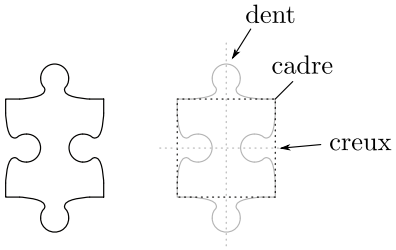
Projet de programmation

Le but de ce projet est d'écrire en `java`, à l'aide de la librairie `javafx`, une application permettant de jouer au jeu de puzzle[1], dans sa forme la plus classique. L'application devra permettre de créer des puzzles de taille libre par découpage aléatoire d'images, et de sauvegarder ou charger des puzzles en cours de résolution.

1 Le jeu de puzzle

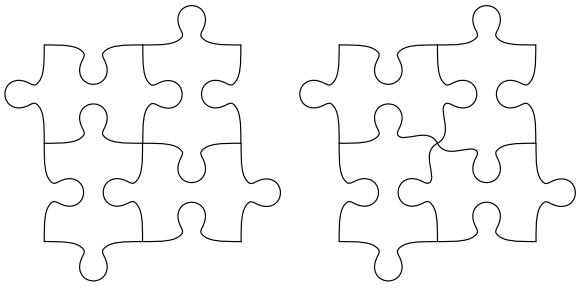
Les pièces d'un puzzle classique sont formées d'un cadre de forme à peu près rectangulaire, le long duquel s'inscrivent, sur chacun des quatre bords, soit une dent, soit un creux. Par exemple, les pièces à deux dents opposées et deux creux opposés sont toutes des variantes de la pièce canonique suivante :



Afin de limiter les possiblités d'emboîtement de pièces – et donc les risques d'erreurs d'assemblage – les pièces d'un puzzle s'écartent en général de cette forme canonique par plusieurs sortes de déformations. Nous les présentons ici individuellement, mais il est évident qu'elles peuvent se combiner les unes aux autres.

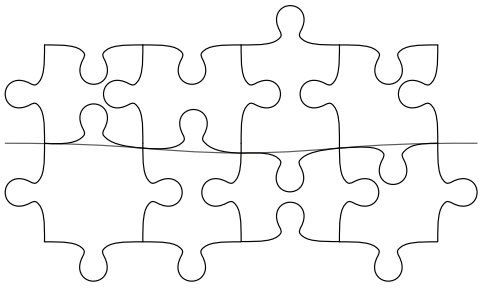
1.1 Déformations locales de cadres

Les cadres de quatres pièces en contact par coin peuvent être déformés par torsion de leur axes autour du point central, comme dans la figure ci-dessous (sans déformation à gauche, avec déformation à droite) :



1.2 Déformations globales de cadres

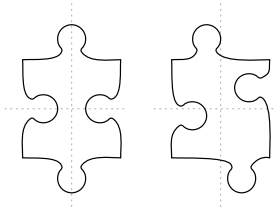
Les lignes verticales ou horizontales sur lesquelles s'inscrivent les bords des cadres peuvent être ondulées :



[1] https://fr.wikipedia.org/wiki/Puzzle

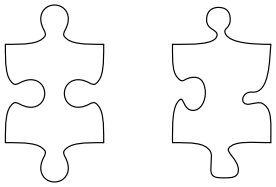
1.3 Déplacements des dents et creux

Les dents et creux peuvent être déportés des axes de symétrie du cadre :



1.4 Déformations des dents et creux

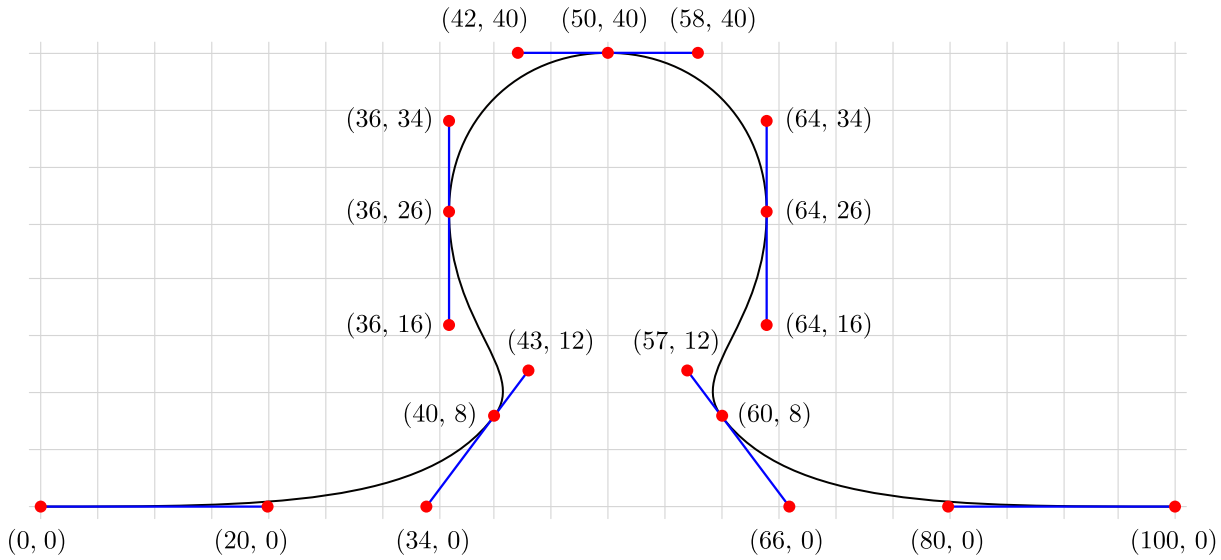
Les dents et creux sont sujets à de multiples variantes de formes et de taille :



2 Création d'un puzzle

Votre application devra permettre de construire un puzzle à partir d'une image quelconque – éventuellement avec un ajustement mineur de ses proportions ou de sa taille. Le nombre de pièces en longueur et en largeur devra pouvoir être librement choisi par l'utilisateur parmi un ensemble de choix possibles, ou si nécessaire en rentrant ces données au clavier. Le decoupage en creux ou en dents des pièces ainsi que les déformations se feront de manière aléatoire.

La création d'une pièce canonique peut se faire à l'aide des classes prédéfinies `MoveTo`, `CubicCurveTo` et `Path` et du diagramme suivant :



L'implementation des différentes sortes de déformations peut se faire par altération des positions de points de pièces canoniques et celles de leurs points de contrôle. Noter que ces altérations sont de difficultés variables : les déplacements de dents et de creux sont simples; les déformations de dents et de creux peuvent se faire par altérations mineures des positions des points et points de contrôle intermédiaires dans la courbe ci-dessus; une déformation locale de cadre peut se faire par rotation des points de contrôle des extrémités de la courbe; la déformation globale de cadre est la plus difficile.

Dans tous les cas, les choix aléatoires devront éviter les superpositions ou le croisement de creux ou de bosses dans une même pièce : il s'agit de la difficulté majeure de ce traitement.

3 Conception de l'interface

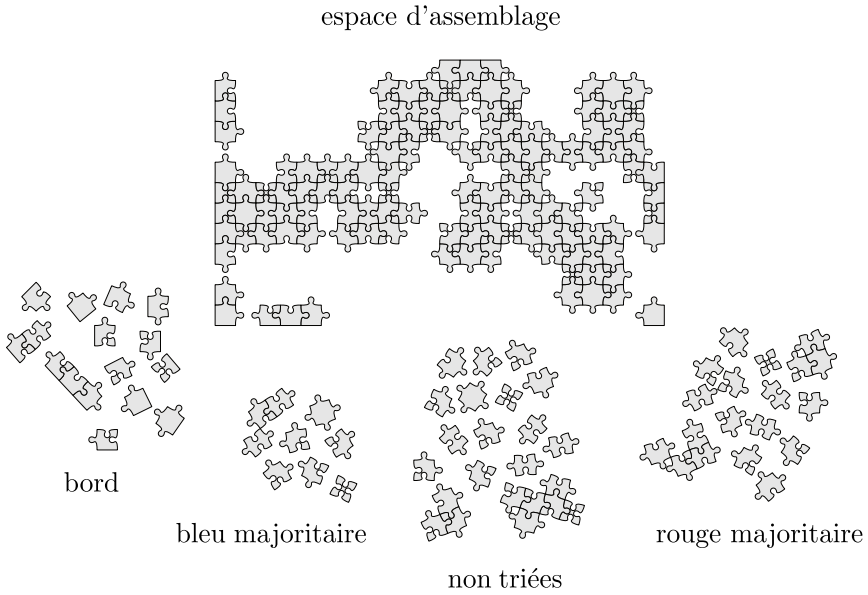
Le lancement de l'application démarrera celle-ci dans un état neutre : fenêtre vide, aucune interaction avec l'utilisateur tant qu'il n'aura pas effectué au moins une action. Un menu permettra à la demande de créer un nouveau puzzle, d'en charger un en cours de résolution, ou encore de quitter l'application.

3.1 Principe général

La résolution d'un puzzle complexe demande de la place, beaucoup de place. L'idéal est de disposer d'une table suffisamment grande sur laquelle les pièces sont toutes placées à plat, face découverte.

Une zone de la table est dédiée à l'assemblage proprement dit. Des zones secondaires sont dédiées au tri des différentes sortes de pièces non encore placées, *e.g.* pièces du bord, pièces de même couleur majoritaire, pièces de formes exotiques, etc. Une ou plusieurs zones secondaires peuvent constituer des réserves de pièces non encore triées.

Il est courant de commencer l'assemblage de portions du puzzle au sein d'une zone secondaire alors que ces portions n'ont pas encore été placées, afin de placer plus tard ces portions en bloc.



Votre interface devra, durant la phase de jeu, respecter le principe de cette séparation en zones, une principale et un nombre libre de secondaires, chaque zone étant inscrite dans sa propre fenêtre.

Au lancement du jeu, et si l'utilisateur décide de créer un puzzle : la fenêtre initiale de l'application sera celle de la zone principale; une seconde fenêtre s'ouvrira sous celle-ci, correspondant à la première zone secondaire. Dans cette dernière, on placera toutes les pièces du puzzle avec un placement et une orientation alátoire, sans aucun chevauchement.

Toute zone pourra à chaque instant contenir des pièces ou des assemblages de pièces non encore placés à leur position définitive, mais du point de vue de l'utilisateur, ces éléments simples et multiples seront maniés avec les mêmes gestes.

3.2 Gestion individuelles des zones

3.2.1 Déplacement, rotation et sélection

L'utilisateur doit pouvoir librement déplacer un élément ou lui faire subir une rotation dans une même zone. Il doit pouvoir également transférer par drag-and-drop des éléments d'une zone à une autre.

Il doit pouvoir sélectionner tout ou une partie des éléments d'une zone par raccourci clavier, par tracé d'un cadre de sélection, par sélections individuelles multiples, ou encore par une combinaison de ces différents gestes. Il doit pouvoir faire subir à un groupe d'éléments sélectionnés une rotation, ou encore un déplacement global dans cette zone ou vers une autre.

Un élément ou un groupe d'eléments en peut survoler d'autres pendant sa rotation ou son déplacement, mais aucun élément ne peut être reposé sur un ou plusieurs autres. Noter que l'usage approprié de scroll-bars peut servir à compenser le manque de place dans une zone.

3.2.2 Pose et assemblage

La pose d'un élément à sa position correcte dans la zone d'assemblage est considérée comme définitive. Dans chaque zone, l'utilisateur doit également pouvoir assembler deux éléments non encore placés, par déplacement de l'un à proximité de l'autre. Les poses ou assemblages incorrects sont interdits. Lorsqu'elle est possible, une pose ou un assemblage est effectuée immédiatement après la fin du déplacement ou de la rotation d'un élément, avec une certaine marge de tolérance dans son orientation et sa position, et par effet d'aimantation.

3.3 Créations de zones et interactions entre les zones

L'utilisateur doit pouvoir demander la création d'une nouvelle zone (et donc d'une nouvelle fenêtre), lui donner un nom (le titre à la fenêtre). Il doit pouvoir gérer librement ces zones et leur contenu : supprimer une zone vide, supprimer une zone en transférant son contenu dans une autre par drag-and-drop, vider une zone dans une autre mais sans la supprimer, renommer une zone, etc.

3.4 Gestion de la partie

Le joueur doit pouvoir sauvegarder la table de jeu dans sa configuration exacte – zones et éléments de zones avec leur placement et orientation, fenêtres placés de la même manière – et la recharger à tout moment.

3.5 Extensions

Afin de consolider votre projet, toute fonctionnalité autre que celles mentionnées ci-dessus sera la bienvenue.

- Outre le chargement ou la création de puzzle, les autres opérations possibles au lancement pourraient être la configuration globable de l'application (choix d'un style, de couleurs, de fonts, redéfinition de raccourcis, etc) – avec, bien sûr, sauvegarde automatique d'un fichier de configuration permettant de retrouver celle-ci au prochain lancement.
- A la demande du joueur, l'application pourrait proposer visuellement une aide plus ou moins précise sur le placement d'un élément : placement automatique, indication de la position exacte dans la zone principale, indication d'un ou plusieurs autres éléments assemblables avec le premier dans la même zone ou dans une autre, assemblage automatique, résolution simulée du puzzle avec ou sans retour à la configuration initiale, etc.
- D'autres formes d'aide sont envisageables : tri automatique des bords, tri automatique des pièces de couleurs proches avec une certaine marge de tolérance.
- L'implémentation d'un Undo/Redo serait un plus pour l'interface.
- On peut aussi envisager des pièces hexagonales (cela demande un haut niveau d'abstraction dans la conception du programme), etc.

4 Critères d'évaluation

4.1 Qualité de la conception objet et du code

Ce cours est aussi un cours de programmation, plus précisément un cours de programmation objet. Servez-vous de l'héritage, des interfaces et de la liaison dynamique partout où ces éléments améliorent la factorisation du code. Prenez le temps de réfléchir à la hiérarchie nécessaire, et à la répartition des données et des traitements.

Ne programmez pas (jamais) par copier-coller.

4.2 MVC

Vous avez vu en cours l'architecture MVC, et ce type de projet se prête idéalement à son usage : votre code sera à la fois mieux organisé, plus facile à développer, plus facile à débugger, et valorisé par le choix de cette architecture si elle est réalisée dans sa forme la plus académique.

4.3 Utilisation de FXML

Nous ne vous obligeons pas à l'utiliser, mais il est clair que cela mettrait en valeur votre projet.

4.4 Ergonomie et esthétique de l'interface

Comme son intitulé l'indique, ce cours est un cours d'interfaces graphiques. Votre interface doit être souple, intuitive, naturellement utilisable par un utilisateur même s'il la découvre pour la première fois. Le recours au clavier ne doit se faire que s'il est indispensable, et suivre les standards habituels (vous les utilisez sans même y penser).

Aucune information déjà rentrée (configuration, choix d'un répertoire de lecture ou de sauvegarde, etc.) ne doit être redemandée à l'utilisateur, y compris d'une exécution à l'autre.

5 Modalités

5.1 Groupes

Le projet doit être réalisé de préférence en binôme – aucun travail en trinôme ou plus ne sera accepté. La répartition des tâches au sein d'un groupe doit être raisonnablement équilibrée. En cas de déséquilibre avéré, les notes finales pourront être individualisées.

5.2 Individualité de chaque projet

De manière évidente, votre code doit être strictement personnel. Nous pouvons tolérer l'emprunt ou l'adaptation d'exemples repris sur le site officiel d'Oracle, mais pas la reprise de code trouvé sur le web ou venant d'une quelconque "aide" trouvée sur un forum[2], encore moins le partage de code entres groupes[3]. Il relève de votre responsabilité de faire en sorte que votre code reste inaccessible aux autres groupes : par exemple, si vous vous servez d'un dépôt `git`, ce dépôt doit être privé.

5.3 Forme du rendu

Les modalités et dates de rendu seront précisées ultérieurement. Votre rendu consistera en:

- un code-source écrit en `java`, compilable et utilisable tel quel sous Linux et/ou sur les ordinateurs de l'UFR,
- un fichier texte nommé README contenant vos noms,
- un rapport de quelques pages au format PDF décrivant votre projet et les extensions réalisées, et expliquant et justifiant les choix de conception ou d'implémentation,
- tout autre fichier nécessaire à la compilation et à l'exécution.

Tous ces éléments seront placés dans une unique archive compressée en `.tar.gz`.

L'archive devra s'appeler *nom1*`-`*nom2*`.tar.gz`, et s'extraire dans un répertoire *nom1*`-`*nom2*`/`, où *nom1* et *nom2* sont les noms des deux personnes constituant le groupe. Par exemple, si vous vous appelez Denis Diderot et René Descartes, votre archive devra s'appeler `diderot-descartes.tar.gz` et s'extraire dans un répertoire `diderot-descartes/`.

[2] Vous savez vous servir d'un moteur de recherche; nous aussi.

[3] La soutenance de projet est un examen comme un autre. Le plagiat en projet constitue une fraude aux examens, passible au pire de lourdes sanctions discplinaires – ce cas s'est produit plus d'une fois dans cette UFR.