1 Modalités

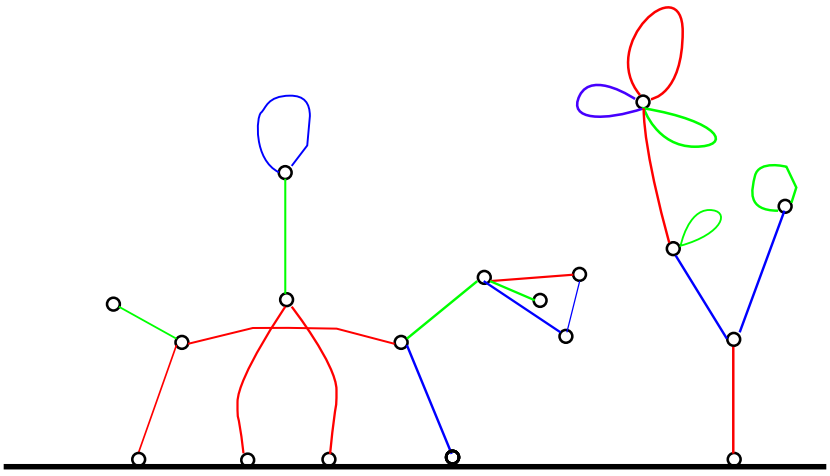
Le projet doit être écrit en JavaFX, et réalisé en binôme (éventuellement, en monôme). Les soutenances auront lieu aux alentours de début mai, la date précise vous sera communiquée ultérieurement. Pendant la soutenance, les membres d'un binôme devront chacun montrer leur maîtrise de la totalité du code. Vous devrez déposer les projets sur Moodle sous la forme d'une archive se décompressant en un dossier portant les noms de votre binôme – la date limite du rendu sera la veille des soutenances [1].

La qualité du code, sa conception objet, ainsi qu'un usage approprié des fonctionnalités de JavaFX, auront un impact important sur la note. En particulier, partout où cela est possible nous préférons :

— l'utilisation de fxml et de fonctions « callback » déclarées dans fxml, et définies dans les contrôleurs à la place de handlers,
— des « bindings » plutôt que des `ChangeListeners`.

2 Le jeu Hackenbush

Le jeu Hackenbush se joue à deux joueurs, Bleu et Rouge, sur un dessin composé d'un certains nombre de sommets auquels sont attachées des arêtes coloriées en bleu, rouge ou vert. Une partie de ces sommets sont disposés sur une ligne noire symbolisant le sol. Le dessin peut par exemple ressembler à ceci :



1. Pour le projet développés sur NetBeans, il suffit de suivre `File` → `Export Projet` → `to ZIP`

Les sommets attachés aux extrémités d'une même arête sont dit *adjacents*. Les deux extrémités d'une arête peuvent être attachées au même sommet, *ie.* un sommet peut être adjacent à lui-même. Un *chemin* est une suite de sommets dans laquelle tout élément sauf le dernier est adjacent à son successeur. Le premier élément d'une telle suite est dit *connecté* au dernier.

Tout sommet connecté à un sommet situé sur la ligne noire du dessin est dit *connecté au sol*. Noter que, nécessairement, deux sommets adjacents sont ou bien tous deux connectés au sol, ou bien tous deux non connectés au sol.

Préparation du jeu. Dans le dessin initial, chaque extrémité d'arête doit être attachée à un sommet, et tout sommet doit être connecté au sol.

Déroulement du jeu. Les joueurs jouent à tour de rôle, en commançant par Rouge. A son tour, chaque joueur choisit une arête : soit une arête de sa couleur, soit une arête verte. Le dessin est alors modifié de la manière suivante :

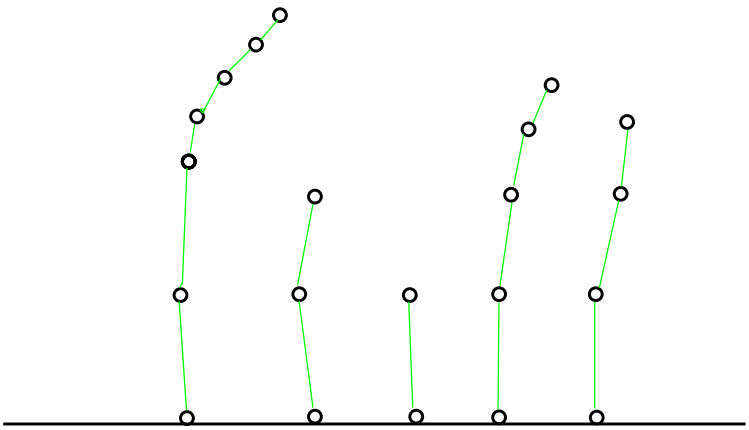
1. L'arête choisie est supprimée du dessin.
2. Les sommets qui ne sont plus connectés au sol sont supprimés du dessin, ainsi que toutes les arêtes attachées à ces sommets.

Noter qu'au terme de cette transformation, les sommets restants sont tous connectés au sol, et les arêtes restantes ont toutes leurs deux extrémités attachées.

Fin du jeu. Le jeu se termine au moment où le joueur dont c'est le tour n'a plus de coup jouable. Dans le jeu *normal*, son adversaire est alors déclaré gagnant. Dans le jeu de type *misère* en revanche, son adversaire est déclaré perdant. Le choix du type de jeu doit évidemment être convenu en début de partie.

2.1 Le jeu de Nim

Le jeu de Nim est un cas spécial de Hackenbush. Dans le jeu de Nim, le dessin est formé de suites filiforme d'arêtes (ou *tiges*), tout vertes, comme dans le dessin ci-dessous :



Il existe pour le jeu de Nim un algorithme permettant de déterminer le joueur gagnant et, à chaque tour de ce même joueur, quel est le coup gagnant. Nous vous fournirons la classe Java implémentant pour les deux variantes normale et misère la stratégie gagnante pour Nim.

3 Interface graphique

Votre travail consistera à programmer une interface graphique permettant à l'utilisateur de créer un dessin permettant de jouer à Hackenbush (les modes "édition générique" et "édition Nim"), et de jouer sur un dessin créé (le mode "jeu").

3.1 Mode édition générique

Cette partie de l'application permettra de construire à la souris (éventuellement en s'aidant du clavier lorsque c'est indispensable) le dessin du jeu. Votre implémentation de l'éditer de dessin doit satisfaire les conditions suivantes :

(1) Votre interface doit permettre d'ajouter facilement des sommets au dessin, et de construire une nouvelle arête entre deux sommets déjà existants. Les arêtes, ainsi que les sommets qui ne sont attachés à aucune arête, doivent pouvoir être supprimés.

(2) Chaque sommet doit pouvoir être librement déplacé à la souris. Les arêtes attachées à un sommet devront, visuellement, rester attachées à lui pendant toute la durée de son déplacement.

(3) La couleur d'une arête doit être facilement modifiable.

(4) L'utilisateur doit pouvoir sélectionner une partie des éléments du dessin, la dupliquer, puis déplacer cette copie.

(5) Nous apprécierons la possiblité d'effectuer sur une partie séléctionnée d'autres types d'opérations : copier/couper/coller, translation, rotation, changement d'échelle, etc.

(6) Nous apprécierons aussi la possibilité de construire des arêtes de formes quelconques (Path) et pas seulement rectilignes. L'utilisateur pourra modifier la forme de l'arête (sans bouger les sommets aux deux extrémités), par exemple en déplaçant à la souris des points de contrôle.

3.2 Mode édition Nim

Ce mode doit permettre à l'utilisateur de construire un dessin pour le jeu de Nim en spécifiant le nombre de tiges souhaité et le nombre d'arêtes de chaque tige, le programme construisant le dessin associé à partir de cette seule spécification. Pour des raisons esthétiques, votre programme pourra construire des arêtes qui diminuent en longueur quand elles s'éloignent du sol.

3.3 Mode jeu

Le mode jeu doit permettre à l'utilisateur de jouer sur un dessin créé à l'aide de l'un des deux modes précédents, soit en mode normal, soit en mode "misère". Pour les dessins créés à l'aide de l'éditeur générique, l'utilisateur devra pouvoir choisir entre jouer contre un adversaire humain, ou joueur contre l'ordinateur.

Pour le jeu de Nim, l'utilisateur pourra aussi faire le choix de faire jouer l'ordinateur contre lui-même [2]. Dans ce cas, il devra pouvoir réguler la vitesse de jeu (le temps entre les coups) à l'aide d'un `Slider`. En particulier, la vitesse 0 mettra la partie en pause.

A chaque étape du jeu, l'application indiquera clairement le joueur dont c'est le tour. Toute tentative d'un joueur pour supprimer une arête de la couleur adverse doit provoquer l'affichage d'un message d'avertissement. (`Alert`). Le programme devra également détecter le moment où l'un des deux joueurs est gagnant, et annoncer dans ce cas la fin de la partie et le vainqueur.

4 Extensions

La section précédente décrit la partie obligatoire du projet. Voici à présent quelques suggestions pour consolider votre rendu.

(a) Utiliser les effets (`Effect`) et les classes dérivées pour embellir les dessins.

(b) Un peu d'animation ? Par exemple, une arête supprimée fera disparaître progressivement la partie du dessin qui n'est plus connectée au sol (en quelques secondes). Ou encore, la partie du dessin qui devient déconnectée du sol s'envole, emportée par le vent.

(c) Implémenter un undo/redo pour le dernier coup (dernier coup du joueur courant), ou plusieurs (remonter dans l'historique).

(d) Implémenter la sauvegarde des dessins de l'éditeur générique (dans un fichier fxml, et peut-être dans un fichier supplémentaire pour les informations qui ne peuvent pas être mis dans fxml).

Cette fonctionnalité augmentera grandement la qualité et l'intérêt pratique de votre application. Elle peut, hélas, s'avérer difficile à écrire, et n'est pas vraiment pertinente pour l'interfaçage graphique, donc ce n'est pas une tâche prioritaire.

2. Comme indiqué ci-dessus, et à moins que cela ne vous intéresse, inutile de chercher l'algorithme optimal pour Nim : nous vous fournirons la classe appropriée