Projet étudiant

15 février 2018

Framework d'animation en Javascript basé sur p5.js

Étudiants :

Pierre GRANIER--RICHARD

Dylan BUNEL

Thibaut ROPERCH

Projet encadré par :

Jean-Michel RICHER

Table des matières

Introduction

Au sein d'une page web, une applet Java permet d'illustrer un concept de par la bibliothèque graphique fournie avec Java et sa possibilité d'interaction. Cette technologie était réputée pour sa portabilité (fonctionne sur la plupart des navigateurs web et systèmes d'exploitation).

Cependant, depuis fin 2015, de nombreux navigateurs ont arrêté de prendre en charge l'intégration de certaines technologies, dont l'applet Java. Ainsi, depuis la version 9 de Java, de nombreux navigateurs considèrent les applets Java comme dépréciées. Il est donc nécessaire de trouver une ou plusieurs alternatives à cette technologie, comme DHTML, Flash ou encore Javascript.

Présentation du framework Java

Le framework Java réalisé par monsieur Richer permet de disposer des objets de base tels qu'une image, un rectangle, du texte, et de les soumettre à des animations comme les déplacer, faire clignoter, faire apparaître ou disparaître.

Pour décrire les objets et les animations, nous devons suivre une DTD et écrire un fichier XML par animation.

L'objectif de notre travail est de nous approprier ce framework codé en Java pour le traduire en Javascript. Nous utiliserons le framework p5.js qui dispose des primitives de base pour le dessin et l'animation.

Notre travail consiste d'abord à comprendre le fonctionnement du framework actuel en analysant le format de représentation d'une animation et en étudiant le code existant. Ensuite, nous apprendrons à utiliser le framework p5.js, puis nous coderons les fonctionnalités implémentées dans la version Java du framework.

Représentation d'une animation

Une animation, écrite en XML, est représentée par des objets de dessin et au maximum un programme associé à chaque objet, programme contenant une suite d'instructions permettant l'animation. Les programmes de chaque objet de dessin sont à exécuter en parallèle, mais leurs instructions sont séquentielles.

Le framework actuel permet donc de lire un fichier XML, d'instancier en Java les objets de l'animation, de les dessiner et exécuter leurs programmes.

Le fichier XML représentant une animation est décrit par une DTD et se compose de quatre parties (noeuds) :

- **init (optionnel)** Affiche un déclencheur permettant de lancer l'animation (bouton)
- **background (optionnel)** Spécifie une image de fond
- **objects** Décrit les objets intervenant dans l'animation
- **programs** Associe des actions aux objets

Les objets

Selon la DTD, un objet est un noeud XML du même nom que l'objet, et ses propriétés sont des attributs. L'identifiant d'un objet est donné dans la valeur du noeud XML.

Les objets disponibles sont les suivants :

- **object_text** Du texte
- **object_rectangle** Un rectangle
- **object_image** Une image
- **object_polygon** Une forme fermée, définie par une suite de coordonnées
- **object_ellipse** Une ellipse

Les propriétés communes aux objets sont les suivantes :

- **identifiant** L'identifiant unique des objets (chaîne de caractères)
- **x** Position horizontale de l'objet dans le dessin
- **y** Position verticale de l'objet dans le dessin
- **fgcolor** et **bgcolor** Respectivement la couleur de la bordure ou du texte et la couleur d'arrière-plan de l'objet
- **layer** La couche de l'objet dans le dessin, les couches supérieures étant celles avec un plus grand nombre
- En interne, l'**état** de l'objet (en mouvement, clignotant, …)

D'autres propriétés, telles que la largeur, la hauteur, la transparence sont propres à certains objets. Nous détaillerons les propriétés de chaque objet plus tard.

Les programmes

Selon la DTD, un programme est composé d'une suite d'instructions, telles que des actions de déplacement, de temporisation, d'affichage.

Les actions de déplacement proposées par le framework sont les suivantes :

- **setx**, **sety**, **setxy** Change les coordonnées de l'objet
- **moveto** Déplace l'objet aux coordonnées spécifiées
- **up**, **right**, **down**, **left** Déplace l'objet respectivement vers le haut, la droite, le bas et la gauche d'un nombre de pixels donné à une vitesse donnée (nombre de pixels par image)

Les actions de temporisation proposées par le framework sont les suivantes :

- **sleep** Immobilise l'objet durant un nombre donné d'images

- **wait** L'objet attend d'être déclenché par un *token* donné
- **trigger** Donne un *token* donné à un objet donné (n'a aucun effet si le *token* donné n'était pas attendu par l'objet)

Les actions d'affichage proposées par le framework sont les suivantes :

- **visible** Rend visible ou invisible l'objet
- **blink** Fait clignoter l'objet un nombre de fois donné à un intervalle de temps donné

Les actions propres à un programme sont les suivantes :

- **label** Place un jalon à un endroit du code
- **goto** Revient à un jalon donné

Les autres actions proposées par le framework sont les suivantes :

- **setproperty** Change la valeur d'une propriété donnée de l'objet
- **click** L'objet attends d'être cliqué
- **stop** Arrête l'exécution du programme

Utilisation du framework

Les objets et instructions proposées permettent beaucoup d'animations possibles. De plus, le framework est suffisamment personnalisable pour rendre possible l'ajout d'autres objets.

Étant utilisé dans un cadre pédagogique, ce framework permet notamment de réaliser des POC (*Proof Of Concept*).

Travail effectué

Remaniement de la DTD

Après avoir analysé le travail existant, notamment la DTD, nous avons trouvé judicieux de la modifier pour plus de clarté et de flexibilité. Voici un résumé des modifications que nous avons apportées et un résumé de la nouvelle DTD.

Nous avons ajouté un noeud `<speed>` permettant de changer la vitesse de rafraichissement du dessin : *very slow*, *slow*, *normal*, *fast*, *very fast*.

Fidèle à la version Java du framework, le bouton de démarrage d'animation (`start_button`) est facultatif et est contenu dans le noeud `<init>`. Le bouton est par défaut visible, centré dans la fenêtre, et contient le texte "Click me to start".

On considère désormais que tous les objets peuvent avoir une bordure et une couleur de bordure. Pour ne pas confondre les attributs et pour des raisons de cohérence, nous avons changé leur nom :

- L'attribut `transparency` est renommé en `bgtransparent` et est disponible pour tous les objets. Cette propriété vaut `true` lorsque l'arrière-plan de l'objet est transparent et `false` sinon (`true` par défaut).
- L'attribut `fgcolor`, indiquant la couleur d'un texte, est renommé en `color` pour cet objet (toujours `rgb(255, 255, 255)` par défaut)
- L'attribut `fgcolor`, indiquant la couleur de la bordure lorsqu'il est appliqué aux autres objets que l'objet `texte`, est renommé en bocolor. L'objet `texte` possède également cet attribut, permettant ainsi de différencier la couleur du texte avec celle de la bordure.
- L'attribut `botransparent`, qui vaut `true` lorsque la bordure de l'objet est transparente et `false` sinon (`true` par défaut), est ainsi mis en place et disponible pour tous les objets.

Aussi, nous différencions la notion de visibilité d'un objet avec sa capacité à avoir un certain degré de transparence. Les propriétés suivantes ont donc été ajoutées pour tous les objets :

- `visible` (`true|false`, `false` par défaut), indiquant si l'objet est visible ou non
- `opacity` (nombre décimal entre 0 et 1, 1 par défaut), indiquant l'opacité de l'objet (utile notamment en cas de superposition d'objets)

Cette différenciation permet d'améliorer l'effet de clignotement d'un objet (nous détaillerons cela plus tard).

Les propriétés suivantes ont été ajoutées pour l'objet texte :

- `width` (entier supérieur à 0, la largeur du texte par défaut) : largeur de la boite contenant le texte
- `height` (entier supérieur à 0, la hauteur du texte par défaut) : hauteur de la boite contenant le texte
- `halignment` (`left|center|right`, `left` par défaut) : alignement horizontal du texte
- `valignment` (`top|bottom|center|baseline`, top par défaut) : alignement vertical du texte

La propriété suivante a été ajoutée pour l'objet rectangle :

- `round` (suite d'entiers supérieurs à zéro, `"0, 0, 0, 0"` par défaut) : rayons d'arrondissement des angles du rectangle, respectivement le coin en haut à gauche, en haut à droite, en bas à gauche et en bas à droite.

Ces propriétés de l'objet image ont été modifiées :

- `width` (largeur de l'image) : largeur de l'image par défaut
- `height` (hauteur de l'image) : hauteur de l'image par défaut

Les objets suivants ont été ajoutés :

- `object_landmark` : c'est un repère instancié avec, en plus des attributs communs aux objets, une largeur, une hauteur, une échelle en abscisse, une échelle ordonnée, le nom des abscisses et le nom des ordonnées.
- `object_copy` : c'est un objet qui copie les caractéristiques d'un autre objet instancié avec, en plus des attributs communs aux objets, l'identifiant de l'objet à copier.
- `object_grid` : c'est une grille instanciée avec, en plus des attributs communs aux objets, le nombre de lignes, le nombre de colonnes, la hauteur des lignes et la largeur des colonnes. L'instanciation d'une grille aurait également pu se faire en spécifiant la largeur et la hauteur finale de l'objet à la place de la hauteur des lignes et la largeur des colonnes, cependant, après en avoir discuté avec notre professeur, nous avons opté pour la première solution.

De façon générale, toutes les propriétés peuvent être données lors de la déclaration d'un objet et modifiées avec l'instruction `setproperty`, hormis la propriété `state` qui est modifiable avec des instructions spécifiques. L'état (attribut `state`) peut avoir les valeurs suivantes :

- *normal* (valeur par défaut) : l'objet n'est pas bloqué, typiquement lorsque son programme a terminé son exécution ou lorsque l'objet vient de sortir d'un état particulier.
- *sleeping* : l'objet est dans cet état lorsque l'instruction `sleep` est exécutée, indiquant à l'objet d'attendre un certain temps.
- *waiting_click* : l'objet est dans cet état lorsque l'instruction `click` est exécutée. Seul un clic sur cet objet peut remettre l'état de l'objet à *normal*. Par ailleurs, lorsqu'un objet est cliqué, une instruction `trigger` est créée en interne et exécutée sur l'objet cliqué avec la valeur *waiting_click*.
- *moving* : l'objet est dans cet état lorsqu'il est en mouvement (i.e. lorsque ses coordonnées sont progressivement modifiées). Les instructions ordonnant un déplacement sont les seules à mettre l'objet dans cet état.
- N'importe quel état donné par une instruction `wait`. Dans cette situation, l'objet attend une certaine valeur et ne peut retourner dans l'état normal que lorsqu'une instruction `trigger` est exécutée sur l'objet avec la valeur qu'il attend.

Cette différenciation d'états assure la séquentialité des instructions d'un programme. En effet, un programme est représenté par une boucle qui dépile et exécute la prochaine instruction. Ainsi, grâce aux états, il est possible de contrôler la temporisation des instructions : la prochaine instruction s'exécute si et seulement si l'objet est dans l'état "normal".

Étant donné que les états des objets peuvent être changés de manière contrôlée - c'est-à-dire en utilisant les états prédéfinis - avec des instructions vues jusqu'ici, l'instruction `state` qui change l'état n'est pas utile ; nous l'enlevons donc de la DTD.

Les instructions `label`, `goto`, `stop` sont propres à l'exécution du programme, elles n'agissent pas directement sur l'objet auquel le programme est associé.

L'instruction `blink` change l'opacité de l'objet plutôt que sa visibilité ; ainsi, l'objet est cliquable lorsqu'il clignote, même pendant ses frames d'invisibilité.

Les instructions de déplacement `MoveTo`, `Up`, `Right`, `Down` et `Left` changent progressivement les coordonnées d'un objet. Pour ce faire, la distance totale du déplacement est calculée et divisée par la vitesse de déplacement donnée.

Framework p5.js

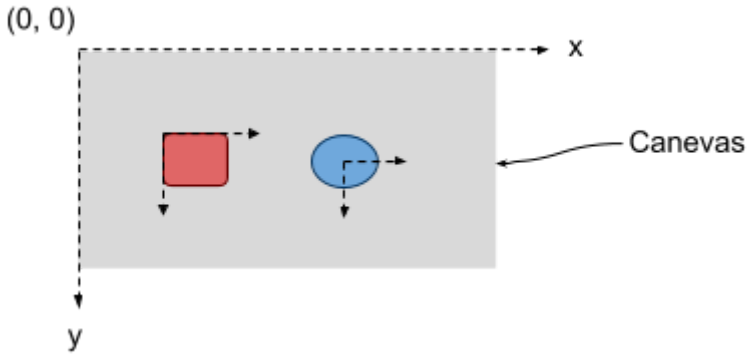
Le framework P5.js est un framework qui permet de dessiner des objets classiques tels qu'un rectangle, une image, une ligne, un champ textuel, en spécifiant entre autres les coordonnées de l'objet, sa couleur de remplissage ou encore la couleur de sa bordure.

L'utilisation du framework p5.js est relativement simple ; la fonction principale implémentée par le framework, `draw`, est appelée en boucle. À chaque itération, les figures déclarées dans cette boucle sont dessinées. Nous obtenons une animation en changeant progressivement les propriétés d'une figure - sa taille, sa position, sa couleur - à chaque itération de la fonction draw.

Prenons par exemple l'animation suivante : nous désirons déplacer une ellipse sur la largeur du canevas, de la gauche vers la droite. Pour ce faire, nous déclarons une variable en dehors de la fonction de dessin. Cette variable représente la position de la figure sur l'axe horizontal, initialisée à 0. À chaque itération de la fonction `draw`, donc à chaque image du dessin, cette valeur est incrémentée de 2. En spécifiant la vitesse de rafraichissement du dessin à 30 images par seconde, notre figure aura parcouru une distance de 2*30 soit 60 pixels vers la droite en une seconde.

Ainsi, il faut conserver les figures dans une tableau et, à chaque itération de la fonction de dessin, les afficher.

L'origine de la plupart des objets est en haut à gauche de ces derniers, l'origine de l'objet ellipse étant en son milieu.



Création des classes

Nous avons une classe par objet et par instruction, comme la version originale codée en Java. Nous avons également une classe associée à une animation pour permettre à une page HTML de contenir plusieurs animations.

Objets

Les objets sont les mêmes que dans la version Java, avec en plus les objets `landmark` et `grid` (`clone` ne fait pas l'objet d'une classe ; le type de l'objet à cloner est récupéré et un nouvel objet de ce type est directement instancié lors de la lecture du fichier XML).

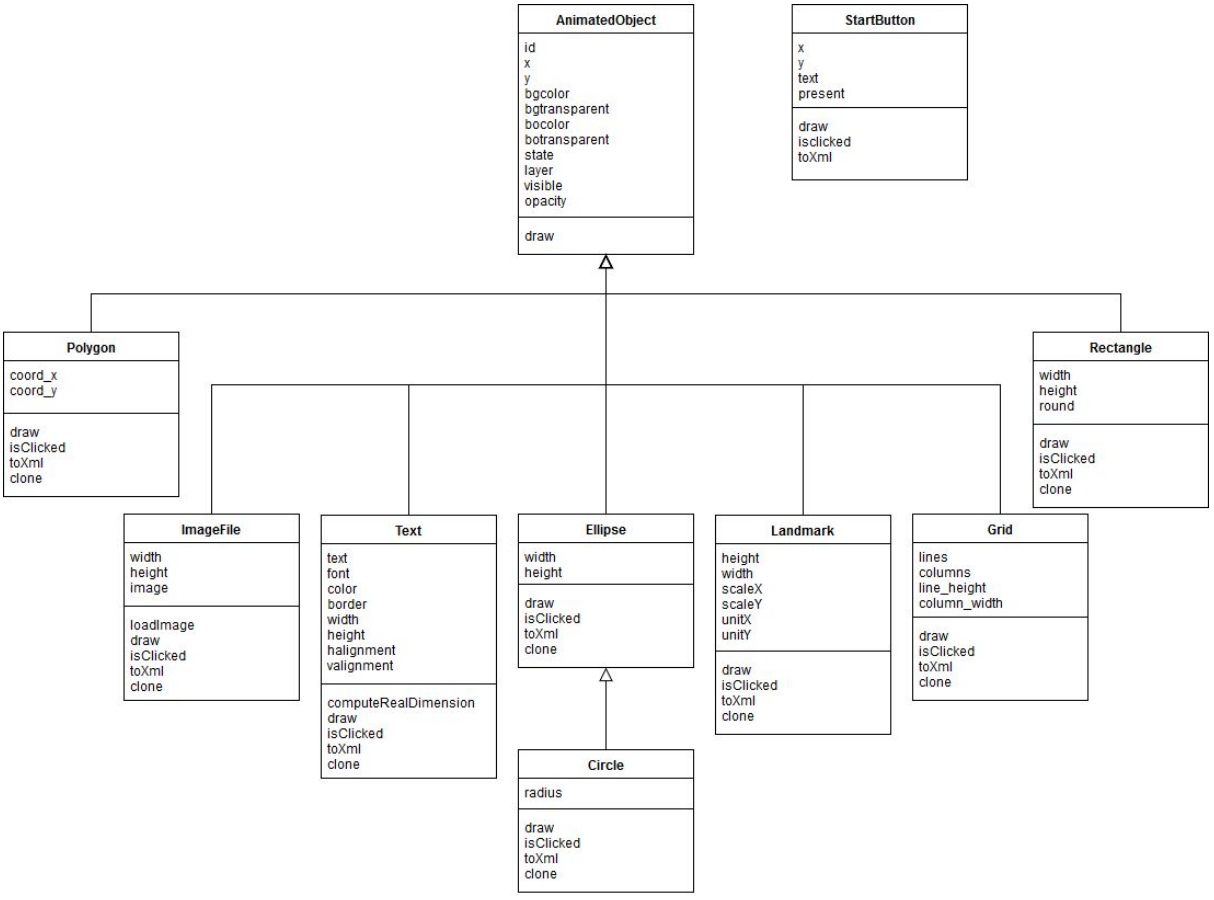


Diagramme de classes des objets

Ce diagramme de classe permet de visualiser l'ensemble des objets proposés par notre framework.

La classe mère `AnimatedObject` implémente une fonction de dessin dans laquelle la couleur de l'arrière plan et de la bordure sont initialisés en fonction de la transparence, la visibilité et l'opacité de l'objet.

Lorsqu'ils sont dessinés, les objets héritant de `AnimatedObject` appellent d'abord la méthode `draw` de la classe parente avant de se créer à leurs coordonnées. Toutes les méthodes `draw` - et la méthode `loadImage` - prennent en paramètre le canevas sur lequel doit être dessiné l'objet.

Chaque classe enfant a sa propre méthode de clonage qui créé un nouvel objet avec les mêmes propriétés que l'objet à cloner.

Chaque classe enfant et la classe `StartButton` possède sa propre méthode de conversion en XML, méthode appelée par l'assistant graphique dont nous présenterons l'intérêt et les fonctionnalités plus tard. De plus, ces classes implémentent une méthode retournant `true` dans le cas où la position donnée de la souris se situe dans l'image, `false` sinon.

La méthode `computeRealDimension` de la classe `Text` permet de calculer la largeur et la hauteur de l'objet en fonction de la taille de sa police. Ce calcul est nécessaire lorsqu'aucune dimension n'est spécifiée lors de l'instanciation de l'objet.

Instructions

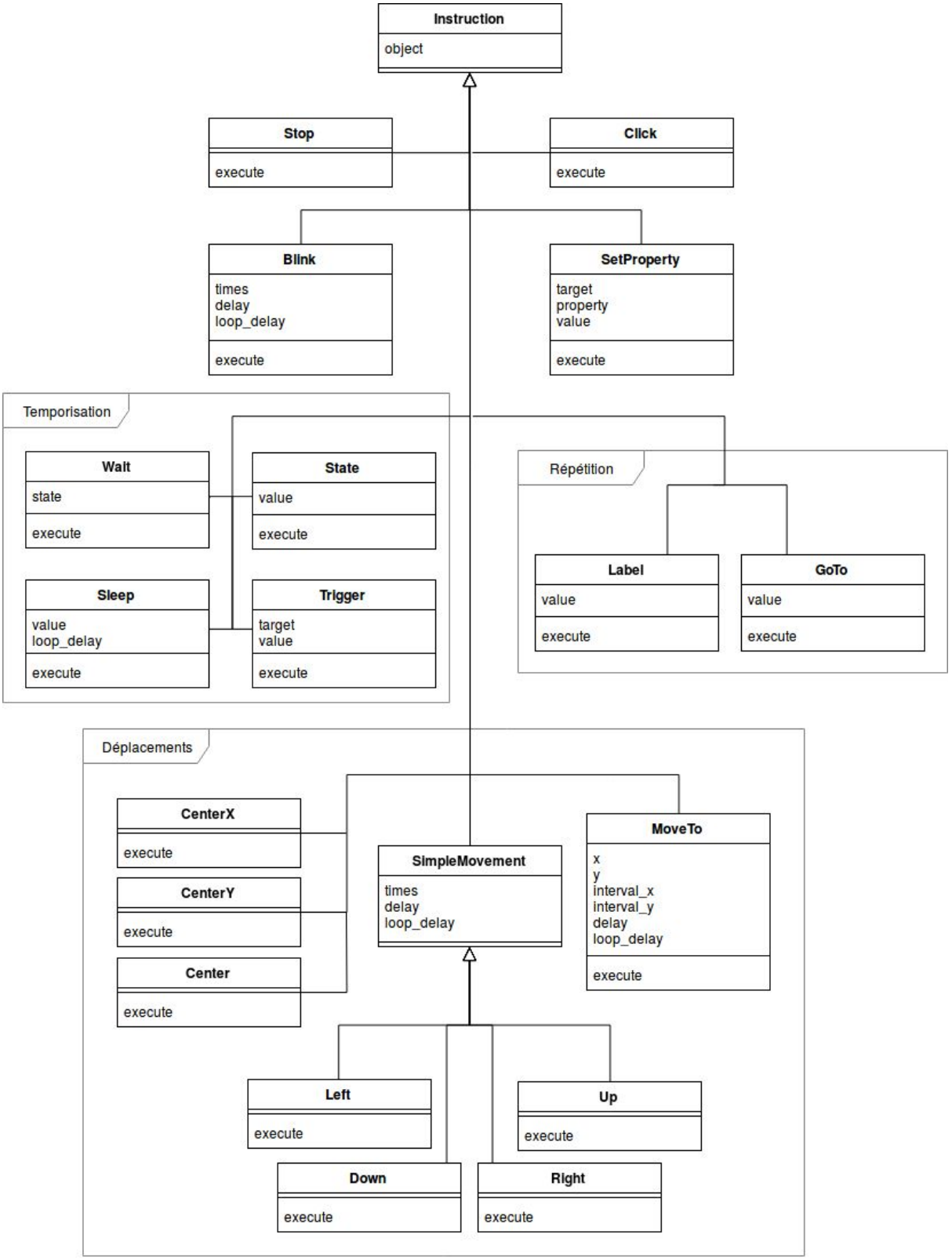


Diagramme de classes des instructions

Une instruction est liée à un programme qui est lui-même associé à un objet. Ainsi, tous les types d'instructions possèdent l'identifiant de l'objet propriétaire de l'instruction. Les classes instanciables doivent toutes implémenter la méthode `execute`, appelée lors de l'exécution du programme auquel elles appartiennent.

La classe `State` est relative à l'instruction `state` que nous avons supprimée de la DTD par rapport à la version Java ; nous l'avons implémentée pour que les animations existantes fonctionnent avec notre framework. Lorsqu'elle est instanciée, un message d'erreur stipulant que l'utilisation de l'instruction `state` est dépréciée apparaît dans la console, mais l'animation reste fonctionnelle.

Étant donné que la classe `setProperty` permet de modifier n'importe quelle propriété d'un objet, les instructions qui modifient une propriété en particulier (`setx`, `sety`, `setxy`, `visible`) entraînent une instanciation de la classe `setProperty` ; ainsi, contrairement à la version Java du framework, nous n'avons pas les classes `SetX`, `SetY`, `SetXY`, `Visible`.

Pour faciliter le positionnement des objets, nous ajoutons les instructions et classes suivantes :

- `centerX` Permet de centrer l'objet sur l'axe horizontal
- `centerY` Permet de centrer l'objet sur l'axe vertical
- `center` Permet de centrer l'objet sur les deux axes

Nous représentons un programme par une pile d'instructions.

Animation

Un objet de type Animation est instancié avec le chemin du fichier XML contenant les objets et les programmes, la balise HTML recevant le canevas de l'animation, la largeur et la hauteur de cette dernière.

La classe Animation est chargée de lire le fichier XML contenant l'animation (affiche un message d'erreur au cas où le fichier est inexistant), c'est-à-dire qu'elle instancie les objets et instructions au fur et à mesure qu'elle parcourt le fichier XML.

A l'issue de l'analyse du fichier XML, le programme de chaque objet est exécuté en même temps.

Écriture du programme principal

Le programme principal contient la fonction `include_animation_files` qui inclut dans la page HTML tous les fichiers Javascript nécessaires au bon fonctionnement du framework, à savoir les classes présentées précédemment et le framework p5.js.

Lorsqu'une animation est demandée à être chargée avec la fonction `load_animation`, le programme principal va instancier une animation, récupérer en AJAX le contenu du fichier

XML contenant l'animation et demander à l'objet Animation créé de traiter le contenu du fichier à l'aide d'un objet DOMParser.

Enfin, la fonction `draw_animation` est automatiquement appelée lorsque tous les objets et instructions de l'animation sont instanciées. En utilisant le framework p5.js, cette dernière demande à l'animation de charger ses images (fonction `preload` de p5.js), créer le canevas dans la balise HTML parente (fonction `setup` de p5.js), et dessiner les objets (fonction `draw` de p5.js). Les évènements associés au clic de la souris sont gérés avec la fonction `mouseClicked` de p5.js.

Assistant graphique

Afin de faciliter la création d'une animation qui, jusqu'ici, se résume à déclarer à la main les objets et les instructions dans un fichier XML, nous avons mis en place une interface graphique proposant de créer des objets, modifier leurs propriétés, prévisualiser le dessin en direct et l'enregistrer dans un fichier XML.

L'assistant graphique permet ainsi de rendre la création d'animation et par extension l'utilisation du framework plus abordables.

Conclusion

Ce projet a aboutit à la création d'un framework fonctionnel d'animation d'objets, aisément modulable. En effet, de part sa flexibilité et sa conception, il est tout à fait possible d'ajouter au framework de nouveaux objets et de nouvelles instructions.

Initialement destiné à une utilisation pédagogique, nous pouvons étendre ce framework à d'autres usages comme par exemple la création de maquettes (web design, level design, …) ou encore la conception de diaporama.

Ce projet nous a permis de découvrir le framework p5.js et de nous améliorer dans ce domaine du développement web et plus spécifiquement dans l'analyse, la conception et le développement d'un framework graphique.

Références

Documentation de p5.js :

https://p5js.org/reference/

Sources du projet :

https://github.com/ThibautRoperch/FrameworkAnimationJS

Sources du framework :

https://github.com/ThibautRoperch/FrameworkAnimationJS/tree/master/VersionJS/js/AnimationFramework

Exemple d'utilisation :

https://github.com/ThibautRoperch/FrameworkAnimationJS/blob/master/VersionJS/cpu_anim1.html

Présentation et guide d'utilisation du framework :

https://github.com/ThibautRoperch/FrameworkAnimationJS/blob/master/VersionJS/index.html