Ministère de l'Éducation Nationale
Université de Montpellier II
Place Eugène Bataillon
34095 Montpellier Cedex 5

Réalisation d'un effet Fisheye en Flex

TAWEB FMIN312 – Flex
Décembre 2012

Travail réalisé par :

Thibaut Marmin
Clément Sipieter

`https ://github.com/fishfinger/fisheyelayout`

1 Contraintes

Réalisation d'un effet *fisheye* en flex. L'idée est d'appliquer l'effet sur les conteneur présents dans le conteneur appliquant le layout *fisheye*, sous la forme d'une grille dont les éléments grandisse en fonction de la proximité avec la souris.

Dans l'idéal, l'effet produit doit minimiser l'espace entre les conteneurs.

2 Analyse & implémentation

Nous avons fait le choix de créer un layout *fisheye* assurant les traitements nécessaire à la réalisation de l'effet. Pour cela, nous avons utilisé le namespace *spark* permettant de dissocier le container du layout. Il sera ainsi possible d'associer le layout personnalisé à n'importe quel container spark. Le layout appliquera l'effet sur tous les containers dont le container parent est celui sur lequel est appliqué le layout *fisheye*.

Pour la réalisation de ce layout, nous avons eu trois approches qui seront détaillées ci-après.

2.1 Approche globale

Cette approche suppose qu'il est possible, en connaissant la position de la souris et la position d'un bloc, de le placer et le dimensionner exactement sur la grille. Les caractéristiques du bloc ne dépendent donc pas du placement des autres blocs voisins.

Le layout peut être paramétré de la manière suivante :

maxSize Taille maximum de l'élément dans la grille,

minSize Taille minimum de l'élément dans la grille,

spread Propagation de l'agrandissement autour de la souris,

Ainsi le layout va initialiser les éléments sur une grille de la manière suivante :

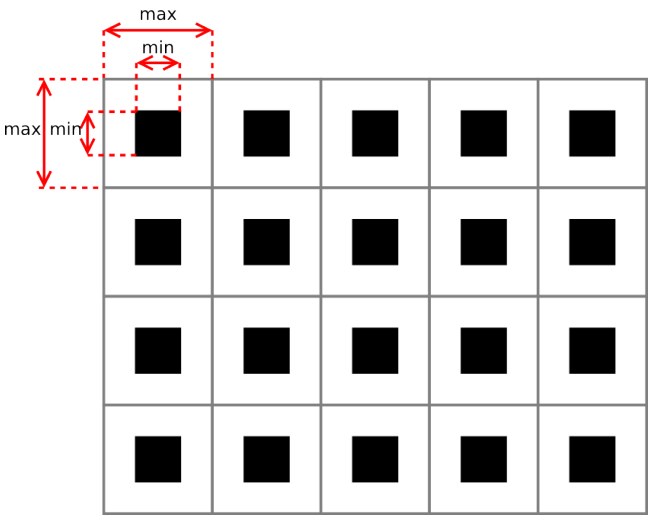

FIG. 1 – Initialisation des éléments avec l'approche globale.

Un *eventlistener* capture les mouvements de la souris dans le but de mettre à jour chaque *container* de la grille.

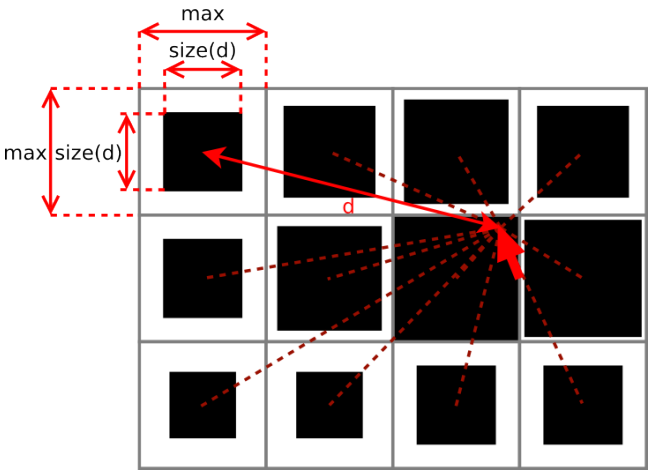

FIG. 2 – Mise à jours des éléments avec l'approche globale.

La taille des éléments sont mis à jours en fonction de la distance entre le centre du block et le pointeur de la souris. Dans tous les cas les éléments restent alignés dans la grille par le fait que les centres des objets ne sont pas déplacés.

Après implémentation et test avec des containers colorés en noir, on obtient ceci :

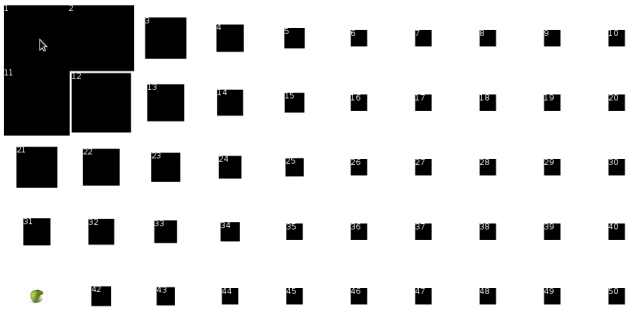

Fig. 3 – Premier aperçu.

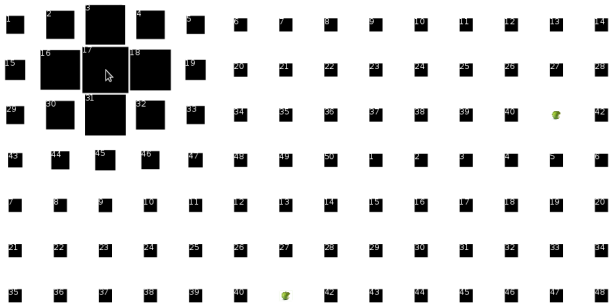

Fig. 4 – Second aperçu.

L'avantage de ce type d'approche est d'arriver rapidement à un résultat fluide et satisfaisant. Le principal problème réside dans l'espacement entre les éléments de la grille. Un solution[1] pourrait consister en la mise en place d'une attraction des éléments vers le pointeur de la souris : plus les éléments sont éloignés, plus ils sont attirés vers la souris. On obtiendrait quelque chose comme suit :

[1]Solution non implémentée.

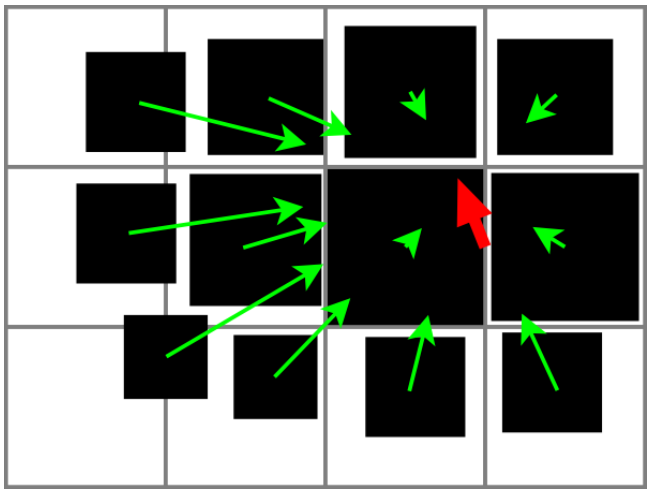

FIG. 5 – Évolution permettant de réduire l'espace entre les éléments de la grille.

2.2 Approche propagée

Cette approche est différente. Les dimensions et la position de chaque élément dépend des autre, et sont affectés au travers d'un parcours séquentiel.

Ici les éléments sont initialisé par une valeur moyenne ainsi qu'une marge entre les éléments.

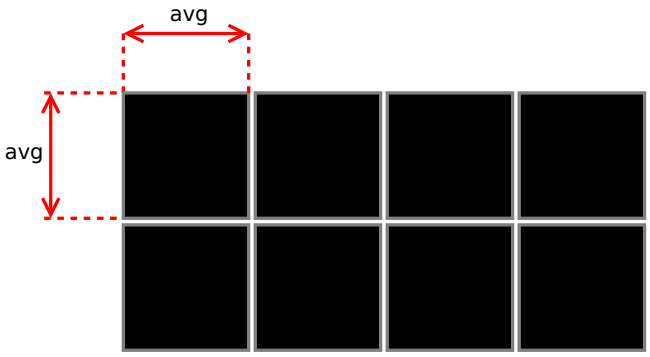

FIG. 6 – Initialisation par la méthode propagée.

Lors d'un mouvement de souris, le parcours débute par l'élément le plus proche de la souris, affecte les positions et tailles des éléments sur la ligne, puis le traitement est effectué de manière récursive pour chaque autre ligne.

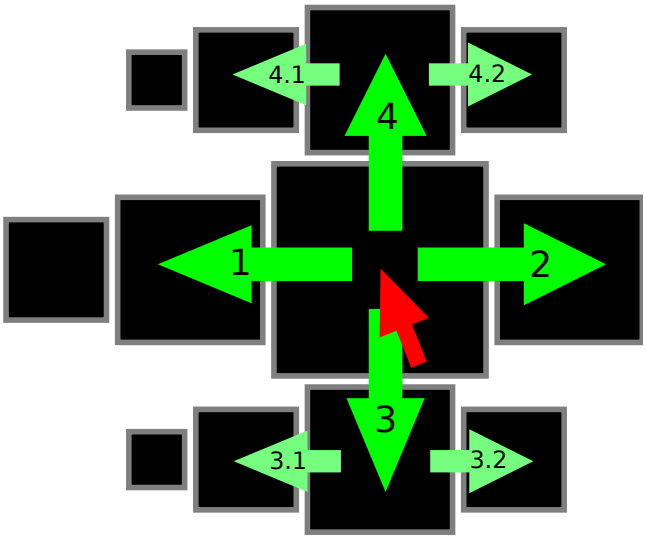

FIG. 7 – Évolution permettant de réduire l'espace entre les éléments de la grille.

Ce rendu a été réalisé mais n'est pas exploitable pour deux raisons :

- pas de fluidité lors du passage de la souris entre deux éléments,
- disposition peu ergonomique.

Dans le but d'améliorer le second point, la disposition des éléments des ligne pourrait être améliorer[2]. La première évolution propose de contraindre la distribution des éléments de chaque ligne en se basant sur la grille générée au fur et à mesure du parcours. La seconde évolution fonctionne de la même manière, mais déplace les éléments dans le coin le plus proche de l'élément sélectionné.

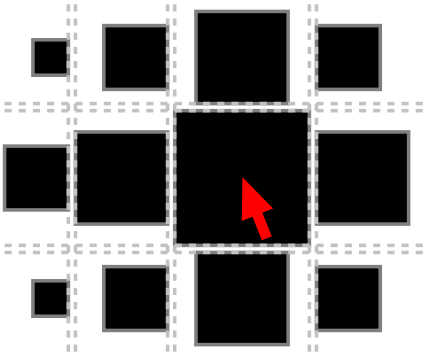

FIG. 8 – Première évolution permettant un meilleur affichage de la grille.

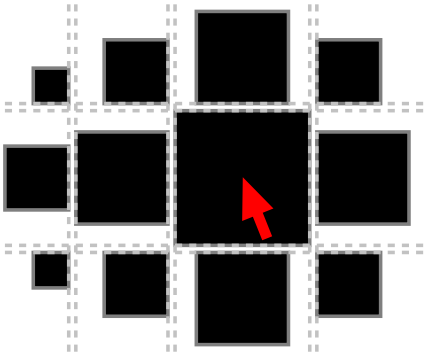

FIG. 9 – Seconde évolution permettant un meilleur affichage de la grille.

[2]Amélioration non implémentée.

2.3 Approche avancée

La dernière approche est une approche avancée axée sur l'aspect mathématique du problème, dans le but de réaliser un « vrai » effet *fisheye*. L'idée est de considérer l'ensemble des éléments comme un tableau d'éléments puis de jouer sur la largeur des colonnes et la hauteur des lignes.

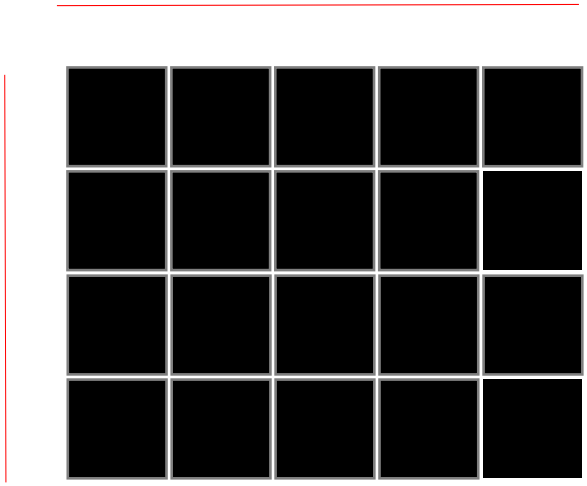

FIG. 10 – Grille d'éléments sans présence de la souris

La figure 10 ci-dessus représente l'état du tableau en l'absence de la souris. Dans ce cas, la largeur et la hauteur des cellules sont représentés par une fonction constante. La figure 11 montre comment la largeur et la hauteur des cellules sont modifiées par la présence de la souris. La nouvelle taille des cellules est calculée grâce à deux fonctions gaussiennes indépendantes, une pour la largeur et une autre pour la hauteur.

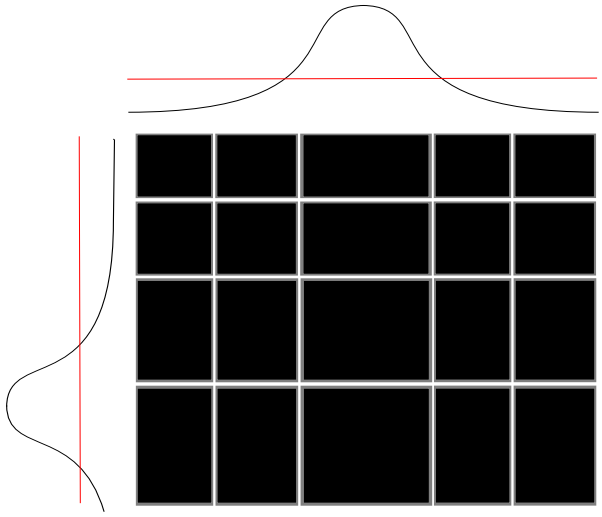

FIG. 11 – Grille d'éléments modifié par la présence de la souris

Ces fonctions sont définies comme suit :

$$taille\_par\_defaut + \frac{facteur\_grossissement}{(dist/attenuation)^2 + 0.5} - facteur\_grossissement \quad (1)$$

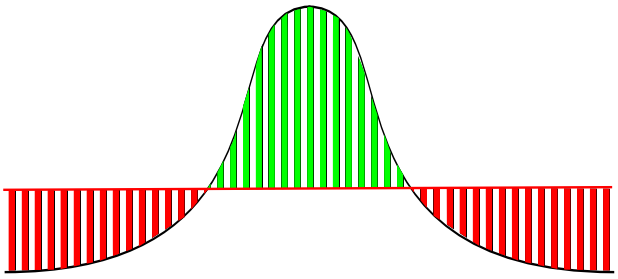

FIG. 12 – Courbe de déformation

La surface verte représente le gain de taille par rapport à la taille par défaut, la surface rouge représente la perte de taille. L'enjeu lors du parcours de la souris est de faire en sorte que ces deux surface soient égales.

Ce problème est mis en évidence dans le pire des cas lorsque la souris est au maximum contre un bord. Comme il est possible de le voir sur la figure 13, dans ce cas la surface de gain est bien plus élevée que la surface de perte. Dans le but d'atteindre une répartition parfaite entre ces deux surfaces, la différence entre la surface rouge et la surface verte est calculée, puis redistribuée sur tous

les éléments de la ligne et/ou de la colonne. La figure 14 représente le résultat obtenu.

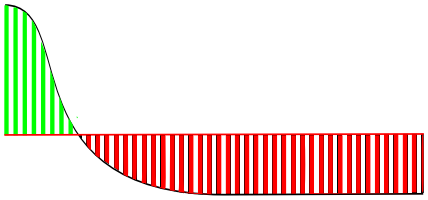

FIG. 13 – Courbe de déformation

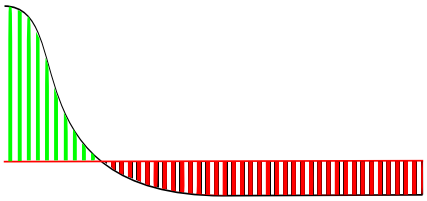

FIG. 14 – Courbe de déformation

2.3.1 Implémentation JavaScript

Un premier prototype a été réalisé en javascript / HTML5, avec deux *sliders* permettant la modification à la volé du facteur de grossissement et de l'atténuation. Une *checkbox* permet également la représentation des éléments sous la forme de carrés.

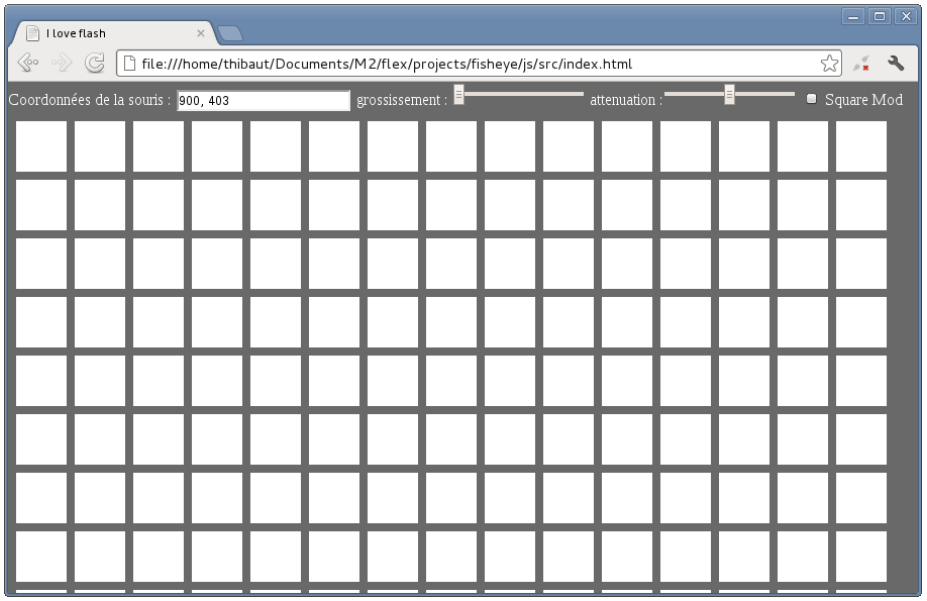

FIG. 15 – Aperçu de l'effet fisheye JS sans facteur de grossissement (grille de base).

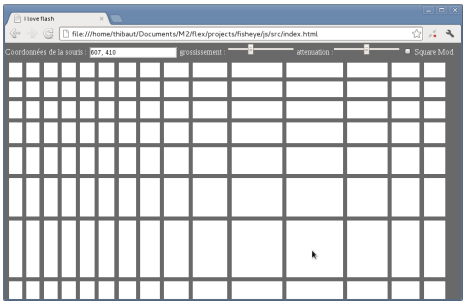

FIG. 16 – Aperçu de l'effet fisheye JS avec facteur de grossissement (mode carré désactivé).

FIG. 17 – Aperçu de l'effet fisheye JS avec facteur de grossissement (mode carré activé).

2.3.2 Implémentation Flex

Un prototype reposant sur les mêmes algorithmes à été développé en flex. Il implémente les mêmes fonctionnalités que son homologues JavaScript, et est basé sur l'implémentation d'un *Layout Spark* personnalisé, comme les version vues dans les sections 2.1 et 2.2.

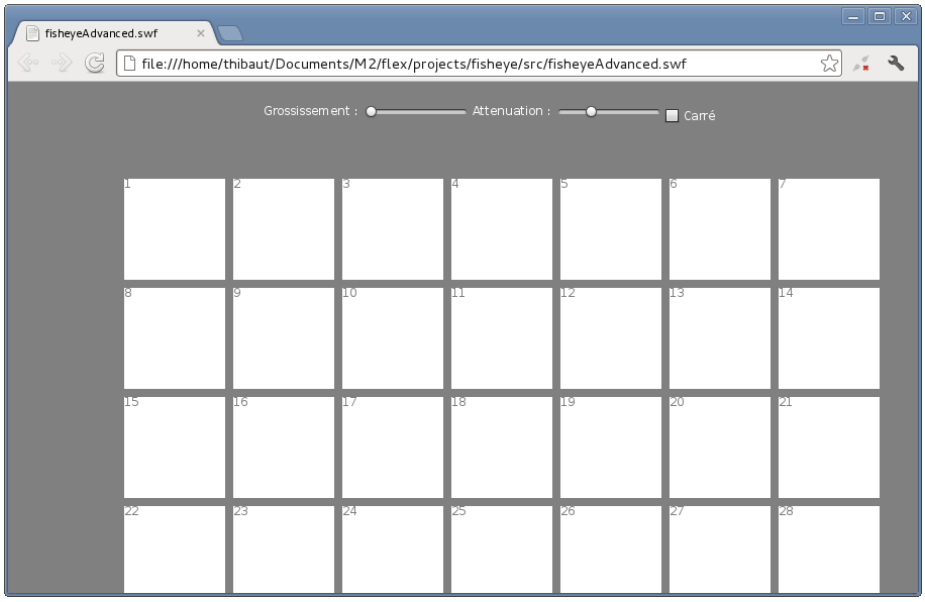

FIG. 18 – Aperçu de l'effet fisheye Flex sans facteur de grossissement (grille de base).

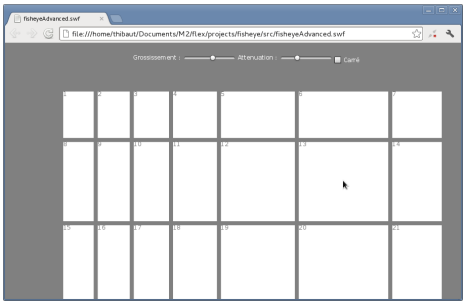

FIG. 19 – Aperçu de l'effet fisheye Flex avec facteur de grossissement (mode carré désactivé).

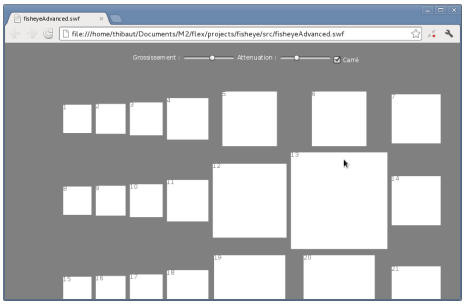

FIG. 20 – Aperçu de l'effet fisheye Flex avec facteur de grossissement (mode carré activé).

3 Environnement de travail

Dans le but de réellement découvrir flex, nous avons fait le choix de ne pas utiliser Flash Builder. Le développement a donc été effectué sous GNU/Linux à l'aide d'un éditeur de texte avec complétion syntaxique pour *mxml* et les classes *actionscript*. La compilation a été exécutée en ligne de commande avec les compilateur *mxmlc*.

Aucune version d'Air n'étant disponible sous GNU/Linux, nous avons testé nos applications avec le FlashPlugin, en lançant les fichiers *swf* avec un navigateur.

4 README

Les sources de ce projet sont disponibles sur notre dépôt GIT (Github) à l'adresse suivante :

`https ://github.com/fishfinger/fisheyelayout`

Fichiers compilés disponibles dans le dossier `swf` à la racine.

Sources disponibles dans le dossier `src`) à la racine. Les différents *Layout* sont disponibles dans le *package* `custom.layouts`.

Documents disponibles dans le dossier `docs`. Y sont présents la présentation et le présent rapport.