Guide d'utilisation d'*algoGGB* (version visuelle)

Présentation de la programmation par blocs pour algoGGB

GeoGebra permet de créer (puis d'explorer interactivement) des figures mathématiques fort intéressantes, et ce de plusieurs façons :

- via une combinaison de commandes gestuelles et textuelles (textuelles : en utilisant son champ de saisie)
- via la création et l'utilisation d'outils fabriqués sur mesure
- via un langage de script, simple mais limité
- via un langage de programmation complet, puissant mais d'utilisation plutôt complexe (*JavaScript*).

Les ressources mises à notre disposition par *GeoGebra* sont tellement riches et variées qu'on peut se demander s'il sera jamais nécessaire d'avoir recours à la programmation *JavaScript* dans *GeoGebra*. Pourtant, il y a parfois des cas où ce type de programmation offre des avantages certains : par exemple

1. *Exploration d'autres types de géométries*
   GeoGebra est conçu pour l'étude et l'exploration de la géométrie euclidienne, que ce soit sous sa forme synthétique que sous sa présentation analytique. Mais on peut chercher à travailler dans le cadre d'autres géométries, comme celle de la **tortue**, popularisée par le mouvement Logo, et reprise par la suite dans d'autres contextes. Et, par sa nature même, la **géométrie de la tortue** nécessite une programmation dans un langage complet : pourquoi ne pas utiliser *GeoGebra* ?

2. *Exploration de figures tellement complexes qu'on doit les décrire par un programme*
   Certaines figures sont trop complexes pour être décrites manuellement, en énumérant les divers éléments qui les composent : les **fractals**, par exemple. En effet, plusieurs types de fractals sont de nature algorithmique, et on doit donc utiliser la programmation si on veut en obtenir des représentations. Ici encore : pourquoi ne pas utiliser *GeoGebra* ?

Si ces deux types d'utilisations font appel à une programmation, on peut se demander si *GeoGebra* est un contexte intéressant pour la réaliser. Après tout, il existe des environnements de programmation plus conviviaux que *GeoGebra*, et des langages plus efficaces et

plus rapides que *JavaScript*, pour réaliser de telles activités. Mais il faut aussi voir que *GeoGebra* offre des avantages non négligeables :

1. *GeoGebra* offre directement des facilités de manipulation des figures, ne requérant aucune programmation supplémentaire :
   - on peut toujours déplacer, agrandir et réduire les figures
   - on peut souvent même modifier celles-ci
   - on peut utiliser directement boutons et glissières

2. *GeoGebra* nous permet de placer aisément nos figures sur le web tout en préservant leur caractère interactif.

3. *GeoGebra* nous permet d'exporter nos figures sous différents formats, utilisables dans d'autres logiciels, notamment
   - le format vectoriel **SVG**, utilisable dans *Libre Office* et dans tous les navigateurs web modernes
   - le format matriciel **PNG**, avec divers choix de résolution, utilisable dans *Libre Office*, *Microsoft Office*, et dans tous les navigateurs web modernes.

Pour ces raisons, et conscient de la complexité de la programmation générale dans *GeoGebra*, j'ai écrit une bibliothèque de procédures permettant de simplifier l'utilisation de *JavaScript* dans *GeoGebra* : à court d'idées pour le nom, je l'ai nommée ***algoGGB***.

Mais, même simplifiée, la programmation en JavaScript peut s'avérer rebutante pour certains. C'est pourquoi j'ai aussi développé un éditeur visuel (par blocs) pour ***algoGGB***, en me basant sur la bibliothèque ***Blockly***, développée par **Google**. Pour plus de renseignement sur ***Blockly***, voir https://developers.google.com/blockly/.

Un exemple, pas à pas

Je vais maintenant décrire comment utiliser l'éditeur visuel d'***algoGGB*** pour réaliser un simple carré, en utilisant la géométrie de la tortue. Bien entendu, on n'a pas besoin de programmer pour réaliser un carré dans **GeoGebra** ; mais le but ici est d'apprivoiser l'éditeur visuel d'***algoGGB***, avec lequel on pourra faire des figures plus complexes par la suite.

Tout d'abord, un mot d'avertissement : mon but, dans ce manuel, est exclusivement de décrire comment utiliser ***algoGGB***, en version visuelle. Je vais donc supposer que le lecteur a des connaissances raisonnables de *GeoGebra*, de la programmation en général, et des domaines mathématiques auxquels je vais faire référence.

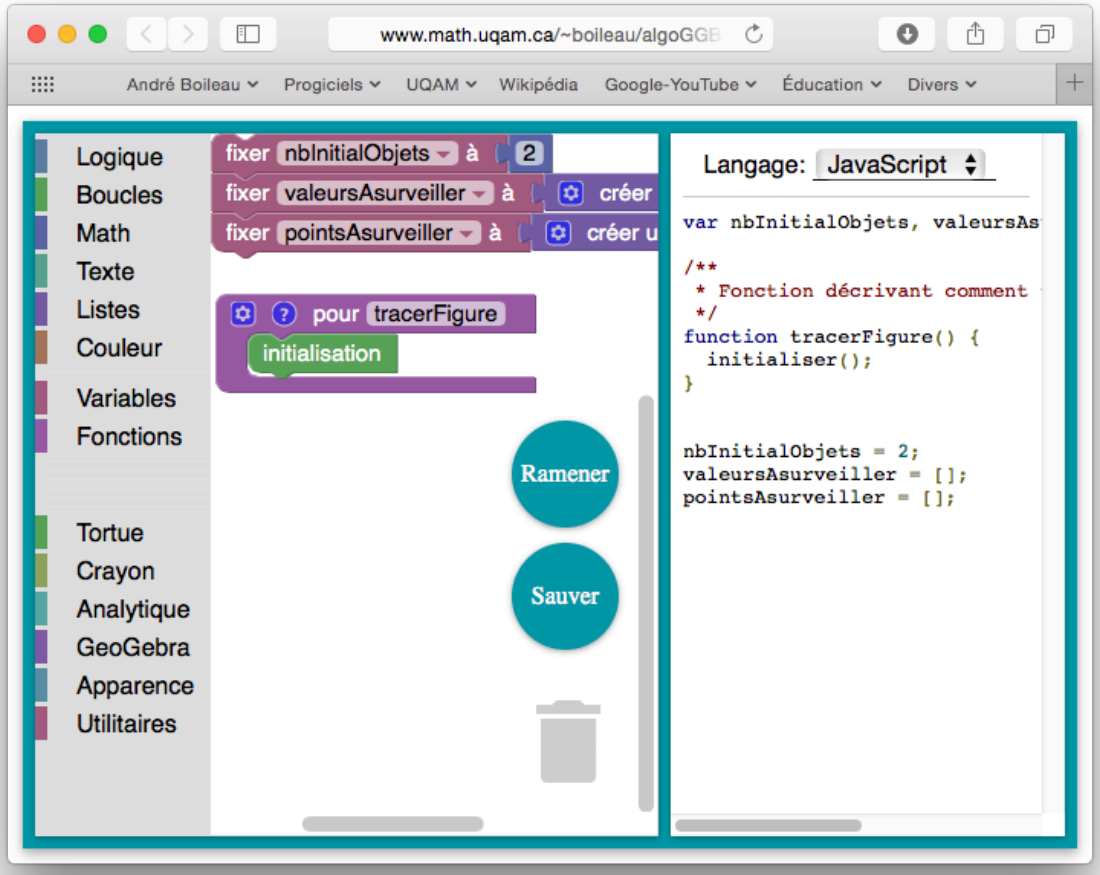
La première étape est d'accéder à l'éditeur visuel d'algoGGB. On y arrive en suivant le lien suivant

http://www.math.uqam.ca/~boileau/algoGGB/BLOCSalgoGGB.html

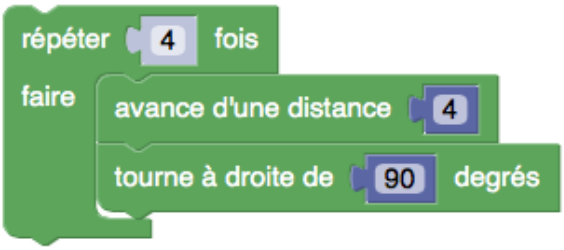
ou en téléchargeant cet éditeur à partir de la page suivante

http://www.math.uqam.ca/~boileau/algoGGB/blocs.html.

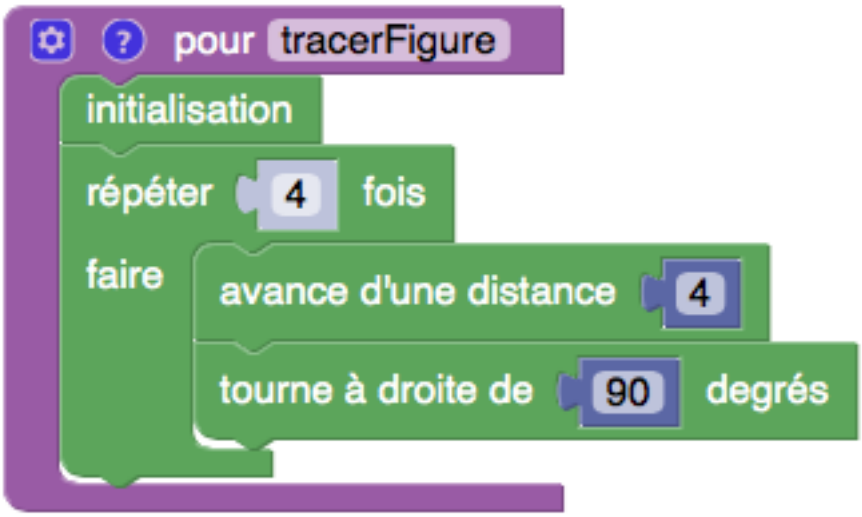
D'une façon ou d'une autre, vous arrivez à une page web qui ressemble à ceci



En géométrie de la tortue, pour tracer un carré, il suffit de répéter quatre fois les deux instructions suivantes : avancer d'une certaine distance, puis tourner de 90 degrés. Ceci se réalise en assemblant les blocs suivants :

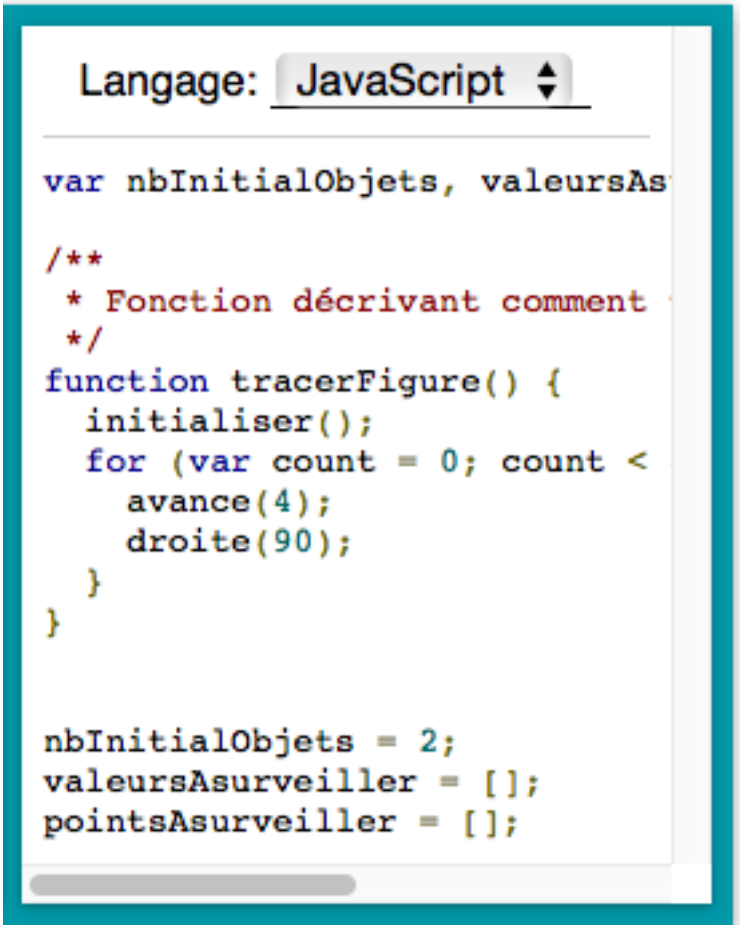


puis en plaçant le tout immédiatement en bas du bloc « initialisation » , dans la fonction « tracerFigure » : on obtient alors le résultat suivant



Voilà, notre programme est complété. Il suffit maintenant de le transférer dans GeoGebra.

Voici comment procéder : on remarque, à la droite des blocs, une zone de texte (ci-dessous) dans laquelle s'est écrit un programme JavaScript au fur et à mesure qu'on plaçait nos blocs. Point n'est besoin de comprendre ce qui est écrit : il suffit de copier la totalité du texte qu'on y trouve.

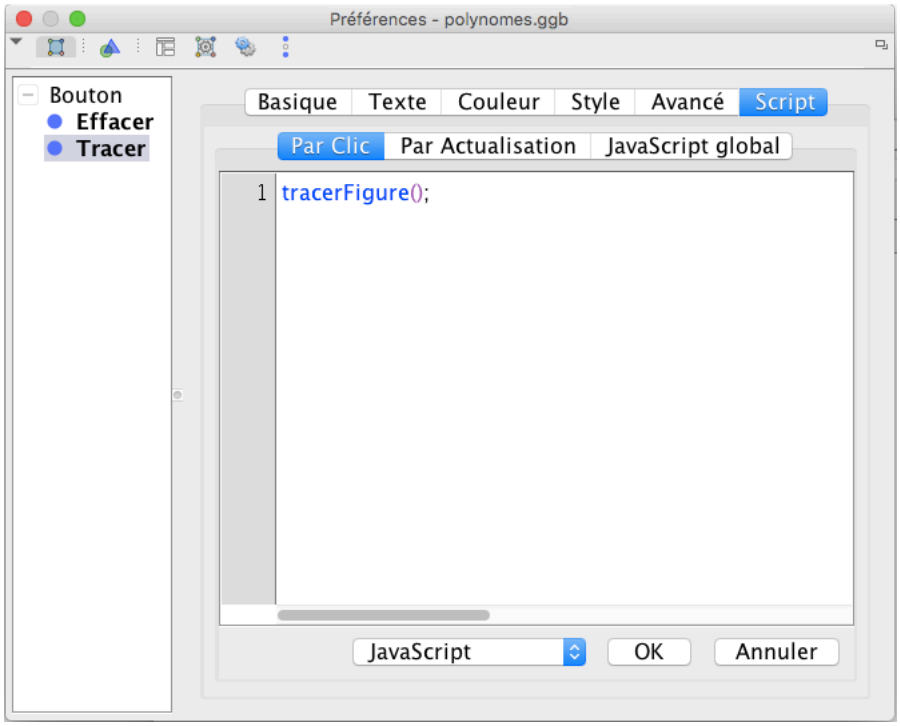


Au préalable, on aura téléchargé ***algoGGB*** à partir de la page web

http://www.math.uqam.ca/~boileau/algoGGB/telechargements.html.

L'environnement ***algoGGB*** se présente sous la forme d'une figure *GeoGebra* (appelée, vous l'avez deviné, ***algoGGB.ggb***) comportant deux boutons (**Tracer** et **Effacer**) et dotée d'une bibliothèque de procédures *JavaScript*. La première étape de notre démarche consistera donc à dupliquer cette figure *GeoGebra*, et à donner à sa copie le nom de ***carres.ggb***.

Examinons maintenant les propriétés du bouton **Tracer**, que l'on atteint par un clic droit sur ledit bouton, suivi du choix de l'item « Propriétés… » dans le menu local qui apparaît alors. Un clic sur l'onglet « Script », puis sur l'onglet « Par Clic » nous conduit à la situation décrite dans la figure ci-dessous.



On peut interpréter ceci comme suit : un clic sur le bouton **Tracer** exécutera la procédure *JavaScript* **tracerFigure**. De même, un clic sur **Effacer** dans la colonne de gauche nous apprendra qu'un clic sur ce bouton exécutera la procédure *JavaScript* **initialiser**.

Un clic sur l'onglet « JavaScript global » fera apparaître la bibliothèque de procédures ***algoGGB***. Mais, ne vous laissez pas impressionner par la quantité d'informations que vous y trouverez. Rendez-vous plutôt vers la fin du texte, pour retrouver la section « Procédures utilisateur » suivante :

```
// ********************************************************************************
// ****************** Procédures utilisateur (ci-dessous) ************************
// ********************************************************************************

var nbInitialObjets = 2; // Nombre d'objets au départ, à ne pas effacer
var valeursAsurveiller = [];  // Liste des noms des valeurs à surveiller
var pointsAsurveiller = []; // Liste des noms des points à surveiller

function tracerFigure(){
        initialiser() ;

}
```

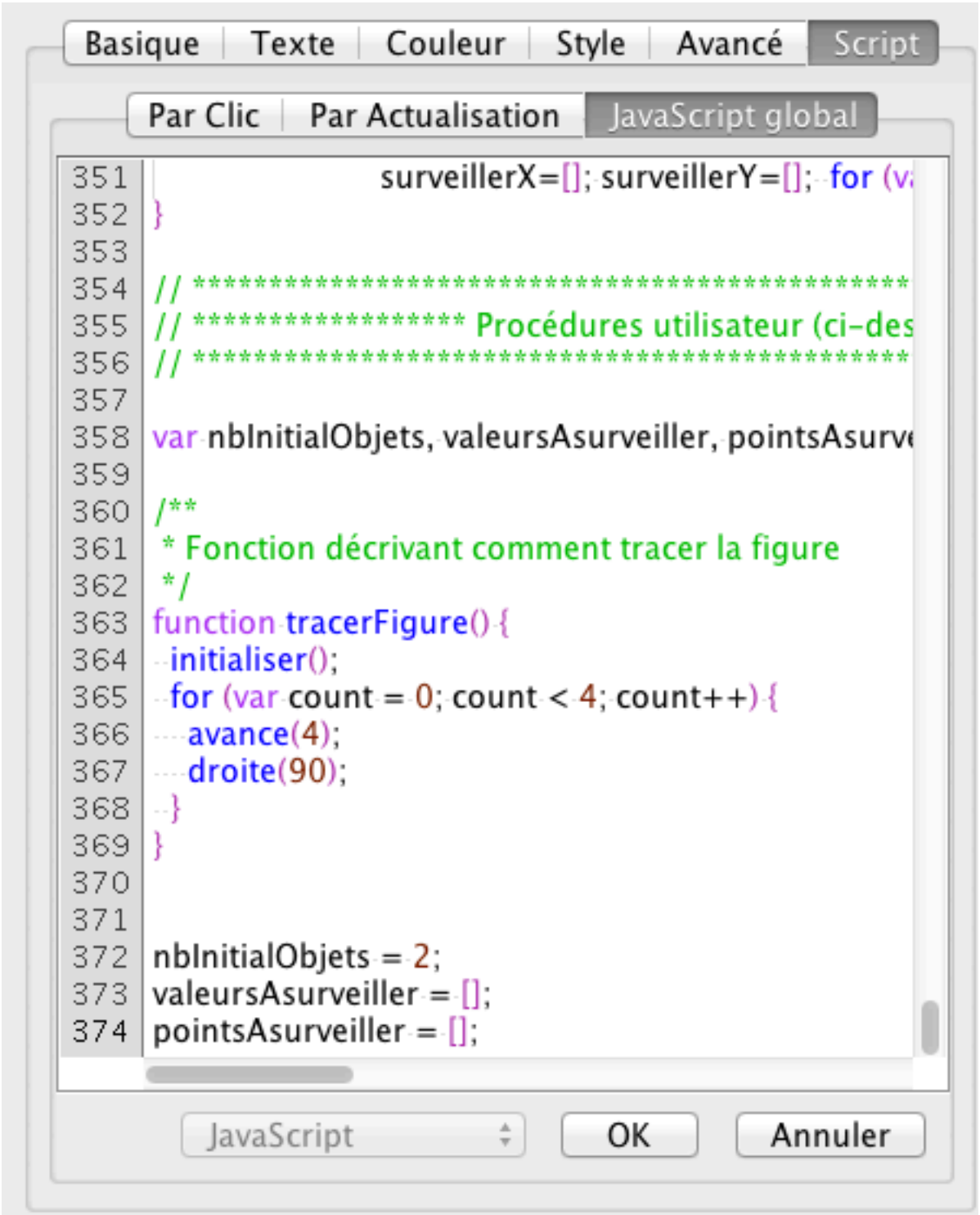
Nous devons maintenant remplacer toutes les instructions après les lignes

```
// ********************************************************************************
// ****************** Procédures utilisateur (ci-dessous) ************************
// ********************************************************************************
```

par celles que nous avons copiées tantôt, alors que nous étions dans l'éditeur visuel d'***algoGGB***. On obtient alors le résultat suivant :

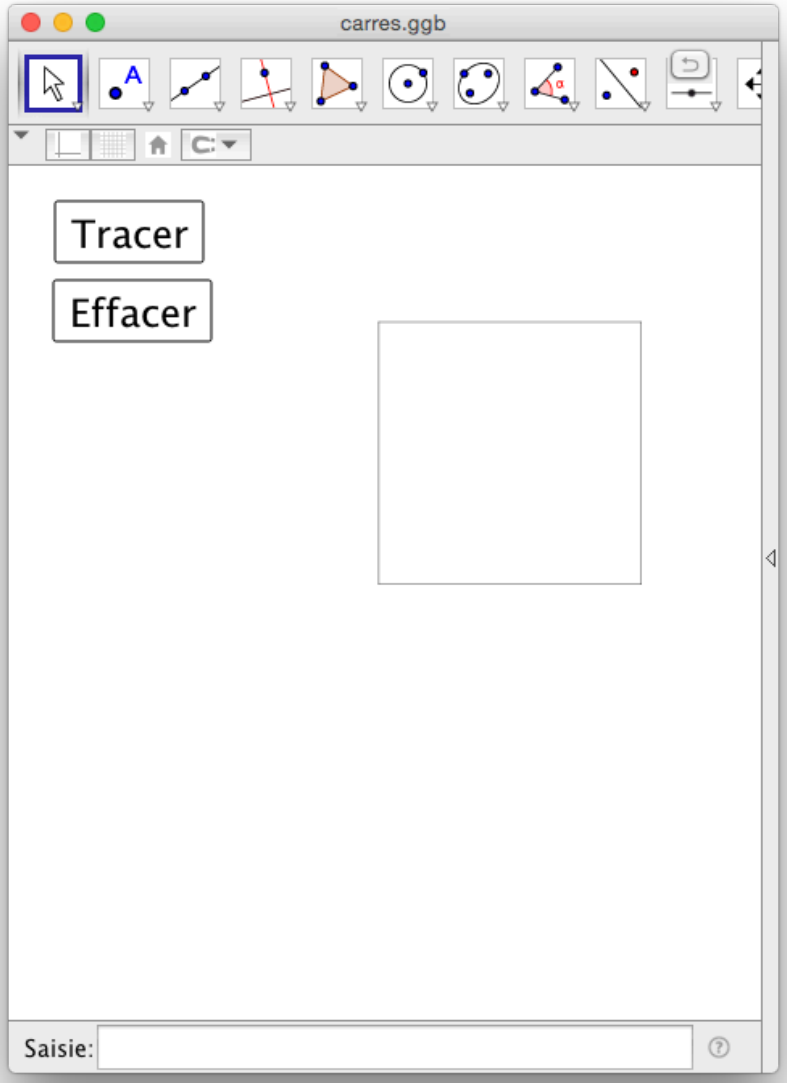
```
// ********************************************************************************
// ****************** Procédures utilisateur (ci-dessous) ************************
// ********************************************************************************

var nbInitialObjets, valeursAsurveiller, pointsAsurveiller;

/**
 * Fonction décrivant comment tracer la figure
 */
function tracerFigure() {
  initialiser();
  for (var count = 0; count < 4; count++) {
    avance(4);
    droite(90);
  }
}


nbInitialObjets = 2;
valeursAsurveiller = [];
pointsAsurveiller = [];
```

Voyons maintenant comment faire tracer notre figure. On commence tout d'abord par accepter les changements que nous venons d'apporter, en cliquant sur le bouton « Ok », puis on ferme la fenêtre « Préférences – carres.ggb ».




```
351                 surveillerX=[]; surveillerY=[]; for (v
352 }
353
354 // ********************************************
355 // ***************** Procédures utilisateur (ci-des
356 // ********************************************
357
358 var nbInitialObjets, valeursAsurveiller, pointsAsurv
359
360 /**
361  * Fonction décrivant comment tracer la figure
362  */
363 function tracerFigure() {
364   initialiser();
365   for (var count = 0; count < 4; count++) {
366     avance(4);
367     droite(90);
368   }
369 }
370
371
372 nbInitialObjets = 2;
373 valeursAsurveiller = [];
374 pointsAsurveiller = [];
```

On retrouve alors la fenêtre initiale de notre figure, et il suffit de cliquer sur le bouton « Tracer » pour faire apparaître notre carré.



Notons que nous pouvons utiliser toutes les commandes de **GeoGebra** (dont des translations et des zooms, si la figure n'est pas disposée à notre goût). Mais le bouton « Effacer » effacera tous les objets créés (y compris notre carré), tandis que le bouton « Tracer » ne fera apparaître que notre carré.

Remarquons que nous pouvons maintenant enregistrer la figure **GeoGebra**, et son programme ***JavaScript***. Mais que faire si nous voulons enregistrer le programme visuel, constitué de blocs ? Après tout, les navigateurs web sont conçus pour protéger l'ordinateur sur lequel ils fonctionnement, notamment en empêchant d'y enregistrer certains fichiers. Pour ce faire, retournons à notre éditeur visuel algoGGB, et remarquons les deux boutons « Ramener » et « Sauver » (ci-contre).

Voici comment procéder pour enregistrer / sauver votre programme

- Un clic sur le bouton « Sauver » fait apparaître une fenêtre contenant un texte qui décrit nos blocs, mais que nous n'avons pas à comprendre en détails.

- On commence par le copier dans le presse-papier.
  Par exemple,
  - On clique dans le texte pour l'activer
  - On choisit ensuite « Tout sélectionner » dans le menu « Édition »
    Pour accélérer, on peut utiliser l'équivalent clavier : cmd/ctrl-A.
  - Puis on choisit « Copier », toujours dans le menu « Édition ».
    Pour accélérer, on peut utiliser l'équivalent clavier : cmd/ctrl-C.
- Après avoir cliqué sur « Fermer », il suffit d'ouvrir un éditeur de textes simples (comme **Brackets, disponible** sur **Macintosh**, **Windows**, et **Linux**) et de le coller dans un document vierge, que nous enregistrons par la suite sous le nom que nous voulons donner à notre programme.

Pour ramener un programme, on fait la démarche inverse :

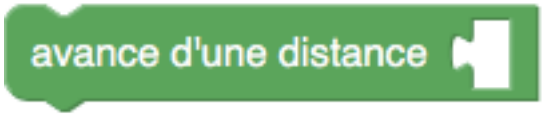
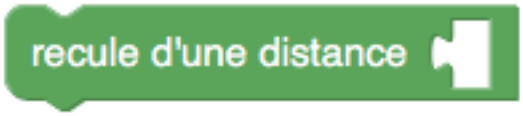
- On ouvre le fichier texte dans lequel il a été enregistré.
- On sélectionne la totalité du texte, et on le copie.
- On passe à l'éditeur visuel algoGGB, dans lequel on clique sur le bouton « Ramener ».
- La fenêtre suivante apparaît

- Il suffit de coller ce que vous avez copié.
  Ne vous en faites pas : même si tout le texte n'est pas visible, il est pris en compte.
- Vous cliquez sur « OK », et les blocs de votre programme apparaissent.

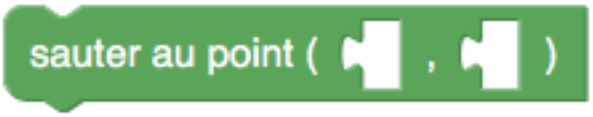
Après cet exemple, nous sommes prêts à examiner plus systématiquement les ressources mises à notre disposition par algoGGB : ce sera l'objet de la section « Référence ***algoGGB*** ». Nous décrirons les blocs spécifiques à algoGGB (à droite, ci-dessous), mais nous laisserons à d'autres le soin de décrire les blocs relatifs à JavaScript (à gauche, ci-dessous),.

Référence *algoGGB*

Tortue

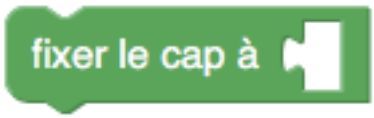
Blocs
 Pas une commande tortue, à proprement parler, mais place la tortue à l'origine, pointant vers l'axe des y positifs. Doit normalement se retrouver au début du programme pour tracer la figure. Efface tous les objets de GeoGebra sauf les **nbInitialObjets** premiers.
 Fait avancer la tortue d'une certaine distance. Les unités de longueur sont celles de GeoGebra.
 Fait reculer la tortue d'une certaine distance.
 Fait pivoter la tortue vers la droite d'un certain nombre de degrés.
 Fait pivoter la tortue vers la gauche d'un certain nombre de degrés.
 Retourne la coordonnée en X de la position de la tortue
 Retourne la coordonnée en Y de la position de la tortue

Amène la tortue en position (x,y). Laisse une trace si le crayon est baissé.



Amène la tortue en position (x,y).
Ne laisse pas de trace,. même si le crayon est baissé.

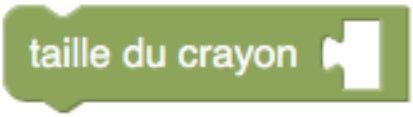
Retourne le cap de la tortue (en degrés)



Tourne la tortue dans la direction indiquée (en degrés).

Donne le cap qu'il faudrait avoir pour aller de la position actuelle de la tortue vers le point (x,y).

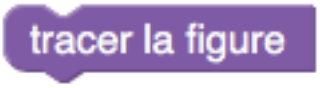
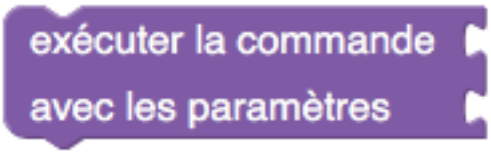
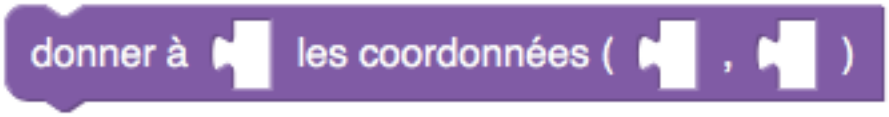
Crayon

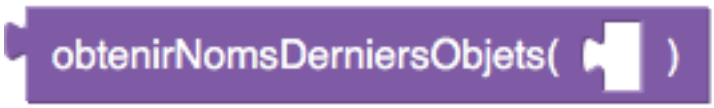
Blocs
 Une tortue avec crayon baissé laisse une trace en se déplaçant.
 Une tortue avec crayon levé ne laisse pas de trace en se déplaçant.
 Fixe l'épaisseur du trait (entre 1 et 13).
 Les paramètres rouge, vert et bleu varient entre 0 et 255.
 Les paramètres rouge, vert et bleu varient entre 0 et 255. Le paramètre a (variant entre 0 et 1) gère le degré de transparence.
 Après cette commande, toutes les commandes traçant des segments (avance, recule, fixePos, segment) contribueront à définir un polygone.
 Signale la fin de la définition du polygone en cours de définition. Celui-ci est alors tracé, en utilisant la couleur du crayon et de remplissage.

Analytique

Blocs
 Trace le segment reliant les points (x1,y1) et (x2,y2).
 Trace le cercle de centre (x,y) et de rayon r.
 Trace le disque de centre (x,y) et de rayon r.
 Trace un arc sur le cercle de centre (x,y) et de rayon r. L'arc va de l'angle a1 vers l'angle a2, dans la direction positive.
 Trace l'ellipse centrée en (x,y) et de demi-axes a et b.
 Trace l'ellipse remplie centrée en (x,y) et de demi-axes a et b. .
 Écris un texte, pouvant comporter plusieurs ligne, à partir de la position (x,y).

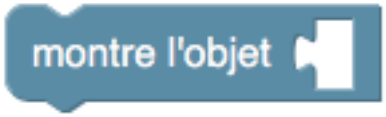
GeoGebra

Blocs
 **Doit être complétée par l'utilisateur.**
 Retrace la figure si une des valeurs de la liste **valeursAsurveiller** ou un des points de la liste **pointsAsurveiller** a été modifié.
 Engendre un nouveau nom commençant par la valeur de l'expression en entrée et suivi d'un nombre. Exemple : si l'expression en entrée est « poly », peut produire le nom 'poly137'. Noter que l'objet nommé n'est pas créé par cette commande.
 La première entrée est une commande GeoGebra (en anglais) comportant des expressions @1, @2, … Avant d'être exécuté, ces expressions seront remplacées par les valeurs des éléments correspondants de la liste placée en seconde entrée.
 Valeur d'un nombre ou d'une glissière GeoGebra identifié par son nom.
 Coordonnée en X d'un point GeoGebra identifié par son nom.
 Coordonnée en Y d'un point GeoGebra identifié par son nom.
 Fixe la valeur d'un nombre ou d'une glissière GeoGebra identifié par son nom.
 Fixe les coordonnées d'un point GeoGebra identifié par son nom.

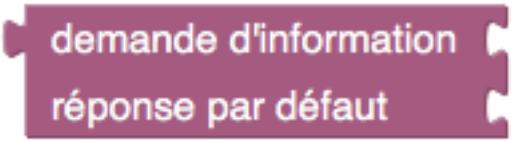


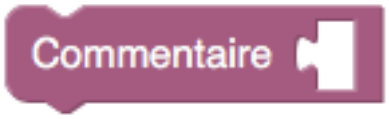
Retoure une liste contenant les noms des *n* derniers objets créés (du plus ancien au plus récent).
Utile quand une commande (exemples : **Polygon** ou un outil défini par l'utilisateur) créé plusieurs objets dont on veut modifier les propriétés.

Apparence

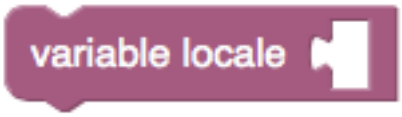
Description
 Cache un objet GeoGebra identifié par son nom.
 Cache l'étiquette d'un objet GeoGebra identifié par son nom.
 Rend fixe un objet GeoGebra identifié par son nom.
 Montre un objet GeoGebra identifié par son nom.
 Montre l'étiquette d'un objet GeoGebra identifié par son nom.
 Rend mobile un objet GeoGebra identifié par son nom.
 Fixe la couleur d'un objet GeoGebra identifié par son nom.
 Fixe le remplissage d'un objet GeoGebra identifié par son nom.
 Fixe l'épaisseur d'un objet GeoGebra identifié par son nom.
 Choisit aléatoirement de nouvelles couleurs pour le tracé et le remplissage.

Utilitaires

Blocs
 Convertit en degrés un angle exprimé en radians.
 Convertit en radians un angle exprimé en degrés.
 sin(radians(angle))
 cos(radians(angle))
 tan(radians(angle))
 Affiche une fenêtre contenant le message spécifié.
 Affiche une fenêtre contenant le message donné. Affiche aussi une zone de texte, comportant initialement la chaîne **parDéfaut**, mais que l'utilisateur peut modifier.



Permet d'insérer un commentaire dans notre programme.



Permet de déclarer une variable locale, à l'intérieur d'une fonction.

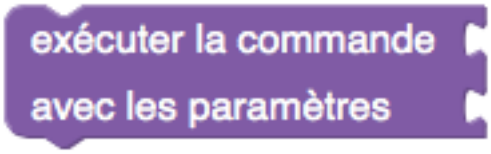
Appendice

Dans cet appendice, nous apporterons quelques précisions sur certains aspects du fonctionnement ***d'algoGGB***. Notons tout d'abord que ce programme ne manipule pas directement les objets *GeoGebra* : il travaille plutôt avec leurs ***noms***.

algoGGB propose en outre une façon aisée d'engendrer de nouveaux noms. Par exemple,

va retourner un nouveau nom constitué de la chaîne *Pt* suivie d'un nombre, par exemple *78*, choisi de telle sorte que le nom *Pt78* n'existe pas précédemment dans *GeoGebra*.

Il est important de noter que la création d'un nouveau nom n'implique pas la création d'un objet *GeoGebra* correspondant : pour cela, il faudra donner à *GeoGebra* une commande spécifique. Ceci se fera souvent à l'aide du bloc **commande**,



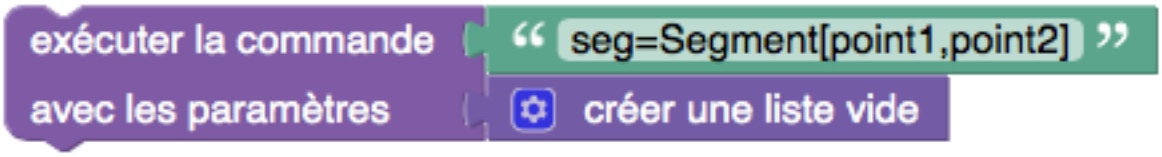
dont le fonctionnement est un peu subtil : prière de rester concentré !

Commençons à apprivoiser **ce bloc** dans un cas simple : si on peut utiliser une commande, par exemple « s=Segment[P,Q] », dans le champ de saisie de *GeoGebra*, alors on peut utiliser

dans nos programmes visuels ***algoGGB***. Mais, dans un tel cas, il faut avouer qu'un programme ***algoGGB*** n'a aucun avantage sur l'action directe. D'autant plus qu'il nous faudra utiliser les **versions anglaises des commandes** dans tous nos programmes ***algoGGB visuels***.

Mais si nous avons à tracer une multitude de segments, comme c'est le cas pour le triangle de Sierpinski, par exemple, nous serons amenés à gérer automatiquement les noms des divers segments, notamment en plaçant ces noms dans des variables.

Ainsi, si le nom du premier point est placé dans une variable *point1*, que le nom du second point se trouve dans la variable *point2*, et que le nom du segment qu'on veut créer se trouve dans une troisième variable *seg*, il serait incorrect d'utiliser la commande



puisque, par exemple, le nom du premier point n'est pas *point1* : il est contenu dans la variable *point1* (par exemple, il pourrait être *P237*). On peut résoudre ce problème en faisant appel aux capacités de manipulation des chaînes de caractères d'algoGGB. On obtient alors la commande

qui est correcte mais qui semble aussi inutilement compliquée. D'autant plus qu'il s'agit ici d'une commandes très simple : imaginez pour les commandes plus complexes !

algoGGB propose un mécanisme pour simplifier ce type de problème : dans la situation ci-dessus, ceci prend la forme

Voici comment interpréter cette nouvelle forme de **commande** : avant d'exécuter la commande **Segment**, ***algoGGB*** remplacera

- l'expression spéciale **@1** par la *valeur* de la variable **seg**
- l'expression spéciale **@2** par la *valeur* de la variable **point1**
- l'expression spéciale **@3** par la *valeur* de la variable **point2**.

On peut voir ce mécanisme à l'œuvre dans le second exemple (le triangle de Sierpinski) fourni avec ***algoGGB visuel***…