatible dans la nouvelle version).

3. Modifier votre menu pour reprendre la main après startAutoCin

Pour rappel, appeler par exemple util::startAutoCin("test1.txt", 50); avant votre menu permet d'automatiser le fait d'entrer des infos en simulant des frappes au clavier : le fichier test1.txt contient les « entrée » qu'un utilisateur pourrait donner. Le paramètre 50 sert à régler la vitesse de frappe simulée... L'intérêt est bien sûr d'accélérer les tests qui peuvent être longs et fastidieux pour une application orientée console (et ne disposant pas encore de chargement de fichiers).

Le problème c'est que pour l'instant soit vous tapez tout à la main, soit vous lancez autoCin mais dans ce dernier cas vous ne pouvez pas reprendre la main pour tester interactivement. L'idéal serait de pouvoir commencer à tester interactivement **après** avoir mis un maillage en place automatiquement (comme un chargement depuis un fichier en somme). Ce n'est pas très compliqué, il suffit d'ajouter un case spécial dans votre menu, avec une valeur qui veut dire « fin autocin » et qui sera mise à la fin du fichier de script. Editer test1.txt (par exemple depuis Code::Blocks File Open...) pour mettre -10 à la fin **à la place de 0**, et ajouter le code suivant à votre menu. Tester.

```
case -10: /// Spécial, fin de script autoCin
   util::stopAutoCin();
   afficherMenu();
   break;
```

1 Il est souvent conseillé de séparer d'un côté les .cpp dans src/ et de l'autre les .h dans une **arborescence parallèle** dans include/. Ce conseil est valable pour des codes matures et/ou qui deviennent des bibliothèques. Pour l'instant notre base de code est immature et risque d'être déménagée, il est préférable de ne pas avoir à maintenir séparément 2 arborescences parallèles. Voir https://stackoverflow.com/a/2924217 et https://stackoverflow.com/a/31581208

4. Travailler avec la cocotte en presque temps réel (expérimental)

De même remplacer 0 par -10 dans cocotte.txt et vérifier que vous pouvez bien reprendre la main après avoir « chargé » la cocotte, par exemple ajouter un point et un triangle à la cocotte. Notez qu'il est un peu fastidieux de devoir à chaque opération taper 1 en console pour redessiner la scène, basculer sur le navigateur et faire Ctrl+r pour recharger output.svg. Nous allons essayer d'améliorer ça, mais **c'est expérimental et ça peut ne pas marcher** (dans ce cas rester sur le protocole de rechargement manuel).

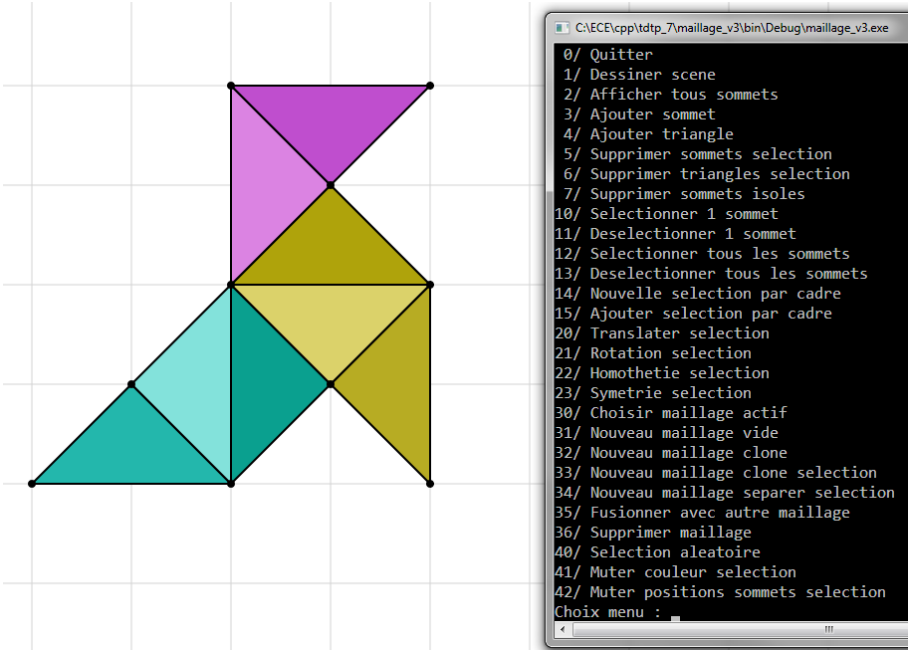
D'abord pour ne pas avoir à taper 1 pour dessiner la scène à chaque opération on peut ajouter ce code **à la fin de** la boucle de menu :

```
        /// Dessin automatique à chaque opération
        {
            Svgfile::s_verbose = false;
            Svgfile svgout;
            svgout.addGrid();
            calqueCourant->dessiner(svgout, styleDessin);
        }
    }
    while ( choix!=0 );
```

Ensuite, le fichier index.html fourni (qui n'est pas un modèle de code javascript...) fait un auto-reload toutes les secondes : **ouvrez index.html dans votre navigateur favori** et si il ne marche pas pour la suite testez un autre navigateur. Normalement déjà vous devez voir la cocotte qui était encore dans output.svg, avec un compteur qui s'incrémente automatiquement 1 fois par seconde. Re-testez l'ouverture de la cocotte avec votre appli et mettez côte à côte l'appli avec le navigateur. Ajoutez un sommet, un triangle... normalement l'affichage coté navigateur doit suivre automatiquement (avec un léger délais d'une seconde au plus). Si ça coince essayez de faire un reload manuel sur le navigateur (ce qui devrait remettre le compteur à 0). Certains navigateurs (IE) peuvent faire des saccades à chaque reload ce qui est agaçant. Firefox est nickel mais décroche parfois. Chrome, Safari... non testés.

5. Les conditions sont réunies, il est temps de muter des cocottes !

Ci dessous le menu « final » à obtenir. Ce n'est pas le projet. Le projet sera plus ouvert. Mais réaliser tout ou partie de ces fonctionnalité sera sûrement un bon investissement. Pour l'instant on va se concentrer sur les options **40 41 et 42**. Voir résultat page suivante.

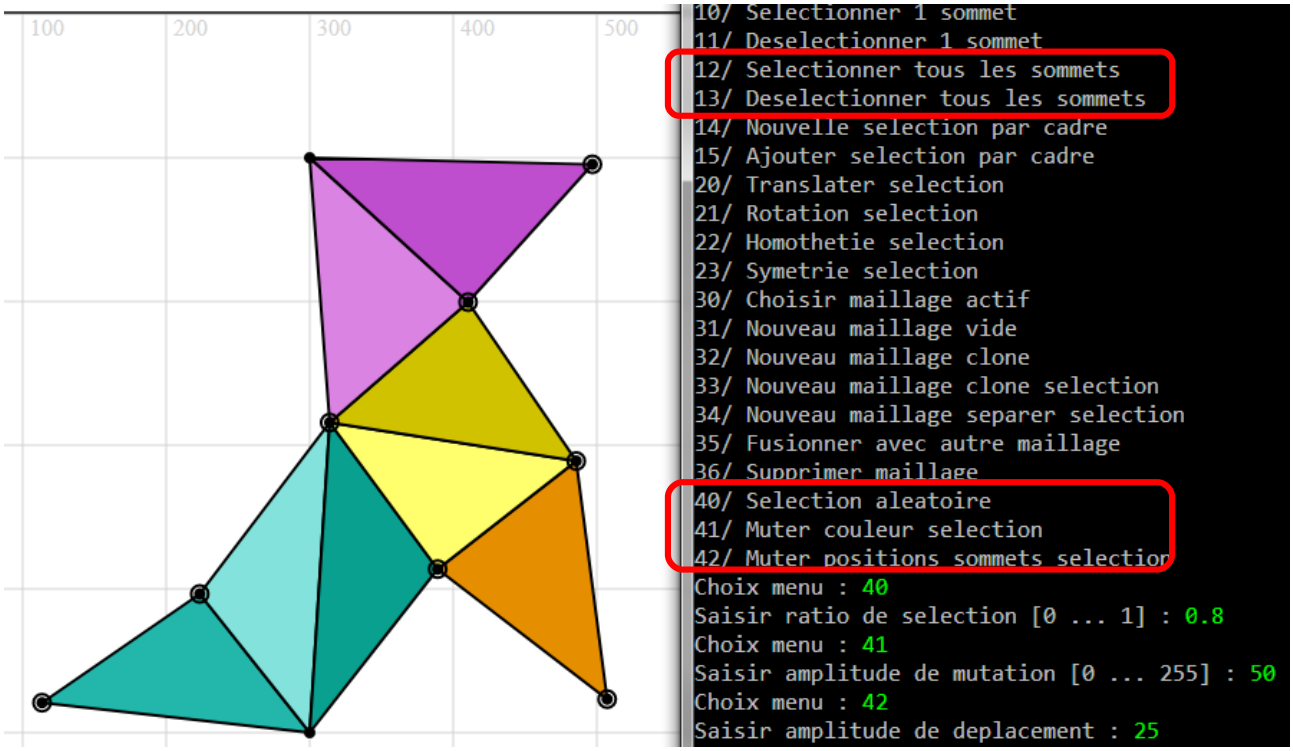


Voilà ce que ça peut donner. La sélection aléatoire utilise une nouvelle fonction aléatoire disponible dans util.h, surcharge de alea au domaine des réels :

```
/// Cette fonction retourne un réel aléatoire dans [min...max[
double alea(double min, double max);
```

Pour réaliser une action selon un ratio (0.8 => 80% de chance)

```
if ( util::alea(0.0, 1.0) < ratio )
```



40 : On voit sur l'exemple que 80 % des sommets ont aléatoirement été choisis pour être sélectionnés (une précédente sélection aurait d'abord été vidée => tous les attributs sélection des sommets à false). Naturellement à chaque fois qu'on identifie un besoin comme ça on fait une méthode ! On en profite pour compléter le menu avec 2 nouvelles options pas difficiles **12 et 13**.

41 : On sélectionne des sommets, pas des triangles. Pour des opérations sur les triangles on considère qu'un triangle est sélectionné si ses 3 sommets sont sélectionnés. Une nouvelle méthode bool getSelection(); dans la classe Triangle retournera Vrai si les 3 sommets sont sélectionnés. De cette façon la méthode de Maillage qui mute les couleurs (des Triangles) est propre.

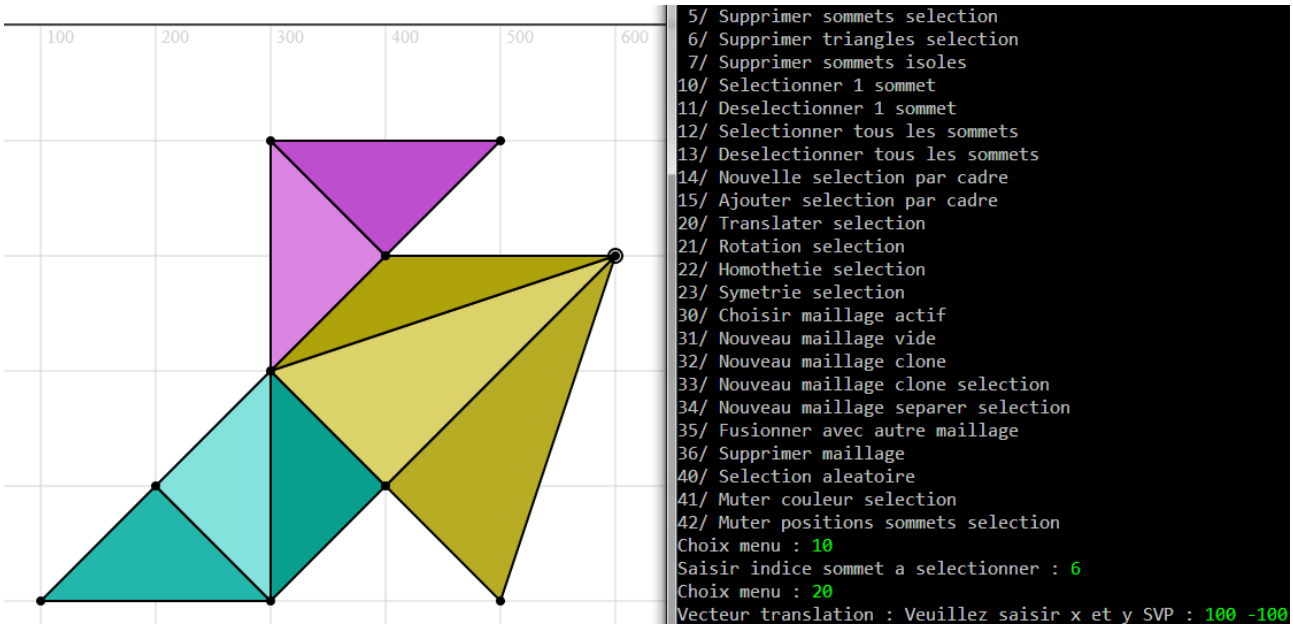
42 : La classe Coords fournie propose une nouvelle méthode pour avoir des Coords aléatoires qui peuvent servir de vecteur de translation aux sommets sélectionnés à « muter » en position. Il est évidemment préférable de tout de suite prévoir une méthode muterPosition dans la classe Sommet. Ensuite il ne reste plus qu'à appeler cette méthode, pour les sommets sélectionnés, depuis une méthode de la classe Maillage.

```
void Sommet::muterPosition(double amplitude)
{
    m_position = m_position + Coords::aleatoire(amplitude);
}
```

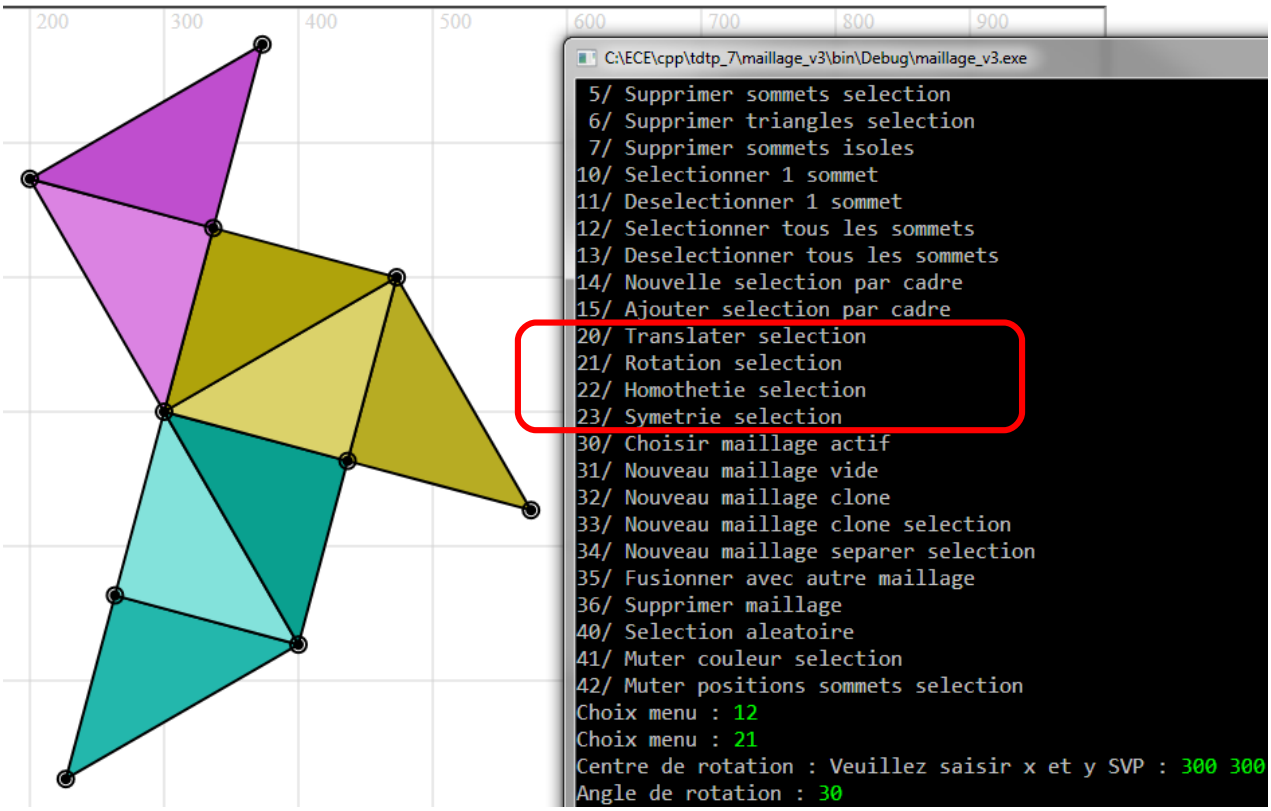
Note : la méthode **`Coords::aleatoire`** est particulière, c'est une méthode « static » qui ne part pas d'un objet cible, on met le nom de la classe suivi de :: pour l'appeler. Ceci pour éviter des ambiguïtés de constructeurs surchargés...

6. Transformations affines de morceaux de cocottes

La fonction transfoTest (main.cpp) vous a montré comment utiliser la classe Transformation. Vous allez pouvoir appliquer ces transformations aux coordonnées des sommets sélectionnés.
20 : Commencer par la translation. Prévoyez tout de suite une seule et même méthode **pour toutes les transformations** dans Maillage : void selectionTransformer(const Transformation& trans);



21 : Rotation, il faut un centre et un angle. Et si vous voulez transformer la cocotte entière, commencer par sélectionner tout grâce à **12** …

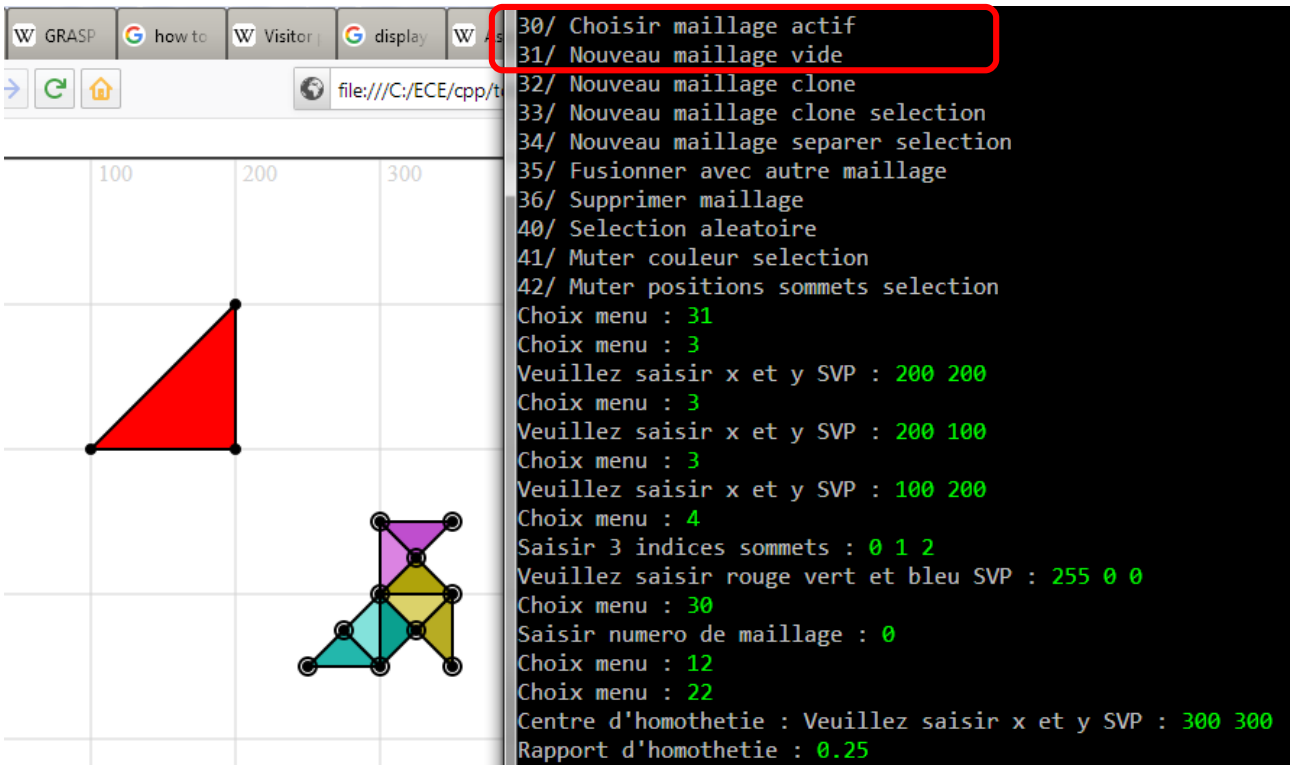


22 et 23 : Homothétie et symétrie sur le même principe...

7. Clonage de cocottes

Le point difficile. D'abord pour cloner des cocottes il faut savoir ou va atterrir le clone. Pour l'instant notre application ne gère qu'un seul maillage à la fois. Il va falloir mettre en place une nouvelle classe qui sera une collection de maillages. Je propose une classe Calque, qui ira se ranger dans le répertoire Page (une page sera composée de plusieurs calques. Plus tard). L'attribut principal de calque sera un vecteur de pointeurs sur Maillage. Elle aura une méthode dessiner qui déléguera l'appel aux maillages qu'elle contient (pour chaque Maillage sur le calque, appeler sa méthode dessiner). Elle aura aussi une méthode ajouterMaillage et enleverMaillage (pas forcément le détruire, il faudra voir, selon l'usage). Et peut-être d'autres méthodes utiles selon les besoins.

Mettre en place cette classe Calque et modifier le menu de l'éditeur pour autoriser le travail avec plusieurs maillages, choix **30** et **31** et **36**. Vérifier.



Enfin nous pouvons travailler avec plusieurs maillages simultanément, ça veut dire qu'on peut cloner des Maillages : le clone pourra atterrir dans le même Calque que l'original. Mais comment cloner, je vous ai dit que c'est difficile ! Pas tant que ça... à condition d'organiser l'opération au niveau de la classe composite Maillage et avec l'aide des composants...

On va déclarer et définir le constructeur par copie de Maillage :

```
Maillage(const Maillage& src);
```

Il reçoit une référence au Maillage original. Dès le départ du constructeur l'objet this existe, mais ses collections sont vides. Il va falloir remplir ses collections sommets et triangles avec des copies de ceux des originaux, mais en transposant les adresses des sommets dans le nouveau contexte. Pour ça nous allons déclarer localement un conteneur associatif, une map (nous y voilà)

```
/// Associer à chaque adresse sommet de la source l'adresse de son clone
std::map<Sommet*, Sommet*> transpose;
```

On va d'abord cloner les sommets de l'original vers la copie en cours de construction, directement, avec le constructeur par copie implicite de Sommet : Sommet(const Sommet&) = default;

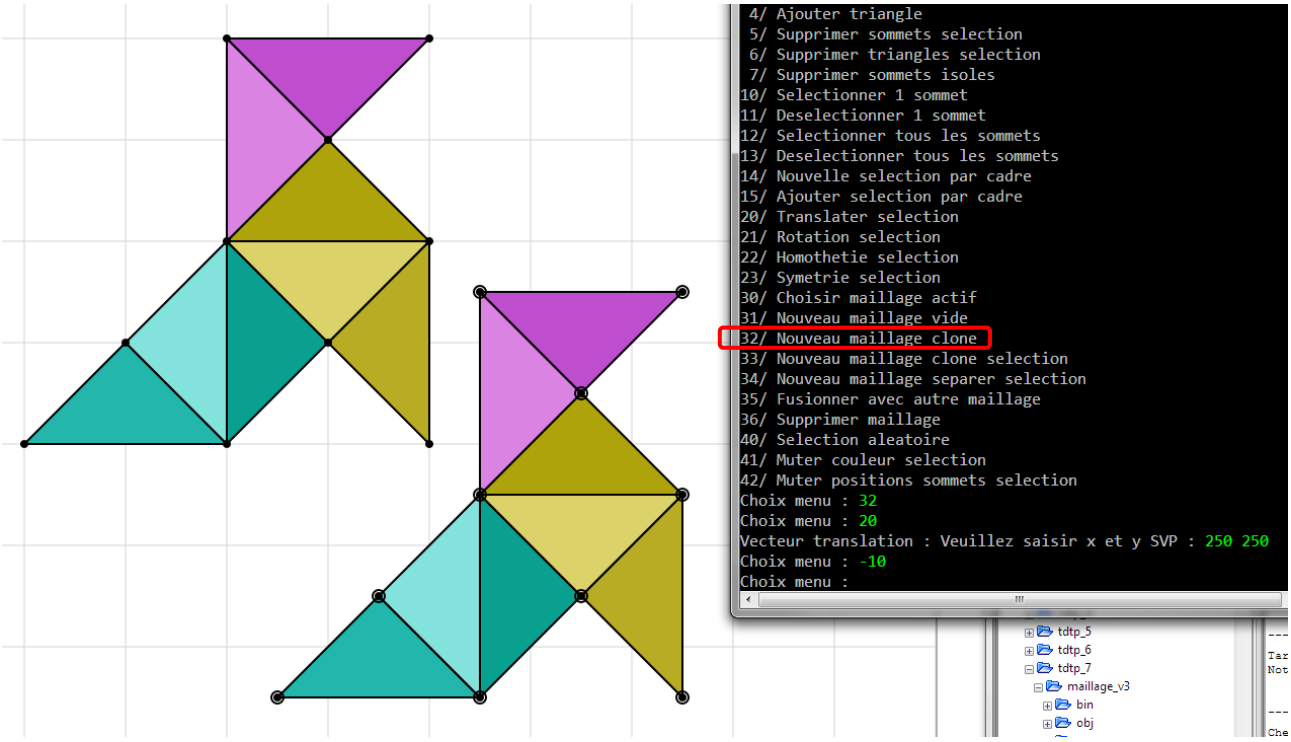
Lors de ce clonage on utilisera bien new pour re-fabriquer des adresses vers de nouveaux objets sommets. **On remplira au fur et à mesure la map transpose** avec map[adresse de sommet de l'original] = adresse de sommet du clone

Ensuite on copiera les Triangles en prenant bien soin de faire en sorte que chacune des 3 adresses vers sommet de l'original soit remplacée par l'adresse vers sommet du clone :

triangle clone adresse vers sommet 1 = transpose [triangle original adresse vers sommet 1]

etc... de telle sorte que les références aux sommets de l'original soient « transposées » vers des références aux sommets du clone. Et voilà !

Le clone apparaît au même endroit que l'original. Pour bien le distinguer de l'original il va falloir le translater. Je suggère que le clone devienne le nouveau Maillage actif automatiquement et que tous ses sommets soient sélectionnés afin de faciliter cette opération de dégagement.



8. Fusion des cocottes, séparation et chirurgie ablative

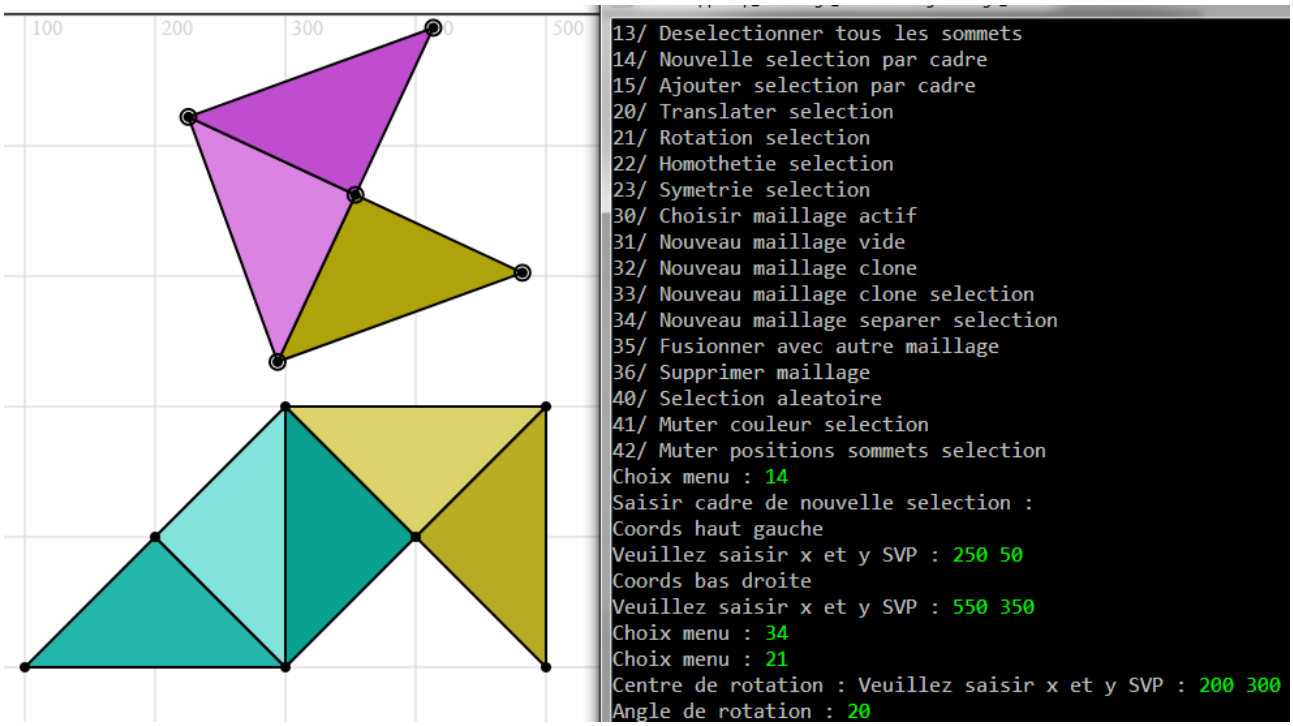
Les autres opérations peuvent être implémentées plus facilement avec un peu de soin

0/ Quitter	20/ Translater selection
1/ Dessiner scene	21/ Rotation selection
2/ Afficher tous sommets	22/ Homothetie selection
3/ Ajouter sommet	23/ Symetrie selection
4/ Ajouter triangle	30/ Choisir maillage actif
5/ Supprimer sommets selection	31/ Nouveau maillage vide
6/ Supprimer triangles selection	32/ Nouveau maillage clone
7/ Supprimer sommets isoles	33/ Nouveau maillage clone selection
10/ Selectionner 1 sommet	34/ Nouveau maillage separer selection
11/ Deselectionner 1 sommet	35/ Fusionner avec autre maillage
12/ Selectionner tous les sommets	36/ Supprimer maillage
13/ Deselectionner tous les sommets	40/ Selection aleatoire
14/ Nouvelle selection par cadre	41/ Muter couleur selection
15/ Ajouter selection par cadre	42/ Muter positions sommets selection

La suppression des sommets isolés peut se faire efficacement avec un conteneur ensembliste en parcourant les triangles : une méthode de triangle reçoit en paramètre par référence ce set et y ajoute les adresse des 3 sommets qu'il utilise.

```
/// Ensemble des sommets effectivement référencés
std::set<Sommet*> utilises;
```

L'opération d'effacement des sommets isolés facilite l'opération de séparation. Pour ces opérations il est commode de pouvoir sélectionner avec des cadres. Jetez un coup d'œil à la classe Cadre (répertoire geometrie) qui peut aider. Sur la figure ci-dessous on a bien 2 maillages sur le calque.



9. L'attaque des cocottes mutantes

En combinant et en répétant des clonages, des translations, des fusions, des mutations, on obtient des scènes de simulation de combat dignes des meilleurs films de science-fiction.
Voir page suivante.