ExecuteMacro ou « ExecuteMakwél[1] » ?

Lorsque que je découvrais l'article de Patrice Debrabant (Mathématice n°36) sur la récursivité et CarMetal, j'avais déjà (tant bien que mal) travaillé sur la récursivité mais la **révélation,** pour moi, venait de la commande **ExecuteMacro** : cet outil qui permet de lancer une macro depuis un carScript. J'ai pu observer par la suite que le forum du logiciel en parlait depuis longtemps déjà mais cela m'avait totalement échappé (tant pis pour moi).

Cet outil est un raccourci formidable et épargne énormément de programmation(…) notamment pour des applications pédagogiques.

L'objet de cette brève est de vous faire part de mon expérience d'utilisateur de cette commande, des écueils que j'ai rencontrés et de la façon dont j'ai pu remédier à quelques uns.

Je n'ai pas trouvé dans le forum ou dans les tutoriels de CarMetal d'articles s'y référant aussi je souhaite partager mes "trouvailles", afin que d'autres ne s'y cognent pas autant que moi…

Préambule...

La lecture de cet article coïncidait avec un projet que m'avait proposé Alain Magen . Il s'agit d'utiliser la puissance des logiciels de géométrie dynamique pour aider quiconque à créer ou disposer d'une abaque de son choix. Le travail d'Alain est déjà colossal, je n'étais au préalable intervenu que pour l'inciter à travailler sur d'autres supports que son LGD fétiche : Cabri…Il s'est donc lancé (entre autres...) à la découverte de CarMetal mais les carScripts qu'il proposait manquaient pour le moins de concision et n'étaient pas, pour reprendre les définition de A Busser construits, mais dessinés… Je voulais donc l'aider à reprendre ce travail pour le rendre dynamique sous CarMetal.

Je n'ai pour l'instant travaillé que sur une abaque à points alignés et à support parallèles, mais elle m'a fait déjà beaucoup avancer et va servir de modèle au moins pour toutes celles qui admettent des supports rectilignes.

Le parcours de la construction fait l'objet du livre d'Alain aussi ne le décrirai-je pas, cela n'est pas l'objet de l'article ; sachez, qu'après la découverte de la commande ExecuteMacro, je souhaitais, pour répondre à ce projet, écrire des scripts qui appellent des macros beaucoup plus simples à construire géométriquement qu'une programmation directe . Le classeur joint "articleEM.zirs[2]" illustre cet article

Cela m'a permis de mettre en évidence quelques obstacles (…).
Les façons dont on peut les surmonter méritent je crois d'être diffusées…

1er obstacle...

Toutes les macros qui fonctionnent sous CarMetal ne fonctionnent pas forcément dans un script…

Cette règle est partiellement évoquée dans l'article de Patrice Debrabant je cite, en avant dernière page de l'article : « *Cette macro fonctionne. Malheureusement, elle ne réussit pas l'examen de passage du CaRScript ExecuteMacro, le script génère une erreur...* »...

1 Prononcer « Màkouèl » ; un nom qui serait peut-être plus approprié vu tous les croches-pattes qu'elle m'a tendus...
2 Les macros utilisées dans le script doivent obligatoirement être présentes dans la bibliothèque (dossier *mesmac*), si ce n'était le cas , il faut ouvrir à partir du classeur le fichier joint **article.mcr** qui contient ces macros ....

Mais quelle erreur et pourquoi ?

Ma première macro ne fonctionnait pas dans le script… Ce ne serait pas la seule…

Je ne comprenais pas pourquoi et ne trouvais pas de réponses dans mes lectures. Des indices cependant dans le forum : je remarquais que certains utilisateurs parlaient de macro-géométriques…

Et je subodorais (*les élèves aiment bien ce mot*...) : « ma macro contient en objets initiaux des paramètres, des valeurs…elle ne serait pas "géométrique !" »

Pourtant dans la fin de l'article, à l'initiative d'Yves Martin, l'auteur introduit un curseur en objet initial. C'était donc possible et je m'entêtais avec raison...

1ère règle

> Lorsque la macro fait intervenir des paramètres, curseurs, valeurs, en objets initiaux, ils doivent être désignés en premiers dans la liste, les objets géométriques, points , droites etc… doivent l'être en dernière partie.

Mes macros vont devoir aussi utiliser des fonctions… nouvel écueil… nouvelle règle...

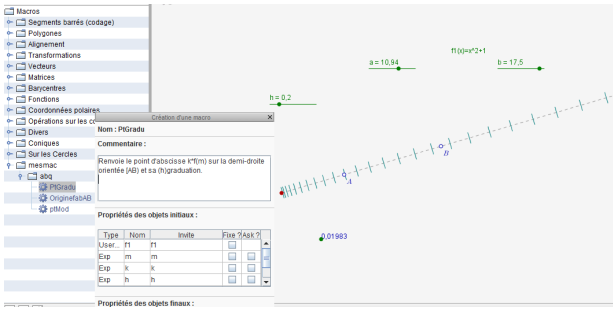
2ème règle.

> Lorsque la macro fait intervenir des expressions (fonctions) en objets initiaux, elles doivent être désignées en premiers, avant les valeurs et les objets géométriques .

Vous devez donc penser à construire la macro avec la typologie de variable dans cet ordre bien défini. (enfin pour moi cela marche ainsi et pas autrement…) :

① Fonctions
② Valeurs, curseurs
③ Objets géométriques.

Sur l'**exemple1** (1er onglet du classeur « articleEM » qui permet de graduer la droite (AB))



Sans trop entrer dans les détails le script gradue une demi-droite d'axe (AB) de sorte que A et B soient les représentants respectifs de $f_1(a)$ et $f_1(b)$ à un multiplicateur près. (je vous renvoie aux articles d'Alain sur les abaques)

Observez grâce à la fenêtre « propriété de la macro » **l'ordre** de désignation des objets initiaux.

Si je ne respecte pas cet ordre la macro fonctionne en tant que telle mais pas la commande ExcuteMacro (*d'où le titre de cet article*…).

Observez sur le fichier que la figure reste bien dynamique que vous modifiiez points, expression de la fonction ou curseur…

Observez enfin que toutes vos modifications sont effacées dès que vous annulez le script...

Je raconterai en annexe le script et les difficultés de liaison script-macro…

Troisième écueil…Les expressions en objets finaux..

Une macro qui ne renvoie qu'une expression en objet final va fonctionner mais va aussi renvoyer un

message d'erreur (« *Ze Hais les messages !!* ») :

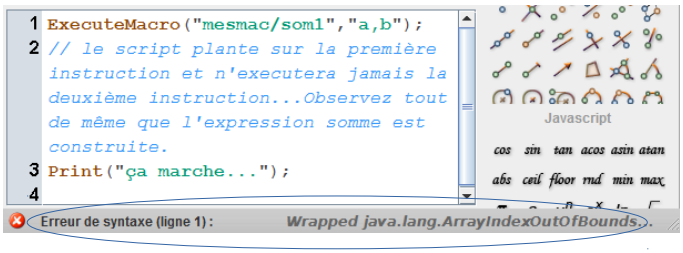
Wrapped java.lang.ArrayIndexOutOfBoundsException : 0

En fait la macro est exécutée et l'expression est construite mais le script s'arrête sur le message d'erreur…

Observez l'onglet **exemple2** du classeur.

Dans cet exemple tout simple, la macro **som1** renvoie la somme de deux curseurs a et b .

Le script « **Somme**1 » qui appelle la macro **som1** construit bien l'expression mais renvoie aussi le message d'erreur en bas de l'éditeur, le script est planté.



Cette erreur est propre à JavaScript. Les expressions ne sont pas des variables au sens JavaScript et elles ne seront pas stockées dans les listes Array(). En fait le script ne trouve rien à stoker dans la liste qu'ExecuteMacro lui fait l'ouvrir…

J'ai trouvé un moyen de biaiser (« *Je ne fais que biaiser* » chante le poète…), il s'agit de construire avec la macro un objet géométrique qui ne servira à rien à priori mais qui va s'avérer fondamental.

Si "s" est l'expression retournée on peut construire par exemple le point S(s,s)… et placer cet objet géométrique en deuxième objet final de la macro…

La commande ExecuteMacro va alors enregistrer ce point (et seulement ce point) dans l'Array ouvert...Le message d'erreur n'apparaît plus et le script peut se poursuivre.

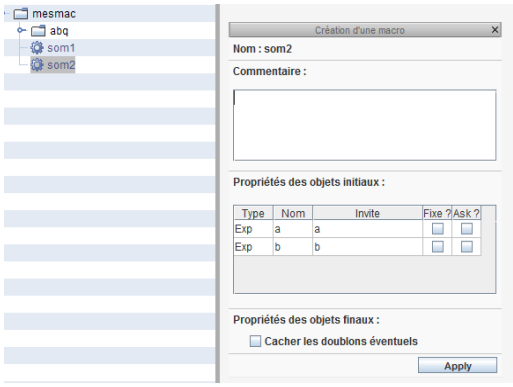
Observez pour cela le script « **Somme2** »…

3ème règle.

> Les expressions retournées par une macro ne sont pas enregistrées en tant que variables par la commande ExcecuteMacro et une macro qui retourne une seule expression génère une erreur lors de l'exécution du script qui retourne :
>
> ***Wrapped java.lang.ArrayIndexOutOfBoundsException : 0***
>
> **Remède** : Si une macro doit retourner une expression seule et doit être appelée dans un script l'expression, dans la construction de la macro, devra être retournée accompagnée d'un autre objet **géométrique** final.

Quelques remarques...

- J'insiste sur ce point : lorsque la macro est construite dans la perspective d'être appelée dans un carscript, il est intéressant de penser **dès sa construction** à **l'état** des objets finaux (l'objet final caché ou super caché sera construit comme tel par le script) cela permet d'éviter les instructions de ménage (mises en forme.) qui alourdissent le script…Dans l'exemple 2, le point construit avec Somme2 est apparent afin que vous puissiez le repérer mais il aurait pu être construit caché dès l'élaboration de la macro…



- Il est aussi intéressant d'observer que les propriétés de la macro ne reflètent pas les différences entre les deux macros

som1 et som2 puisqu'elles ne font pas apparaître les objets finaux. Ainsi un lecteur non averti pourrait peut-être s'y perdre...

- Je reviens à présent sur l'intérêt du point S(s,s) qui accompagne la création d'une macro expression dont un l'objet final est s. L'expression est bien créée dans la figure, mais elle n'existe sous aucune forme en tant que variable dans le programme. Il est possible de la rappeler à l'aide de la commande GetExpressionValue mais dans ce cas la variable rappelée est **figée** et la figure perd son dynamisme autour de s.
  Par contre le point créé sous ExecuteMacro reste lui dynamique et l'expression s **existe** (est reconnue dans le script) en tant qu'abscisse de ce point, elle est donc utilisable dans le programme sous forme construite (dynamique) et non plus figée.

Dernier écueil : les axes…

4ème règle.

> Une macro utilisant les axes en tant qu'objet initiaux ne peut pas être appelée par l'ExecuteMacro.

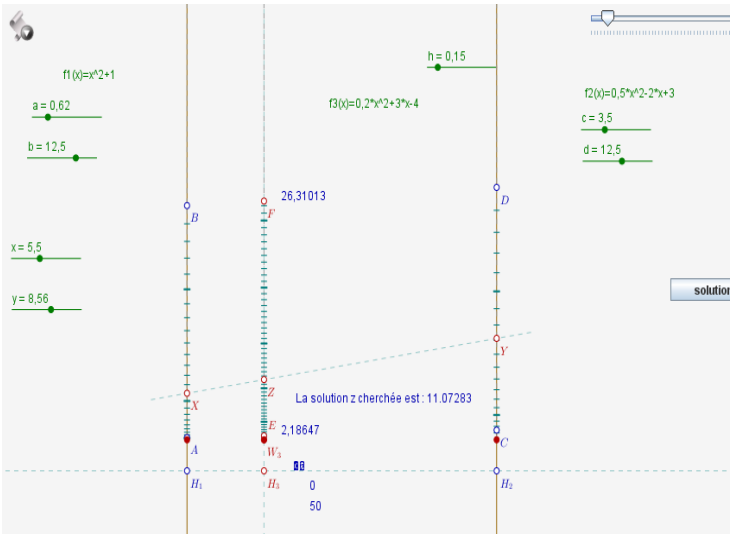
Cette règle, je le souhaite pourra être prise en défaut mais pour ma part je n'ai jamais pu, malgré moultes tentatives, contrer ce constat.

A chaque fois j'ai du rebâtir la macro concernée pour qu'elle n'utilise pas les axes. Et cela **même** si les axes sont définis comme **objets initiaux fixes** dans les propriétés de la macro…

Voilà les règles d'utilisation que j'ai pu mettre en évidence et que j'observe dès que je construis une macro dans l'optique de l'appeler dans un script.

Peut être seront-elles vite obsolètes avec la venue de prochaines versions du logiciel mais en attendant…

En conclusion



Le 3ème onglet du classeur propose une abaque qui permet de résoudre une équation du type $f_1(x)+f_2(y)=f_3(z)$.

Vous pouvez observer le dynamisme de la figure en regard des paramètres mobiles… Les trois fonctions peuvent aussi être modifiées.

Tout n'est pas encore parfait… mais l'abaque est déjà bien utilisable.

Sur cette figure il est intéressant d'observer que si le script « solution » (...*Tout en un*… ou bouton) génère d'une seule étape toute la figure, il est parfois fort utile de construire la même figure en plusieurs scripts. On peut annuler la dernière construction sans avoir à tout reprendre.

Lorsque qu'on annule un script toutes les modifications qui ont été faites sur la figure depuis son lancement sont effacées aussi. Dès lors, si vous lancez trois scripts l'annulation du troisième n'effacera pas ce que vous avez pu modifier sur la figure entre les lancés du deuxième et troisième. Certains scripts peuvent engager beaucoup de calculs et de temps, il est bon aussi d'éviter de les relancer si cela s'avère inutile...

Annexe (ou compléments…)

Dans l'exemple 1 (comme pour les graduations de l'abaque), la fonction f1 et les paramètres a et b étant donnés dans la figure, le script gradue la demi-droite d'axe (AB) de sorte que A et B soient les représentants respectifs de $f_1(a)$ et $f_1(b)$ à un multiplicateur près.

Il s'agit donc de calculer quel doit être ce multiplicateur qu'Alain appelle module ($k=\frac{AB}{f(b)-f(a)}$) puis de déterminer l'origine W de la demi-droite sur (AB) et enfin de procéder à la graduation.

Ici j'utilise une macro qui dessine une graduation sur un point donné. Il s'agit donc d'utiliser cette graduation dans une boucle pour un compteur allant de 1 à *n* (*n :* valeur demandée à l'utilisateur sur l'exemple). Là la variable qui désigne le compteur, i dans le script, est une variable JavaScript et elle n'existe pas pour la figure, je ne peux alors pas utiliser cette variable comme paramètre pour la macro. Il y a ici une difficulté de communication entre les deux outils du logiciel.

Tous les **paramètres** appelés par la macro doivent être des **objets de la figure**

Pour y remédier je construis l'expression m de valeur i dans la figure et c'est cette expression m et donc la valeur de i que je récupère dans la macro…

```
15        for (i=1; i<n; i=i+1){
16           var t =new Array();
17           m=Expression(i,0,-i);SetHide("_m",true);
18           t =
   ExecuteMacro("mesmac/abq/PtGradu",f+","+m+","+k+","+h+","+w+","+B);
19        }
```

Bien entendu ce « bidouillage » alourdit et ralentit la construction lors de l'exécution du script et il faut espérer que nous trouverons de meilleures réponses.

Néanmoins ceci illustre bien les croc-en-jambes sur lesquels nous risquons de trébucher en utilisant cette commande qui demeure cependant un superbe outil. Il faut juste prendre le temps de l'amadouer un peu...

Merci aux concepteurs, René Grothmann, Eric Hakenholz et autres ouvriers de ce puissant logiciel **libre**.