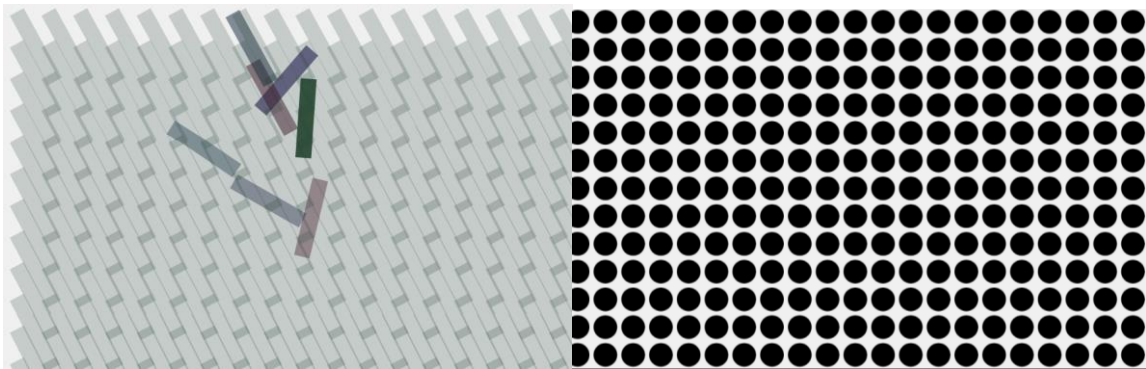


Construire un moteur d'animation 2D en JS

Le projet AutomatosJS



Version 1.1

Construire un moteur d'animation 2D en JS

Table des matières

Notes de l'auteur	3
Introduction	4
Canvas, une API géniale, mais...	6
Cours sur la construction d'un moteur d'animation 2D	8
Bon alors, t'en es où mec ?	11
Premiers tests	12
Conclusion provisoire	14
Bibliographie	15
Changelog	16

Notes de l'auteur

Je m'appelle Grégory Jarrige.

Je suis développeur professionnel depuis 1991. Après avoir longtemps travaillé sur le développement d'applications de gestion, sur gros systèmes, avec des langages et technos propriétaires, j'ai fait le pari de me former aux technos et langage open source vers 2006. J'ai commencé à développer des applications webs professionnelles à partir de 2007, avant d'en faire mon activité principale à partir de 2010. L'arrivée du HTML5 dans la même période a été pour moi une véritable bénédiction, et surtout un formidable terrain d'expérimentation (avec des API comme Canvas, WebAudio, etc...).

Aujourd'hui, je développe des interfaces utilisateurs en Javascript sur des applications de type intranet, des tableaux de bord basés sur des solutions de dataviz comme D3.js, et des serveurs d'API basés sur NodeJS.

Si j'ai longtemps dû faire le grand écart entre des technologies professionnelles et des technos plus ludiques, ce n'est plus le cas aujourd'hui : car j'utilise dans mon travail le même outil de base que dans mes projets de coding créatif, le langage Javascript (dont je suis fan). En clair, je suis un développeur heureux.

Ce document n'est pas exhaustif, il est forcément partiel et certainement aussi un peu partial. C'est un "work in progress", réalisé un peu dans l'urgence, pour une présentation faite au meetup [CreativeCodeParis](#), le 20 décembre 2018.

Le présent document est publié sous Licence Creative Commons n° 6. Il est disponible en téléchargement libre sur mon compte Github, dans le dépôt suivant (cf. doc "moteur2D_and_JS.pdf") :

<https://github.com/gregja/automatosjs>

Introduction

Dans le présent document, je souhaite expliquer les raisons qui m'ont amené à vouloir créer un petit moteur d'animation 2D en Javascript, et je souhaite présenter un certain nombre d'auteurs qui m'ont inspiré et guidé dans mon développement.

Il faut souligner que, si le sujet me passionne, je ne suis en aucune manière un expert du développement d'animation. Mon expertise se situe plutôt dans le développement d'applications d'entreprise, aussi j'ai véritablement tout à apprendre dans le domaine, et ça tombe bien, j'adore apprendre des choses nouvelles.

Je me suis efforcé de construire ce petit moteur en utilisant le meilleur des techniques disponible dans les versions ES5 et surtout ES6 du langage Javascript. Si vous n'êtes pas familiarisé avec ce que sont ES5 et ES6, lisez ce qui suit, sinon vous pouvez passer directement au prochain chapitre.

Le langage Javascript n'existe pas en tant que tel, c'est un nom d'usage, adopté par le plus grand nombre, pour désigner l'implémentation dans chaque navigateur de la norme ECMAScript 262 (souvent abrégée en ECMA-262, bon il est vrai qu'avec un nom pareil...).

Cette norme est régulièrement mise à jour et est publiée sur ce site :

<http://www.ecma-international.org/publications/standards/Ecma-262.htm>

Chaque navigateur est donc équipé d'un interpréteur de code Javascript, et il faut souligner que les éditeurs de navigateurs font aujourd'hui de gros efforts pour coller au plus près de la norme, et pour implémenter ses évolutions rapidement. Et c'est tant mieux pour les développeurs.

Si la norme ECMA-262 en est aujourd'hui à la version 9, publiée en juin 2018, il est intéressant de s'arrêter sur deux périodes clés de son évolution :

- L'édition 5 publiée en 2009, généralement appelée ECMAScript 5, constituait une avancée majeure. Elle apportait beaucoup de nouveautés qui auparavant étaient dispersées dans divers frameworks Javascript. Elle est arrivée en même temps que la norme HTML5 qui de son côté apportait un large panel d'API, toutes programmables en Javascript. Bref, l'année 2009 est véritablement une année charnière dans le monde du web. J'aime à penser qu'une petite révolution pacifique et bienveillante s'est produite, en 2009, dans nos navigateurs.
- L'édition 6 publiée en 2015, généralement appelée ECMAScript 6, ou quelquefois ECMAScript 2015 (ce qui a parfois entraîné des confusions dans les esprits). Plus discrète, mais tout aussi importante, cette édition a consolidé les bases posées par l'édition précédente, en apportant tout un tas de nouveautés très pratiques, notamment pour la manipulation de données. Nous en découvrirons certaines au fil de l'eau.

Ce qui est intéressant à souligner ici, c'est que nous sommes à la fin de l'année 2018 (au moment où j'écris ces lignes), que la norme ECMAScript 6 est là depuis 3 ans, et qu'elle est pleinement supportée par les navigateurs actuels (sous réserve qu'ils soient à jour). Si vous êtes un ou une « creative coder », vous pouvez donc vous lâcher. Si vous êtes un développeur professionnel, obligé de surveiller la compatibilité avec des navigateurs plus anciens ne supportant pas ES6, vous pouvez utiliser des solutions de type « transpilage » comme le projet Babel (qui permet une conversion du

code ES6 vers ES5), donc vous pouvez vous lâcher aussi sur ES6, dans une certaine mesure. Et puis si vous développez sous NodeJS, vous savez sans doute que l'interpréteur Javascript de NodeJS est à la pointe en ce qui concerne le support de la norme ECMAScript 262, donc tout ce que nous allons évoquer ici devrait vous intéresser (même si nous allons plus particulièrement nous intéresser à l'utilisation de Javascript côté navigateur).

A qui est destiné ce support ?

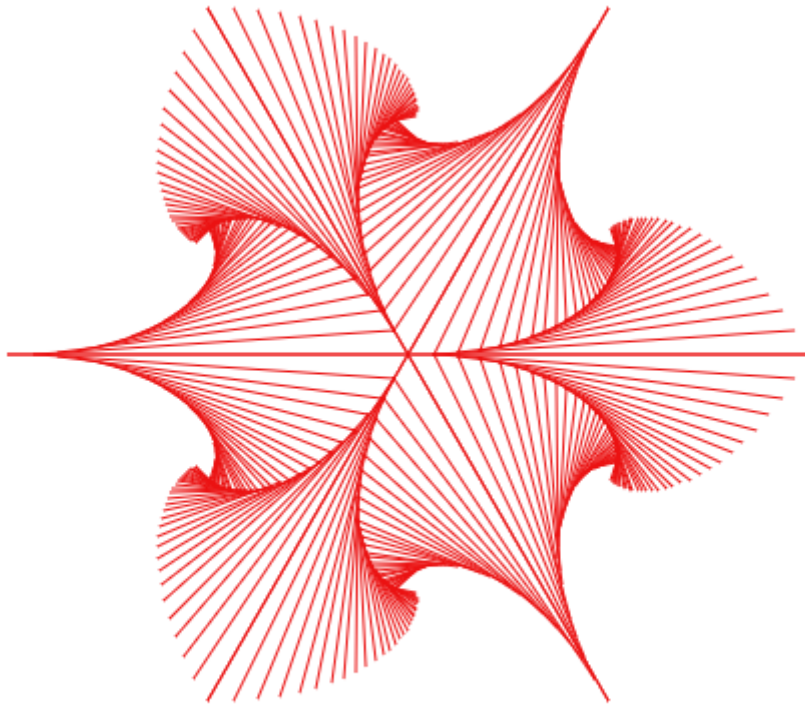
A un large public, à commencer par les membres du meetup [CreativeCodeParis](#), qui sont de plus en plus nombreux à se lancer dans des projets créatifs exploitant l'API Canvas du HTML5.

Attention, ce document n'est pas destiné à de grands débutants en programmation (en tout cas dans pas dans cette première version). Aux débutants, je recommanderais plutôt le document PDF qui se trouve dans ce dépôt :

<https://github.com/gregja/p5Corner>

Canvas, une API géniale, mais...

Quand j'ai commencé à jouer avec l'API Canvas de la norme HTML5 (c'était aux alentours de 2010, ou 2011), mon premier réflexe a été de sortir mes vieux bouquins de programmation des années 80 et 90, et de m'essayer à la programmation de graphiques sympas, comme celui-ci :

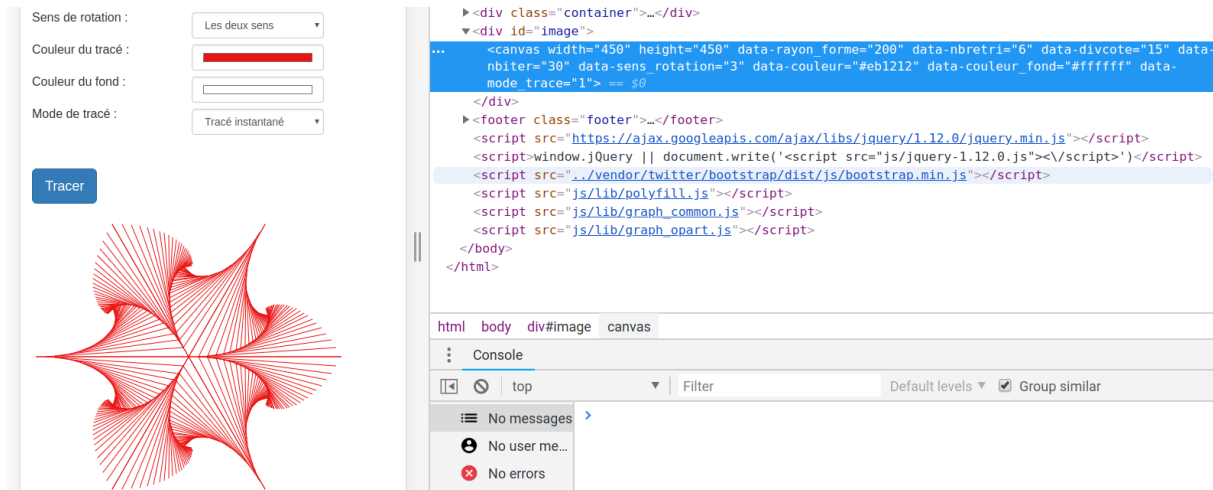


Pour info, ce graphe est généré à partir d'un algorithme pris dans un livre de Robert Dony qui s'intitule "graphisme dans le plan et dans l'espace avec Turbo Pascal" (éditions Masson, 1991).

C'était sympa de me replonger là dedans, j'étais ravi, alors j'ai poursuivi par l'étude des techniques d'animation disponibles avec l'API Canvas. Cela fonctionnait plutôt bien, mais très vite je me suis heurté à un problème.

Pour dessiner des formes dans un canvas, on utilise des fonctions Javascript qui s'apparentent à du graphisme vectoriel, on obtient d'ailleurs un excellent rendu (cf. ci-dessus), mais une fois la forme dessinée dans le canvas, elle n'existe plus en tant qu'objet vectoriel. On obtient juste un objet de type "bitmap" (une matrice de pixels). C'est tout.

Quand on utilise l'inspecteur d'élément du navigateur, on voit très bien que le graphique à gauche est représenté à droite par une unique balise HTML unique de type "canvas". Cette balise ne contient que quelques attributs comme la taille ("width" et "height") et d'autres plus spécifiques que j'ai ajoutés pour stocker les paramètres initiaux ayant servi à générer la forme (tous ceux qui commencent par "data"). Mais on ne trouve aucune trace des instructions ayant généré le dessin à l'intérieur du "canvas" :



Or, j'aimerais bien lui tirer les oreilles à cette forme, saisir à la souris les fils, les tordre, les étirer, ça serait fun non ? Oui je sais, je suis un geek, et les geeks ont de drôles d'amusement... c'est comme ça, on ne se refait pas ;).

Bon, vous pourriez m'opposer qu'il y a une autre API, qui s'appelle SVG. Elle est dédiée au graphisme vectoriel justement, et elle est peut-être un meilleur candidat pour ce que je viens de décrire (vous savez... tirer les fils et tout ça...). Oui mais moi je veux faire la même chose avec Canvas... je vous l'ai dit je suis un geek, et les geeks sont généralement têtus.

La solution à mon problème m'est apparue quelques temps plus tard, avec la découverte d'un cours proposé sur Paris par un développeur du nom de Nicolas Legrand.

Cours sur la construction d'un moteur d'animation 2D

Dans le courant de l'année 2013, je découvre un cours sur Paris proposé par Nicolas Legrand. Je m'inscris à une session.

Lors de cette session, je découvre que Nicolas a publié une partie importante de son support de cours sur developpez.com, où il est en téléchargement libre (je vous invite à le récupérer) :

<https://javascript.developpez.com/tutoriels/html5/creer-moteur-affichage-2d-html5/>

Le cours de Nicolas est très bien construit, il donne les bases pour bien démarrer avec l'API Canvas, mais surtout il donne les éléments permettant de dépasser les limites de cette API.

Voici la table des matières du support de Nicolas, pour vous faire une idée :

I. Rapide Historique

I-A. JavaScript & ECMAScript

I-B. HTML5 et Canvas

II. Dessiner avec canvas

II-A. Fichier de base

II-B. Créer un canvas

II-C. Dessiner des primitives (lignes, cercles, carrés...)

II-C-1. Dessiner une ligne

II-C-2. Dessiner un cercle

II-D. Alpha, Scale, Rotation et Translation

II-D-1. L'alpha

II-D-2. La translation

II-D-3. Le scale

II-D-4. La rotation

II-D-5. Cumul des transformations, sauvegarde et restauration du contexte

III. Dessiner une image

III-A. Charger une image (ou texture)

III-B. Dessiner une texture.

III-C. Méthodes de dessin avancées

III-D. Dessiner à travers un masque

IV. Structure de base du moteur

IV-A. La POO en JavaScript, le prototypage

IV-B. Héritage et ordre d'inclusion

IV-C. Les Namespaces

V. Gestion des médias (ou assets)

V-A. Introduction aux spritesheets

V-B. Gestion des Assets : AssetsLoader et AssetsManager

V-B-1. La classe AssetsLoader

V-B-2. La classe Texture

V-C. Regrouper toutes les textures, la classe TextureAtlas

VI. Les bases de l'affichage

VI-A. Structure arborescente et DisplayList

VI-B. Le point de départ : La classe DisplayObject

VI-C. Enfin des textures : La classe Bitmap

VI-D. Objets imbriqués : La classe DisplayObjectContainer

VI-E. Racine de la DisplayList : classe Stage

VII. Manipuler les objets : transformations et calculs matriciels.

VII-A. Le problème des transformations imbriquées.

VII-B. Les matrices, pour quoi faire ?

VII-C. Comment utiliser les matrices ?

VII-D. Implémentation des matrices dans le moteur.

VIII. Modèle événementiel et design pattern

VIII-A. Pourquoi utiliser des événements ?

VIII-B. Comment gérer un modèle événementiel, le design pattern observer

VIII-C. Implémentation sur les objets d'affichage, la classe EventDispatcher et la classe Event

IX. Collisions et événements utilisateurs

IX-A. Comment détecter les événements souris ?

IX-B. Théorie des collisions, les formules

IX-C. Implémentation d'une méthode hitTest sur un DisplayObject

X. Les animations

X-A. Créer une classe MovieClip permettant de jouer une animation.

XI. Les filtres

Le cours de Nicolas Legrand m'a beaucoup apporté. Il explique comment construire un moteur similaire à Tomahawk, un moteur de niveau professionnel qu'il a construit sur le même modèle.

Code source de Tomahawk :

<https://github.com/thetiny spark/tomahawk/>

Site web de l'agence de Nicolas Legrand :

<http://the-tiny-spark.com/>

A l'issue de la formation, je n'ai pas trouvé le temps nécessaire pour approfondir tout ce que Nicolas m'avait appris (trop de boulot sur le moment, sur d'autres sujets trop éloignés de celui-ci).

Mais par la suite, j'ai découvert le projet P5.js (le petit frère du projet Processing), et j'ai décidé d'en explorer les possibilités. Mais après la phase de découverte de P5, je me suis rendu compte que je ne parvenais pas à aller beaucoup plus loin, avec P5, qu'avec l'API Canvas. Car P5 est lui aussi dépourvu de ce moteur d'animation que Nicolas m'avait appris à construire.

A noter qu'il existe dans le projet Processing, un objet PShape, qui pourrait combler partiellement les lacunes de P5, mais aux dernières nouvelles, c'est l'un des rares éléments de Processing qui n'a pas été reporté dans P5 :

<https://www.processing.org/reference/PShape.html>

Bon alors, t'en es où mec ?

De l'eau a coulé sous les ponts, la norme ES6 (de 2015) est arrivée avec plein de nouveautés sympas (comme les classes et l'héritage), dont j'ai très vite compris qu'elles allaient me permettre de revenir au sujet étudié avec Nicolas. Et puis j'ai réussi à me bloquer un peu de temps pour me replonger dans le sujet.

Il faut savoir que le moteur de Nicolas est fondé sur la norme ES5 (version 2009). Or avec ES5, il fallait jongler (truander ?) pas mal pour contourner certaines limitations, notamment l'absence d'un mécanisme d'héritage natif. Nicolas proposait dans son cours une solution de contournement assez élégante, mais quand même très spécifique. Avec ES6, certaines contraintes sont levées, et cela devient plus simple de construire certains des designs patterns nécessaires au moteur d'animation proposé par Nicolas, en s'appuyant sur les classes et le mécanisme d'héritage, disponibles désormais de manière native dans Javascript.

Donc après avoir longtemps tergiversé, en cette fin d'année 2018, j'ai décidé de m'attaquer au problème, avec un double objectif :

- reconstruire un moteur sur le modèle de celui de Nicolas, mais en profitant du meilleur de la norme ES6 (classes et héritage inclus)
- essayer de construire un moteur qui permette de pallier la principale carence de P5 (son absence de moteur d'animation justement), et peut être construire un moteur qui puisse s'utiliser à la fois avec P5 et indépendamment de P5 (là c'est ambitieux, mais ça se tente...).

Et je suis heureux de pouvoir vous dire que mon moteur existe, il est loin d'être terminé, il y manque par exemple une gestion de collision plus avancée (pour des formes autres que le parallélogramme), mais ça avance doucement. Et il manque surtout une vraie documentation, alors je vais essayer de pallier partiellement le manque avec ce document.

Le code source se trouve sur ce dépôt Github :

<https://github.com/gregja/automatosjs>

Vous pouvez le télécharger et vous amuser avec. Vous pouvez aussi me proposer des améliorations, toute remarque constructive est bienvenue.

Et j'espère que ceux des lecteurs qui n'ont pu être présents lors du meetup CreativeCodeParis du 20 décembre 2018 voudront bien me pardonner, mais il est temps de passer à la démo "live".

Premiers tests

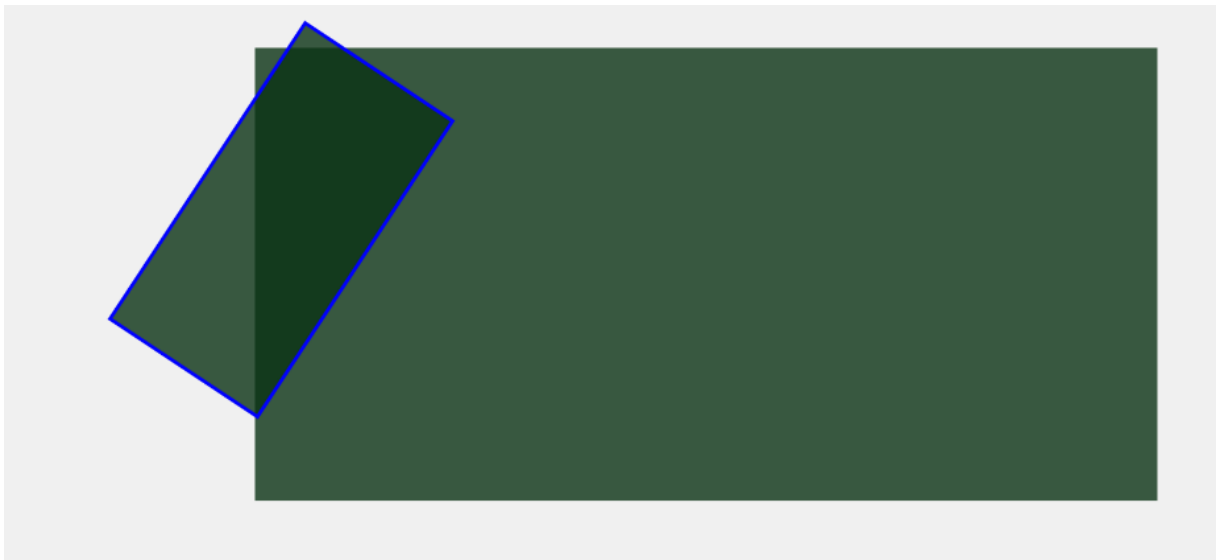
J'ai créé une première batterie de tests pour vérifier le bon fonctionnement du moteur. Vous pourrez les lancer à partir de la page index.html du dépôt, après l'avoir installé sur votre machine.

Le menu actuel affiché par le fichier index.html se présente de la façon suivante (dans la version du 22 décembre 2018) :

[Source code of the project AutomatosJS on Github](#)

- [sketch 1 - premiers objets mobiles](#)
- [sketch 2 - détection du click sur les objets en mouvement \(check console via F12\)](#)
- [sketch 3 - l'armée des clones \(détection du clic sur objets également\)](#)
- [sketch 4 - famille de rectangles \(liste chaînée\)](#)
- [sketch 5 - détection de collision avec une barre verticale](#)
- [sketch 6 - combinaison des sketches 3 et 4 \(passez "A" pour effet de rotation supplémentaire\)](#)
- [sketch 7 - drag and drop de 2 rectangles, zoom avec molette de la souris](#)
- [sketch 8 - effet d'expansion avec des billes](#)
- [sketch 9 - premier test d'une fonction "Easing"](#)

Chaque test m'a permis de valider différents aspects du moteur, mais à mon sens l'un des tests les plus importants, c'est le sketch n°7, qui gère un effet de type "drag & drop" (glisser déposer) sur 2 rectangles.



L'un des 2 rectangles est en rotation constante, mais tous deux sont sélectionnables à la souris, et déplaçables. Un liseré rouge entoure un rectangle en cours de déplacement, un liseré bleu indique quel est le dernier rectangle sélectionné après relâchement du bouton de la souris. L'échelle du dernier élément sélectionné peut être modifiée avec la molette de la souris (effet de zoom avant/arrière).

Le moteur permet donc à ce stade d'identifier précisément où se situe un élément, de savoir si j'ai cliqué dessus alors même qu'il est en rotation et/ou que son échelle a été augmentée ou diminuée.

Cette étape est importante car cela signifie que je peux interagir avec des formes en mouvement, ce qui était mon but principal en démarrant la construction du moteur.

Pendant la présentation, au meetup CreativeCodeParis du 20 décembre, Stan m’a demandé s’il n’aurait pas été possible d’effectuer la même chose avec les fonctions standards de P5, selon une approche plus classique, que l’on pourrait qualifier de procédurale. La réponse est “oui, c’est possible”, notamment avec 2 objets comme dans l’exemple ci-dessus, mais cela devient vite très compliqué dès lors que le nombre d’objets augmente. Compliqué du fait du nombre, mais surtout du fait que chaque élément graphique vit sa vie à l’intérieur du canvas selon ses propres paramètres de translation, rotation et échelle (un point très bien expliqué dans le support de cours de Nicolas Legrand, dont on n’a pas eu le temps de discuter durant la présentation).

En aparté de la présentation, Patrick m’a parlé d’une solution alternative, qui permettrait de simplifier la détection du clic de souris par rapport aux éléments présents sur le canvas. La solution consiste à créer un canvas caché (appelons-le “ghost canvas”), dans lequel sont dessinés les mêmes éléments que dans le canvas visible, mais où chaque élément a une couleur distincte, référencée dans une base interne. Ainsi, en fonction de la couleur du pixel sélectionné, on sait identifier précisément quel élément a été “cliqué”. Personnellement, c’est une astuce que je trouve astucieuse et élégante, et j’ai bien envie de l’implémenter dans une prochaine version du moteur. En plus ce pourrait être amusant de pouvoir afficher alternativement le canvas “maître” ou son “ghost”, ou les deux en parallèle... bref, j’aime beaucoup l’idée.

Conclusion provisoire

Je dois préciser que je n'ai pas suivi scrupuleusement les indications de Nicolas pour la construction du moteur. D'abord parce que j'avais la contrainte de l'intégrer dans P5, mais aussi parce que j'avais envie d'expérimenter quelques idées trouvées dans certains livres consacrés à Actionscript (cf. chapitre bibliographie). J'ai notamment pris des libertés dans la structure du pattern "display object" que j'ai appelé "Sprite" et que j'ai orienté davantage vers la manipulation de formes de type vertex, alors que dans son cours Nicolas présentait surtout la manipulation d'assets de type bitmap.

A ce stade du développement, le moteur d'animation est loin d'être terminé, il reste encore beaucoup de travail, comme par exemple :

- améliorer le système de collision, qui pour l'instant ne fonctionne bien qu'avec les parallélogrammes,
- explorer différentes possibilités d'améliorer le comportement des sprites, de leur donner un peu d'intelligence et d'autonomie
- tester des pistes alternatives pour la gestion d'un DOM virtuel ("shadow DOM" ou "canvas ghost")
- vérifier la transmission du clic de souris sur les enfants et petits-enfants d'un "sprite" parent (j'ai l'impression qu'à ce stade cela ne fonctionne pas très bien, mais je n'ai pas eu le temps d'investiguer)
- documenter correctement le code (doc quasi inexistante pour l'instant)
- refactoriser et nettoyer certaines parties du code,
- packager le projet pour le rendre plus facilement intégrable dans d'autres projets
- pour l'instant, le moteur est très dépendant de P5. Or j'avais souhaité le rendre utilisable aussi bien avec P5 que sans P5, ce qui n'est peut être pas si réaliste, car cela me contraint à certaines contorsions peu satisfaisantes au niveau du code. Peut être serai-je amené à faire une version pour P5, et une autre version indépendante, mais je vais retarder la décision car j'ai d'autres points plus prioritaires à améliorer.

... bref, la vie normale d'un projet, en somme.

D'un point de vue algorithmique, la création d'un moteur d'animation 2D est extrêmement intéressante. Cela amène à réfléchir à des problématiques d'architecture, à manipuler des design patterns souvent abstraits, qui deviennent plus concrets avec un projet visuel comme celui-ci.

Bref, si c'était à refaire, je referais exactement pareil, avec les mêmes tâtonnements, les mêmes "retour arrière", les mêmes découvertes et surtout le même plaisir.

Ce projet va certainement avancer de manière irrégulière, car c'est un projet mené sur mon temps libre (qui n'est pas hyper extensible). J'espère trouver le temps de poursuivre ce travail qui est passionnant.

Bibliographie

Comment créer un moteur d'animation 2D, de Nicolas Legrand :

<https://javascript.developpez.com/tutoriels/html5/creer-moteur-affichage-2d-html5/>

Plusieurs ouvrages importants pour maîtriser certains concepts graphiques, mathématiques et physiques de l'animation 2D. Certains de ces livres sont consacrés à Actionscript, une version de Javascript bâtie par les ingénieurs d'Adobe pour les besoins de Flash.

Programming Macromedia Flash MX, par Robert Penner's, ed Mc Graw Hill (2002)

Le livre de Robert Penner présente beaucoup de notions essentielles pour coder de l'animation 2D. Le livre est écrit pour le langage Actionscript qui est une déclinaison de Javascript. Donc de nombreuses techniques présentées dans le livre sont transposable en Javascript (moyennant parfois quelques adaptations). Attention, c'est un livre destiné à des développeurs aguerris, je ne le recommande pas à un développeur débutant en JS. Le livre est aujourd'hui épuisé et difficile à trouver, mais il y a un chapitre qui est téléchargeable librement en PDF, c'est celui sur les fonctions « easing ». Ces fonctions permettent de module le rythme d'une animation, et elles ont été largement utilisées dans des frameworks comme D3.js, jQuery et quelques autres. Le chapitre du livre dédié aux fonctions "easing" est disponible sur cette page, avec en bonus des exemples d'implémentation dans différents langages : <http://robertpenner.com/easing/>

Foundation ActionScript 3.0 Animation, by Keith Peters, ed Friends of ED / Apress (2007)

Très bon livre de Keith Peters, qui est l'animateur de la chaîne Youtube "the codin math". Ce livre est très complémentaire de celui de Robert Penner. Il existe une version JS de ce livre, c'est celle que je décris ci-dessous.

Foundation HTML5 Animation with Javascript, Billy Lamberta, ed Friends of ED / Apress (2011)

Excellente adaptation au JS, par Billy Lamberta, du livre de Keith Peters. Tous les exemples du livre de Keith Peters sont ici traduits en pur JS, et on peut dire que ce livre constitue une sorte de pierre de Rosette pour tout développeur désireux de comprendre les différences entre Actionscript et Javascript.

Flash Math Creativity, 2nd edition, ed Friends of ED / Apress (2004)

Plein de belles idées à explorer dans ce beau livre, qui démontre qu'en terme de créativité, les développeurs Actionscript étaient véritablement au top, durant l'âge d'or de Flash (soit les années 2000).

Les livres Apress peuvent être achetés en format numérique sur ce site : <https://www.apress.com/>

Changelog

version 0.2 : présentée au meetup CreativeCodeParis le 20 décembre 2018

version 1.0 : correction de quelques coquilles, ajout d'un chapitre "premiers tests" qui contient notamment quelques questions posées durant le meetup. Ajout aussi d'un chapitre "conclusion provisoire"

version 1.1 : correction de nombreuses coquilles dans le texte