Programmation Fonctionnelle Avancée (Scheme)
2017/2018

Projet : Visualisation de Graphe
Nom du projet : ***GraphX***
Réalisé par : ROMDHANE Mohamed-Amine, ZHANG Noé

• Rapport de projet :

Après ces quelques semaines nous vous présentons le rendu final de notre projet GraphX. Celui-ci encore loin d'être parfait à cependant les mérites nous le pensons, d'être présentable.

Vous trouverez ci-dessous un rapport sur le déroulement du projet, mais aussi de la documentation sur celui-ci, si plus d'informations sont nécessaires sur les divers modules ou fonctions, elles seront présentes dans le code lui-même cordialement.

Déroulement du projet :

Tout d'abord, lors du commencement de ce projet, nous avons commencé par créer les différents modules proposés simplement, mais sous différentes formes, nous avons aussi pu réaliser le projet en POO, mais pour un meilleur rendu et plus de facilité, nous sommes tournés vers une programmation des fonctions simple. Les modules nécessaires à la réalisation du projet ainsi que leurs principaux tests ont pu être rapidement réalisés, nous permettant de nous concentrer sur les "fonctionnalités avancées" de ce projet. Nous avons commencé par nous tourner vers les fonctionnalités les plus simples tels que : play, pause, avance rapide… Puis une fois celles-ci fini, nous avons décidé de continuer avec l'interaction entre l'utilisateur et le canvas avec comme des fonctions telles que l'ajout de nœuds ou d'arêtes, qui ont d'abord été implémentées en mode aléatoire puis à la volonté de l'utilisateur. De même pour la suppression de nœuds et d'arêtes. Une fois la plus grande partie des fonctions avancées proposées fini, il nous sembla important de porter notre attention sur l'interface graphique afin de proposer un rendu à la fois visuellement correct mais aussi simple d'utilisation pour l'utilisateur. Plus tard, nous vînt à l'idée d'implémentation une fonctionnalité permettant le changement de couleur du graph, qui fut la première insérer de notre plein gré. Par manque d'idée, nous nous sommes mis d'accord afin de pouvoir implémenter une représentation graphique en trois dimensions à notre projet, pour ce faire nous avons utilisé la libraire pict3D. Nous avons donc commencé par réécrire les différents modules afin qu'il soit compatible avec une telle représentation. Cependant, la plus grande partie de notre temps fut prise par la mise en place la représentation du graph dans l'espace 3D, en effet dût à l'inexistence d'une fonction 'draw-line' comme pour la représentation 2D, il fallut un temps conséquent avant de réussir à dessiner celui-ci correctement. Puis pour permettre une meilleure approche envers l'utilisateur de l'usage de la présentation en trois dimensions, nous avons mis en place un déplacement de type jeux FPS qui pourrait encore être grandement optimisé. Après quoi, nous sommes revenus sur la représentation en deux dimensions du graph afin d'y ajouter encore quelques fonctionnalités. Ce qui ne permis pas une grande avancée puisque, après choix, nous nous sommes permis de ne sélectionner que celles qui nous semblaient importantes. Par la suite, dû à la non-optimisation des calculs (→ l'animation s'affaiblit au-delà de 250 nœuds sur un ordinateur correct)

avons procédé à différentes méthodes afin de pouvoir régler ce problème, suppression des racines, utilisation de mémo-fonctions… Cependant par soucis de temps et de la non résolution de plusieurs autres soucis dans notre projet nous avons mis de côté ces changements pensant que l'optimisation qu'ils apportés n'était pas suffisante par rapport à l'apport de code qu'il nous fallait ajouter. Puis, dans les derniers moments de ce projet, nous avons portés une attention particulière à la division de notre code qui n'est clairement pas optimale, problème qui, ne fut malheureusement pas résolus (→ une baisse importante des performances, problème pour modifier les variables globales…). Pour finir, nous avons rédigé ce rapport ainsi qu'un guide d'utilisation de toutes nos fonctions afin de vous permettre une prise en main plus rapide de l'interface. Ainsi s'achève le travail que nous rendons, ce fut une expérience même si qu'elle que peu minime, nous permis d'effectuer un travail d'équipe sur un même code.

• Modules :

Vector2D : Contient les fonctions nécessaires à la création et à la manipulation de vecteur de dimensions deux : (ROMDHANE Mohamed-Amine)

- Création de vecteur à partir de 2 coordonnées :
  → (make-vect x y)
- Récupération des coordonnées x et y d'un vecteur :
  → (coord-x vect)
  → (coord-y vect)
- Somme de deux vecteurs :
  → (vect-sum vect1 vect2)
- Somme de plusieurs vecteurs :
  → (vect-sum* v …)
- Produit scalaire d'un vecteur par un réel :
  → (vect-scalar k vect)
- norme d'un vecteur :
  → (vect-norm vect)
- Vecteur unitaire d'un vecteur :
  → (vect-unit vect)
- Distance au carré entre deux vecteurs de dimensions 2 :
  → (distance^2 vect1 vect2)

Aucun module n'est prérequis, tout le module est exporté.

Vector3D → même fonctions que pour Vector3D mais pour un vecteur de dimensions trois, aucun module n'est prérequis, tout le module est exporté. (ZHANG Noé)

Graph : Contient les fonctions nécessaires à la création et à la manipulation d'un graph :

- Création d'un arbre vide : (ROMDHANE Mohamed-Amine)
  → (empty-graph)

- Est-ce-que le graph est vide ? :
  → (empty-graph? graph)
- Ajout d'un noeud, si déja existant ne fait rien :
  → (add-node! graph node-id)

- Ajout d'une arête entre deux nœud, si nœud non existants les ajoutent au graphe:
  → (add-edge! graph node-id1 node-id2)
- Obtenir une liste des nœuds du graph :
  → (get-nodes graph)
- Obtenir une liste des couples (nœud1 – nœud2) s'ils sont reliés par une arête :
  → (get-nodes graph)
- Obtenir les voisins d'un nœud du graph :
  → (get-neighbors graph node-id)
- Suppression d'un nœud, supprimer automatiquement les arêtes auxquelles il était rattaché :
  → (rm-node! graph node-id)
- Suppression d'une arête entre deux nœuds donnés :
  → (rm-edge! graph node-id1 node-id2)

Aucun module n'est prérequis, tout le module est exporté.

Positioning : Contient les fonctions nécessaires à l'association ou à la manipulation des coordonnées d'un nœud : (ROMDHANE Mohamed-Amine & ZHANG Noé)

- Création d'une fonction de placement vide :
  → (positioning)
- Renvoie les coordonnées d'un nœud dans une fonction de placement positioning:
  → (apply-positioning positioning node-id)
- Renvoie une fonction de placement à partir d'une liste d'association nœud/position :
  → (positioning-of-asso-list L)
- Création d'une fonction de placement partir d'une liste de point avec coordonnées aléatoire dans une dimension donnée :
  → (random-positioning-of-node-list w h L)
- Changement de la position d'un nœud dans une fonction de placement (ajout d'un vecteur donné) :
  → (positioning-move-node! positioning node-id vect)
- Affichage des coordonnées de nœud spécifiques dans notre fonction de placement :
  → (print-positioning id-list positioning)
- Changement de la position d'un nœud dans une fonction de placement (ajout d'un vecteur donné pour un vecteur de dimension 3) :
  → (positioning-move-node-3D! positioning node-id vect)

Les modules Vector2D et Vector3D sont prérequis, tout le module est exporté.

Relaxation : Contient l'objet soft (new-relaxator) quit permet de calculer les coordonnées de l'état suivant du graph en appliquant la force mécanique et la force électrique à chaque point. La fonction ne renvoie rien, elle effectue les mutations directement sur notre fonction de placement en fonction du graph choisi (→ pour vecteur2D + vecteur3D). (ROMDHANE Mohamed-Amine & ZHANG Noé)

Les modules Vector2D, Vector3D, Positioning et Graph sont prérequis, tout le module est exporté.

GraphGenerator : Contient les fonctions nécessaires à la création d'un graphe spécifique : (ROMDHANE Mohamed-Amine & ZHANG Noé)

- Création d'un graph chainé :
  → (chain-graph n)
- Création d'un graph cyclique :
  → (cyclic-graph n)
- Création d'un arbre complet :
  → (complete-tree-graph arity depth)
- Création d'un graph grille :
  → (grid-graph n m)
- Création d'un graph avec tous les nœuds reliés entre eux :
  → (clique-graph n)

Le module Graph est pré-requis , tout le module est exporté.

UserInterface : Il s'agit du fichier principal du projet, il regroupe les éléments nécessaires à la création de l'interface graphique et à l'affichage du graphe : (ROMDHANE Mohamed-Amine & ZHANG Noé)

- Les fonctions de dessins du graphe pour le mode 2D et 3D :

→ Dessiner nœud
→ Dessiner arête
→ Dessiner trou noir

- Les éléments de l'interface faisant office de passerelle entre l'utilisateur et les fonctions permettant d'interagir (Boutons, Eventful-Canvas-2D% sous classe de canvas%, Eventful-Canvas-3D% sous classe de pict3d-canvas%…)

- Les fonctions qui permettent d'interagir avec le graph :

→ Réinitialisation du canvas
→ Pause/Play du graph
→ Zoom -
→ Zoom +
→ Recentrage du graph
→ Passage du mode 2D → 3D
→ Passage du mode 3D → 2D
→ Se déplacer dans le Canvas 2D
→ Possibilité de mouvoir un nœud
→ Ajout d'un nœud au clic de la souris
→ Ajout d'une arête au clic de la souris
→ Suppression d'un nœud au clic de la souris
→ Suppression d'une arête au clic de la souris
→ Ajout d'un trou noir (→ position aléatoire)
→ Suppression aléatoire d'un trou noir
→ Changement de couleur pour les nœuds
→ Changement de couleur pour les arêtes
→ Déplacement dans l'espace aux touches du clavier (mode 3D) (Mode FPS)
(ZQSD + Espace + (L/R)Ctrl + Flèches [↑↓→←])
→ Rotation des directions dans l'espace à la souris (mode 3D) (Mode FPS)

Afin d'utiliser l'extension 3D de GraphX, il est nécessaire d'installer la librairie pict3d !
raco pkg install pict3d
(L'installation de cette librairie est plus ou moins longue (5-10min), si l'installation échoue :
'*raco pkg remove pict3d*' puis réinstaller)

Les modules Vector2D, Vector3D, Graph, Positioning, Relaxation, GraphGenerators sont prérequis, le module n'est pas exporté.

• Exécutables et Tests Unitaires :

Exécutables :

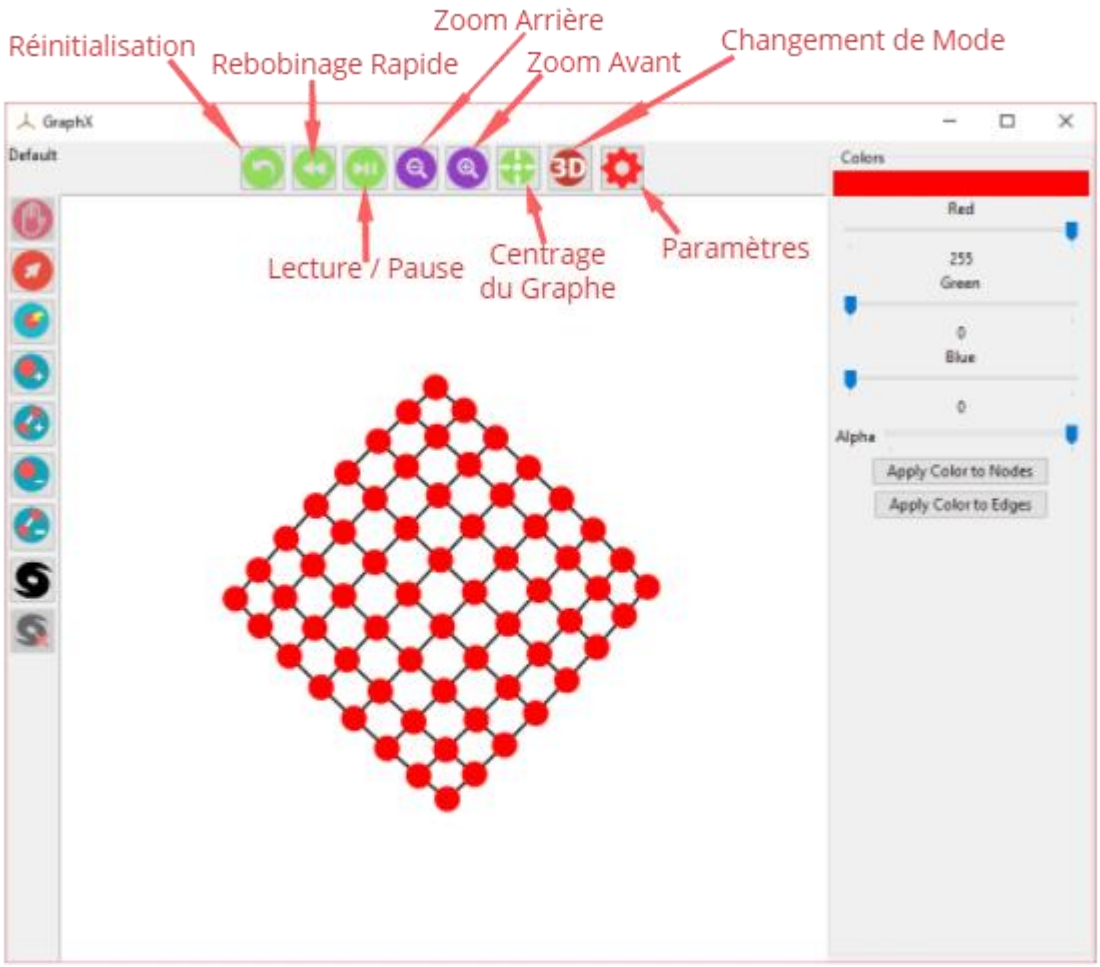
- SimulationRestrainte.rkt : Calcule puis affiche la fonction de placement du graphe (grid-graph 3 3) après N itérations de la fonction relaxation. L'erreur de 10% décrite dans l'énoncé du projet n'est jamais satisfaite. On a ajouté une méthode spéciale dans le module relaxation pour faciliter le changement de l'erreur toléré pour avoir N. En effet, pour le graph demandé, on a une erreur minimale de ~36%. Pour éviter les divisions par zéro pour canvas aussi petit qu'un 5x5, on a opté pour un petit changement de la fonction 'random-positioning-of-nodes-list' situé dans le module Positioning. La solution étant d'ajouter un réel dans]0, 1[pour ne pas avoir des nœuds qui se superposent et dont leurs distance est égale à 0. Ceci dit, leur distance est maintenant tellement proche de zéro que la force électrique entre ces deux nœuds est très importante, ce qui a pour effet d'augmenter le nombre d'itérations N avant d'atteindre un état d'équilibre total (qui n'est pas forcément unique).

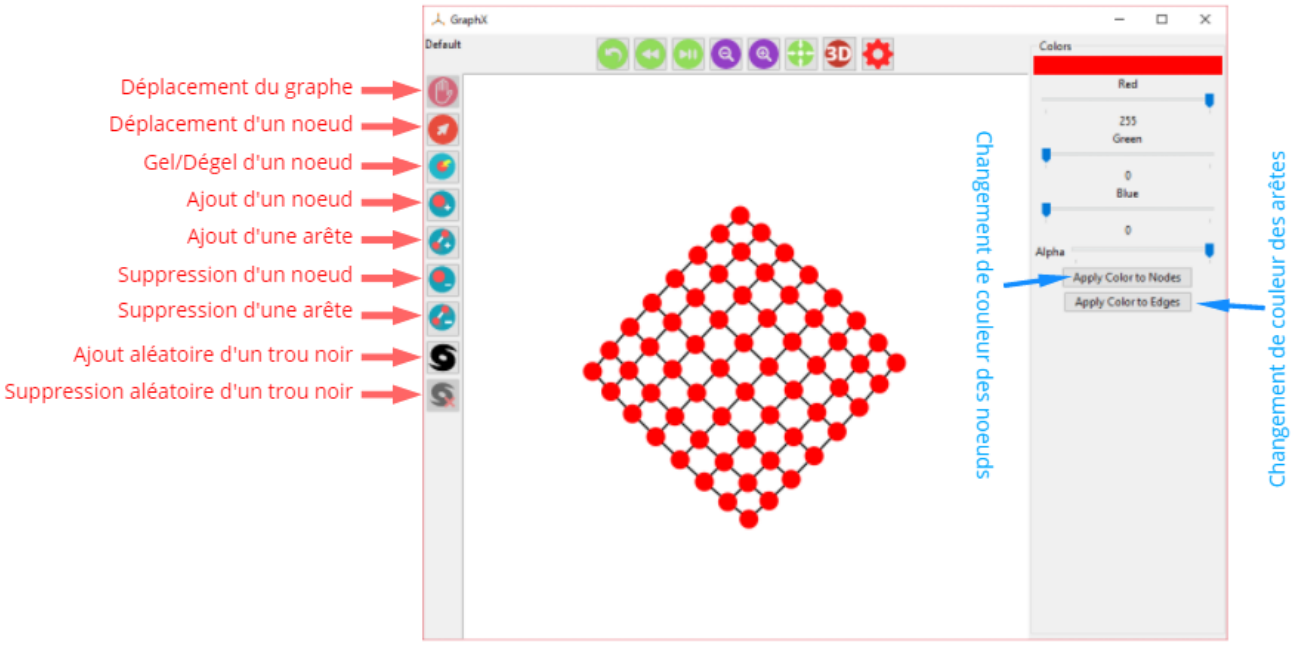
- UserInterface.rkt : Affiche un graphe dans un canevas et à mettre à jour le rendu au fur et à mesure de l'évolution de la fonction de placement. Comporte toutes les sous classes, les fonctions avancées, l'interface graphique et les variables globales. Le fichier fait plus 1000 lignes de code qu'on a pris bien soin de documenter.

Tests Unitaires :

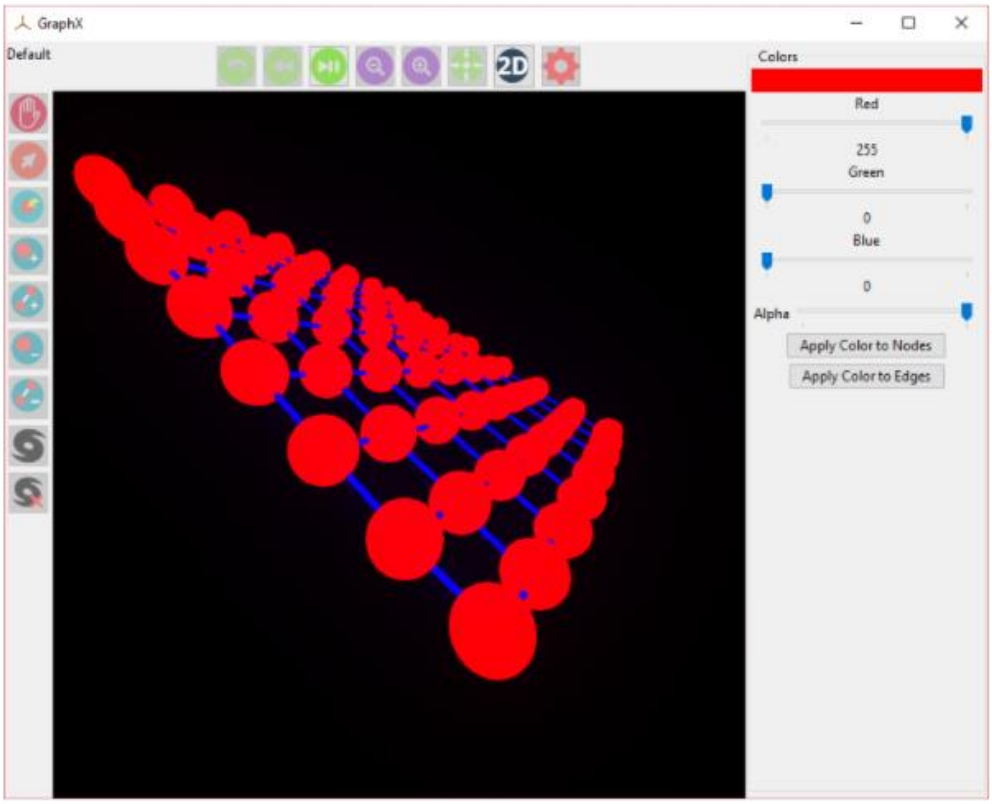
- Vector2D-Tests : Tests Unitaires des fonctions du module Vector2D.
- Vector3D-Tests : Tests Unitaires des fonctions du module Vector3D.
- GraphGenerators-Tests : Tests Unitaires des fonctions du module GraphGenerators.
- Graph-Tests : Tests Unitaires des fonctions du module Graph.
- Relaxation-Tests : Tests Unitaires des fonctions du module Relaxation.
- Positioning-Tests : Tests Unitaires des fonctions du module Positioning.

• Guide de l'Interface Graphique :





Clique Gauche Maintenu: Déplacement de la Caméra



Z
Q S D

↑
← ↓ →

• Bugs et problèmes persistants :

- Le rembobinage après une ou plusieurs suppressions de nœuds, venant du trou noir ou de l'outil 'Suppression des nœuds' ne fonctionne pas comme souhaitable. Les nœuds supprimés ne sont plus calculés dans le Relaxator.

- Le Zoom Arrière/Avant ne respecte pas à 100% un zoom qui est correct. En effet, on a opté pour la solution qui modifie directement le positioning et qui joue un peu sur les tailles des arêtes et des nœuds, parce que la solution qui agit directement sur le canvas via les méthodes get-initial-matrix et set-initial-matrix crée des bugs au niveau de la gestion des évènements de la souris, qui est la base pour tous les outils de la barre latérale gauche.

- Dans le mode 3D, puisqu'on n'utilise pas de Quaternions pour les rotations effectués à la Caméra, des fois, quand on regarde trop haut ou trop bas, on se trouve dans une situation de 'Gimbal lock'.

- Le (Relaxator 'final-state?) prend une valeur booléenne qui est totalement dépendante de l'erreur tolérée qui est une variable qu'on peut modifier à tout moment via (Relaxator 'set-tolerated-error e) avec e l'erreur en pourcent.

- Le module 'UserInterface.rkt' est trop chargé. On n'a pas pu le diviser en plusieurs sous modules, principalement dû au fait que toutes les interactions entre les objets de l'interface graphique et les variables globales sont fortement établies par des mutations. On a pris une approche impérative, non-fonctionnelle lors de la conception des solutions pour ce projet.