M1 Imageurs, Poitiers 2017: Introduction aux macros ImageJ

Fabrice Cordelières, Bordeaux Imaging Centre

La problématique

La plate-forme d'imagerie de Poitiers dispose d'un outil matériel et logiciel permettant de réaliser une image de phase. A mesure qu'elles sont capturées, le logiciel d'acquisition peut envoyer automatiquement les images vers ImageJ, un gratuiciel permettant le traitement et l'analyse d'images. Cette fonctionnalité vient pallier le défaut d'outils d'analyse 'live' du logiciel d'acquisition.

Votre mission:

Concevoir une macro d'analyse sous ImageJ permettant: 1. D'assembler les images individuelles en une pile d'images, à mesure qu'elles sont poussées vers ImageJ; 2. Mesurer l'intensité moyenne de chaque régions d'intérêt préalablements tracées et enregistrées dans le ROI Manager (gestionnaire de régions d'intérêt); 3. Tracer en 'live' l'intensité au cours du temps.

Les outils à disposition

Pour vous aider dans cette tâche, vous diposez des outils suivants:

- ImageJ ou Fiji: à télécharger et installer;
- Un greffon (plugin), LiveFeed_Simulator.jar, simulant l'arrivée de nouvelles images à intervalel de temps régulier;
- Des images test, permettant au greffon de fonctionner;
- Une page de référence sur les bases du language macros;
- Une page de référence sur les fonctions macros.

Les bases du language macro ImageJ

L'outil de base: le macro recorder

Le macro recorder permet d'enregister (preseque) toutes les opérations réalisées par l'utilisateur. Il suffit de le lancer pour voir se créer une série d'instructions permettant de reproduire le protocole de traitement et d'analyse sur une autre image. Il se trouve dans le menu `Plugins>Macros>Record` et fonctionne de la même manière sous ImageJ et Fiji.

Exercice *A l'aide du macro recorder, enregistrer les instructions correspondant aux actions suivantes:* * Ouvrir une première image; * Renommer l'image en "Stack" (`Image>Rename...`) * Ouvrir une seconde image; * Copier le contenu de toute l'image (`Edit>Copy`); * Fermer la seconde image; * Activer la première image; * Ajouter une "coupe pour constituer une "pile" (`Image>Stacks>Add Slice`); * Coller le contenu de la seconde image dans cette image vide (`Edit>Paste`).

Le contenu du macro recorder ressemble à ce qui suit:

```
java open("/Imageurs 2017/sequence avec FLAT/EXPORT PHA SGL 02-11-2017 15_15_15-81480.tif"); rename("Stack"); open("/Imageurs 2017/seque
```

Structure des fonctions: une absence de consensus (mais on fait avec !)

En observant le contenu du Recorder, on constate que plusieurs structures de commandes co-existent: * `run("Nom_De_Commande", "Arguments")` : mot-clé run prenant 2 arguments 1-le nom de la fonction, 2-une chaine de caractères contenant les arguments, séparés par des espaces; * `nomDeCommande("Arguments")` : nom de la fonction, suivi d'une chaine de caractères contenant les arguments.

Ce ne sont que 2 des formes de commandes macros: il n'y a pas de consensus sur la structure des fonctions. La raison est historique: les fonctions en "run" correspondent aux premières fonctions implémentées à l'origine du logiciel; les fonctions plus récentes dont la syntaxe est plus proche d'un pseudo-java.

Trois types de comportements co-existent: * **Les méthodes**: ce sont des fonctions qui exécutent une opération ex: redimensionner une image, ajouter une coupe à une pile etc; * **Les fonctions**: elles renvoient un résultat qui peut être stocké dans une variable; * **Type intermédiaire**: c'est le cas par exemple de la fonction `getStatistics(area, mean, min, max, std, histogram)` qui stocke dans les variables fournies en arguments les informations demandées.

NB: Tout comme en Java, le `;` est obligatoirement utilisé pour marquer la fin d'une ligne d'instruction.

Editer une séquence et la sauvegarder: .txt ou .ijm ?

Bien que permettant l'édition des instructions, le Macro Recorder n'est pas l'éditeur à proprement parler des macros ImageJ. Pour le faire apparaitre, il suffit de cliquer sur le bouton `Create` .

Un éditeur plus complet s'affiche, différent suivant qu'on utilise ImageJ our Fiji. Sous ImageJ, l'éditeur est un simple éditeur de texte, sans coloration syntaxique. En revanche, il

embarque quelques fonctionnalités interessantes: * Un outil de déboggage, accessible via le menu `Debug`. Il permet notamment l'exécution pas-à-pas de la macro. * le `Function Finder`, accessible via le menu `Macros>Functions Finder...`: il permet de visualiser et recherche l'ensemble des fonctions macros. Le bouton de bas de fenêtre `Help` renvoie sur vers la section correspondante de la page macro function du site d'ImageJ.

Sous Fiji, l'éditeur de macros offre une coloration syntaxique. En revanche, le `Function Finder` n'est pas disponible. Les outils de deboggage/d'exécution pas-à-pas ne sont pas disponibles. ***Une astuce permet néanmoins de faire apparaitre l'éditeur de macros ImageJ sous Fiji***: il suffit pour de sauvegarder le fichier macro sans extension.

Justement, quelle extension pour nos fichiers macros ? Deux conventions sont possibles: au final, le fichier est un simple fichier texte. Peu importe, on pourra utiliser au choix txt ou ijm (Image J Macro).

Exécuter une macro ou la stopper

Sous ImageJ, pour exécuter une macro, depuis la fenêtre de l'éditeur, aller dans le menu `Macros>Run Macro` (racccourci clavier `CTRL ou CMD+R`). Pour en stopper l'exécution, presser la touche `Esc`.

Pour exécuter une macro sous Fiji, ustiliser le bouton `Run` en bas de la fenêtre de l'éditeur ou le menu `Macros>Run Macro` (racccourci clavier `CTRL ou CMD+R`). Pour en stopper l'exécution, presser la touche `Esc` ou le bouton `Kill` en bas de la fenêtre de l'éditeur.

Retrouver ses marques avec ce "nouveau" language:

A l'évidence, pour des tâches simples, le `Macro Recorder` peut suffire. En revanche, dès lors qu'il faudra modifier dynamiquement un paramètre, boucler un processus sur un ensemble d'éléments ou adapter le comportement de la macro en fonction d'une entrée, il drviendra nécessaire de connaitre la syntaxe d'un ensemble d'éléments de base que sont les variables, les boucles et les structures de choix. Comme dans tout langage de programmation, il sera également important de documenter son code: il est nécessaire de connaitre les balises à utiliser. Ces éléments vont à présent être détaillés.

Commenter son code: quelles balises ?

Les balises de commentaires sont identiques aux balises utilisées en Java: * **Les commentaires courts**: ils sont délimités par une seule balise, le `//` ; * **Les commentaires longs**: la balise ouvrante est `/*`, la balise fermante est `*/`.

Les variables: types, opérateurs et portée

Types de variables

Dans le language Macros ImageJ, les variables ne sont pas typées. En revanche, trois catégories de variables sont utilisables:

- **Les variables simples**: elles sont déclarées par leur nom et l'attribution d'une valeur se fait au moyen du signe égal. Les guillements permettent de définir la nature "chaine de caractère" du contenu. ex: `maVariable=3;` ou `monAutreVariable="chaine de caractères";` ;
- **Les variables de type tableau**: elles sont déclarées par leur nom et l'attribution (`=`) suivi du mot-clé `newArray()`. Cette instruction peut prendre trois types d'arguments (voir ci-après). **Attention**: le language Macros ImageJ ne permet pas de créer de tableaux multi-dimensionnels.
  - `monTableau=newArray();` : le tableau est initialisé mais n'a pas de dimension;
  - `monTableau=newArray(n);` : le tableau est initialisé sans contenu mais à une dimension de "n" cases;
  - `monTableau=newArray(1, 2, 3);` : le tableau est initialisé et dimensionné en fonction du contenu. Ses cases sont initialisée telles que `monTableau[0]` contient la valeur 1, `monTableau[1]` contient la valeur 2 etc. On peuit déterminer la dimension du tableau en ayant recours à l'instruction `monTableau.length`.
- **La liste de couples clé-valeur**: On peut utiliser une unique liste de clés-valeurs. Voici une liste des fonctions les plus utiles:
  - `List.clear()` : efface le contenu de la liste actuelle;
  - `List.size` : renvoie le nombre d'éléments dans la liste actuelle;
  - `List.set(cle, valeur)` : ajoute ou modifie la paire clé-valeur dans la liste;
  - `List.get(cle)` : renvoie la valeur associée à la clé en tant que chaine de caractères, ou une chaine vide si la clé n'existe pas dans la liste;
  - `List.getValue(cle)` : renvoie la valeur **numérique** associée à la clé. Si elle est absente de la liste ou n'est pas une valeur numérique, l'exécution de la macro est stoppée.

Opérateurs

Les opérateurs applicables aux variables individuelles sont relativement classiques et résumées dans le tableau suivant.

Operateur	Priorité des opérateurs	Description
++	1	Pré ou post increment
--	1	Pré ou post decrement
-	1	Soustraction bit à bit
!	1	Complementaire du booléen
~	1	Bit complémentaire
*	2	Multiplication
/	2	Division
%	2	Reste entier de la division
&	2	Et logique (AND, opération sur les bits)
I	2	Ou logique bitwise (OR, opération sur les bits)
^	2	Ou exclusif logique (XOR, opération sur les bits)
<<, >>	2	Rotation de bit (vers la gauche ou la droite)
+	3	Addition arithmétique ou concaténation de chaines de caractères
-	3	Soustration arithmétique
<, <=	4	Comparaisons: plus petit que, plus petit ou égal à
>, >=	4	Comparaisons: plus grand que, plus grand ou égal à
==, !=	4	Comparaisons: égal à, différent de
&&	5	Et logique (AND, opération sur les booléens)
II	5	Ou logique bitwise (OR, opération sur les booléens)
=	6	Attribution
+=, -=, *=, /=	6	Attribution combinée à une opération

NB: certaines opérations entre variables de nature différentes sont possibles. La plupart du temps, elles aboutissent à un transtypage implicite des contenus des varaibles en chaines de caractères au moment d'effectuer l'opération.

Portée des variables

Le language Macros ImageJ permet de créer ses propres groupes fonctionnels. Cette possibilité permet de mieux organiser son code et d'en réutiliser certaines parties dans plusieurs macros sans pour autant avoir à tout re-coder. Ces groupes peuvent nécessiter la déclaration de variables en leur coeur: leur contenu ne sera alors pas disponible à l'extérieur de la fonction à moins qu'un retour explicite soit implémenté. De même, une variable déclarée en dehors de la fonction définie par l'utilisateur n'y sera pas directement utilisable: il faudra alors passer son contenu en argument, au moment de l'appel.

Une autre stratégie est envisageable: modifier la portée des variables. L'utilisation du mot-clé `var` devant une variable placée dans le corps de la macro (plutôt en en-tête, pour plus de lisibilité) permet de la transformer en **variable globale**, accessible depuis n'importe quel point du code.

Les boucles

Boucle définie à priori: `for`

Sa structure est identique à celle utilisée en Java, à la différence qu'il n'est pas possible d'utiliser un itérateur sur une liste:
`java for(initialisation; condition de poursuite; itération) { //Instructions }`

Boucle indéfinie à priori: `while`

Sa structure est identique à celle utilisée en Java: `java while(condition) { //Instructions }`

Boucle indéfinie à postériori: `do/while`

Cette boucle est exécutée au moins une fois, l'évaluation de la condition s'effectuant à la fin du bloc. Il se déclare comme suit:
`java do { //Instructions } while (condition);`

L'unique structure de choix disponible: `if/else - else if`

Une seule structure de choix est disponible: la structure if/then/else - else if:

`java if(condition1) { //Instructions 1 }else{ //Instructions 2 }else if(condition2){ //Instructions 3 }` **NB**: Inutile d'essayer, la structure `switch/case` n'est malheureusement pas implémentée...

Pour démarrer l'implémentation

Etape 1: Assembler les images en une pile

But

Créer une macro permettant de monitorer l'arrivée d'images dans ImageJ: * Pour éviter que la macro ne tourne à l'infini, instaurer 2 paramètres: * Un **délai**: c'est le temps à attendre entre l'arrivée de deux images successives (valeurs par défaut: 1000msec); * Un **time out**: sous la forme d'un nombre entier, il indique le nombre d'images maximum que l'on peut manquer avant de considérer que le flux d'images a été interrompu (valeur par défaut: 5). * Lorsque la première image arrive, la renommer "Stack"; * Lorsque les images suivantes arrivent, copier leur contenu et l'ajouter en tant que nouvelle coupe de la pile "Stack"; * Fermer la dernière image; * Attendre l'arrivée de l'image suivante.

Afin d'alimentation ImageJ avec un flux d'images individuelles, on utilisera le greffon `LiveFeed Simulator,Multiple Frames...` .

Quelques indications pour le développement

- La macro doit attendre **tant qu'** aucune image n'est présente;
- Il existe une **variable système** qui porte le nombre d'images actuellement ouvertes dans ImageJ: trouvez laquelle;
- La première image doit être renommé en "Stack": on a déjà enregistré cette instruction;
- Depuis le language Macros, il existe deux manières d'activer une image:
  - En l'appelant par son titre: `selectWindow("Nom de l'image");` (voir le site des fonctions macros pour sa description);
  - En l'appelant par son identifiant unique: `selectImage(identifiant);` . L'identifiant unique peut-être un entier négatif attribué à chaque image ouverte pendant la session active d'ImageJ: la première image a pour id -1, la seconde image ouverte -2 etc. Si un entier positif est fourni, ImageJ activera alors la id-ème image de la liste des images visible dans le menu `Window` (voir le site des fonctions macros pour plus de détails).
- Il existe une fonction permettant de **mettre en pause l'exécution** d'un macro pendant un certain nombre de millisecondes.
- Bien penser à tenir un **compte du nombre d'images manquées** et à ré-initialiser le compteur chaque fois qu'une nouvelle image a été détectée.
- Lorsque plusieurs images sont chargées, elles sont affichées en cascade: la dernière image recouvre la première. Le but étant de bien visualiser la pile, il serait bon de **la déplacer de sorte à ce qu'elle soit visible** en permanence: une fonction existe pour réaliser cette opération.

Implémentation

AVANT DE REGARDER LA SOLUTION, ESSAYEZ D'IMPLEMENTER LA MACRO PAR VOUS MEME !!!

Algorithme

C'est la première étape de la conception d'une macro. Elle consiste à décrire étape par étape ce que le code doit réaliser. Une bonne pratique consiste à écrire l'algorithme sous forme de commentaires dans l'éditeur de macro, puis à faire suivre chaque commentaire du code correspondant.

```
Déclarer une variable "delai" et l'initialiser à 1000
Déclarer une variable "timeOut" et l'initialiser à 5

Tant qu'aucune image n'est ouverte, attendre le délai
Renommer l'image en "Stack"

Déplacer l'image vers le bord droit de l'écran

Initialiser un compteur d'images manquées, nTimeOut, à 0
Tant que nTimeOut est inférieur à timeOut faire:
    Si le nombre d'images ouvertes est supérieur à 1, faire:
        Sélectionner la dernière image ouverte
        Copier son contenu
        Fermer l'image

        Sélectionner la pile
        Ajouter une coupe
        Coller le contenu

        Réinitialiser le compteur d'images manquées, nTimeOut, à 0
    Fin Si

    Attendre le délai
    Incrémenter nTimeOut
Fin Tant que
```

Le code est relativement simple à implémenter. Il suffit d'avoir une connaissance de base des macros, et d'enregistrer une partie des actions, et de chercher les fonctions non

enregistrable sur le le site des fonctions macros.

Une implémentation possible

```
delai=1000;
timeOut=5;


while(nImages==0) wait(delai);
rename("Stack");

//Déplace le stack pour éviter la superposition avec les images arrivant
setLocation(screenWidth-getWidth, 0);

nTimeOut=0;
while(nTimeOut<timeOut){
    if(nImages>1){
        selectImage(nImages); //Sélectionner la dernière image ouverte
        run("Copy");
        close();

        selectWindow("Stack");
        run("Add Slice");
        run("Paste");

        nTimeOut=0;
    }
    wait(delai);
    nTimeOut++;
}
```

Etape 1 bis: organiser le code de l'étape 1 en un bloc fonctionnel

A l'exception des deux premières lignes, le code précédent constitue un bloc qui peut se suffire à lui même. Le language Macros permet de créer des fonctione personalisées: il suffit d'utiliser le mot-clé `function` suivi d'un nom personalisé, d'une paire de parenthèses (ouvrante et fermante) entre lesquelles on peut insérer des paramètres sous forme de varaibles séparées par des virgules. Le code à exécuter à l'appel de la fonction est inclus entre accolades. L'appel de la fonction se fait alors depuis n'importe quelle partie de la macro (avant ou après la déclaration du bloc fonctionnel) en utilisant son nom, suivi des valeurs de paramètres à utiliser.

En appliquant cette stratégie au code précédent, on obtient:

```
delai=1000;
timeOut=5;

formerLaPile(delai, timeOut);

function formerLaPile(delai, timeOut){
    while(nImages==0) wait(delai);
    rename("Stack");

    //Déplace le stack pour éviter la superposition avec les images arrivant
    setLocation(screenWidth-getWidth, 0);

    nTimeOut=0;
    while(nTimeOut<timeOut){
        if(nImages>1){
            selectImage(nImages); //Sélectionner la dernière image ouverte
            run("Copy");
            close();

            selectWindow("Stack");
            run("Add Slice");
            run("Paste");

            nTimeOut=0;
        }
        wait(delai);
        nTimeOut++;
    }
}
```

Etape 2: Mesurer l'intensité dans les régions d'intérêt

But

L'utilisateur aura au préalable tracé et ajouté les régions d'intérêt à monitorer dans le ROI Manager. Pour chaque point de temps, l'utilisateur souhaite rappatrier l'intensité moyenne de chaque région. Que le ROI Manager dispose ou non de régions d'intérêt, l'intensité moyenne globale de l'image devra également être déterminée.

Quelques indications pour le développement

- ImageJ ne permet la manipulation que d'une seule région d'intérêt à la fois: le ROI Manager permet de les activer tour à tour. Certaines des fonctions relatives au ROI Manager sont enregistrables, d'autres devront être trouvées sur le site les répertoriant. Il faudra notamment savoir comment:
    - Dénombrer les ROIs;
    - Activer une ROI spécifique;
- On doit être en mesure d'éliminer une ROI de l'image.
- Il existe une fonction permettant de rappatrier des statistiques dans une ROI s'il en existe une sur l'image, ou de l'image entière si aucune n'est activée.
- Les données doivent être stockées dans une structure appropriée, pourquoi pas un "tableau de résultats".
- Pour le tracé des graphiques, il sera utile d'avoir un relevé du point de temps courant.

Implémentation

AVANT DE REGARDER LA SOLUTION, ESSAYEZ D'IMPLEMENTER LA MACRO PAR VOUS MEME !!!

Algorithme

La seule difficulté réside dans l'utilisation des tableaux de résultats. Voici donc quelques indications: * Un tableau de résultats n'a pas besoin d'être instancié pour qu'on y pousse des données: il suffit d'utiliser la commande `setResult("Nom de la colonne", ligne, valeur)` : la colonne portant le titre fourni sera créée et la valeur poussée sur la ligne dont l'index est fourni (les index démarrent à 0). Si on pousse la valeur sur une ligne dont l'index est supérieur au nombre de lignes actuel (`nResults`), alors une nouvelle ligne est créée. * Après y avoir poussé des valeurs, une table de résultats peut être actualisée au moyen de l'instruction `updateResults()` ; * Une `Results Table` existante peut être effacée au moyen de l'entrée de menu `Analyse>Clear Results` , commande qui est enregistrable.

```
Sélectionner l'image "Stack"
Eliminer toute ROI de l'image
Rappatrier les statistiques
Pousser le numéro du point de temps courant dans le tableau de résultats, colonne "Timepoint": il s'agit en fait du nombre de lignes dans le tableau
Pousser la moyenne d'intensité dans le tableau de résultats, colonne "Full_Image"

Pour i allant de 1 à la taille du tableau "quantif"
    Sélectionner l'image "Stack"
    Activer la i-ème ROI-1
    Rappatrier les statistiques
    Pousser la moyenne d'intensité dans le tableau de résultats, colonne "ROI_(i+1)"
Fin pour

Mettre à jour l'affichage du tableau de résultats
```

Le code est relativement simple à implémenter. Il suffit d'avoir une connaissance de base des macros, et d'enregistrer une partie des actions, et de chercher les fonctions non enregistrable sur le le site des fonctions macros.

Une implémentation possible

```
run("Clear Results");
quantifier();

function quantifier(){
    //Stocke les infos sur toute l'image
    selectWindow("Stack");
    run("Select None");
    getStatistics(area, mean, min, max, std, histogram);
    setResult("Timepoint", nResults, nResults);
    setResult("Full_Image", nResults-1, mean);

    //Stocke les infos pour chaque région
    for(i=0; i<roiManager("Count"); i++){
        selectWindow("Stack");
        roiManager("Select", i);
        getStatistics(area, mean, min, max, std, histogram);
        setResult("ROI_"+(i+1), nResults-1, mean);
    }
    updateResults();
}
```

Etape 2 bis: Les problèmes commencent...

Constat d'échec... et solution

Ouvrez une pile d'images, dessinez quelques ROI que vous ajoutez au ROI Manager puis lancez la macro précédente: que ce passe-t-il ? Maintenant, déplacez vous dans la pile au moyen de l'ascenseur du bas puis renouvellez l'opération: que constatez-vous ?

Lors de l'ajout de ROI, ImageJ a gardé une trace de la coupe sur laquelle elle a été dessinnée. Si l'on veut pouvoir quantifier le signal au cours du temps, il faut pouvoir s'affranchir de cette information pour que la coupe active ne change pas à chaque rappel des ROIs.

On va donc devoir implémenter une fonction de mise à jour des ROIs. On en profitera pour attribuer une couleur et un nom normalisé à chaque ROI, une indication visuelle bien pratique qui nous servira plus tard à rapprocher une trace du graphique à une ROI sur l'image.

Implémentation

AVANT DE REGARDER LA SOLUTION, ESSAYEZ D'IMPLEMENTER LA MACRO PAR VOUS MEME !!!

Algorithme

```
Déclarer une variable "colors" de type tableau contenant les couleurs à utiliser pour afficher les ROIs

Désélectionner toutes le ROIs dans le ROI Manager
Oter les informations de position relatives au canal
Oter les informations de position relatives à la profondeur dans la pile
Oter les informations de position relatives au temps

Pour toutes les ROIs du ROI ROI Manager
    Sélectionner la i-ème ROI
    Créer une variable "color" et y stocker le résultat du reste entier de la division i/la longeur du tableau de couleurs
    Assigner à la i-ème ROI la couleur de tracé "color"
    Assigner à la i-ème ROI le nom "ROI_"+nombre démarrant à 1
Fin pour
```

La seule difficulté réside dans les index à utiliser pour activer tour à tour les ROIs, dont la numérotation commence à 0, et le stockage des données dans le tableau (numérotation des cases démarrant à 0 également), sachant que la première case est occupée par la mesure réalisée sur l'image entière.

Un autre point qui pourrait poser problème est en lien avec les couleurs à utiliser pour afficher le contour de chaque ROI: on pourra définir un tableau de valeurs, mais se posera le problème de la situation où le nombre de ROI est supérieur au nombre de couleurs disponibles. Il faudra alors utiliser l'opérateur modulo `%` pour obtenir le reste entier de la division réalisée entre l'index de boucle et la longueur du tableau.

Une fois le code écrit, il suffira de l'encapsuler pour créer une fonction, en veillant à sortir du corps de la fonction les variables qui pourraient être définie comme globales.

Une implémentation possible

```
var colors=newArray("blue", "green", "cyan", "yellow", "magenta");

cleanUpROIs();

function cleanUpROIs(){
    //Si des ROIs ont été sélectionnées, il faut s'assurer que la position n'est pas enregistrée
    roiManager("Deselect");
    roiManager("Remove Channel Info");
    roiManager("Remove Slice Info");
    roiManager("Remove Frame Info");

    for(i=0; i<roiManager("Count"); i++){
        roiManager("Select", i);
        colorNb=i%colors.length;
        roiManager("Set Color", colors[colorNb]);
    roiManager("Rename", "ROI_"+(i+1));
    }
}
```

Pourquoi une variable globale ? 1. Parce que c'était l'occasion d'utiliser une modification de la portée des variables; 2. Parce que cette variable sera utile aussi pour le tracé des graphiques.

Etape 3: Prenons le temps d'intégrer l'ensemble des fonctions

But

Après avoir codé les trois premières fonctions, il est temps de les assembler en une seule macro: c'est ce que nous allons faire à présent.

Quelques indications pour le développement

Aucune information n'est réellement requise à ce stade. Il va falloir penser à la séquence de traitement et appeler les différentes fonctions, dans le bon ordre. On gardera simplement à l'esprit les points suivants: 1. Une fonction définie par l'utilisateur peut être appellée... depuis une autre fonction définie par l'utilisateur; 3. On peut d'ores et déjà préparer les éléments permettant à l'utilisateur d'interpréter les résultats: afficher toutes les ROIs sur l'image en cours d'analyse est l'un des éléments participant à l'ergonomie générale de l'outil que nous développons. 4. On profitera de la restructuration du code pour veiller à le commenter !

Implémentation

AVANT DE REGARDER LA SOLUTION, ESSAYEZ D'IMPLEMENTER LA MACRO PAR VOUS MEME !!!

Une implémentation possible

```
//------------------------------------------------------------------
// VARIABLES SIMPLES
//------------------------------------------------------------------
delai=1000;
timeOut=5;

//------------------------------------------------------------------
// VARIABLES GLOBALES
//------------------------------------------------------------------
var colors=newArray("blue", "green", "cyan", "yellow", "magenta");

run("Clear Results");

cleanUpROIs();

formerLaPile(delai, timeOut);

//------------------------------------------------------------------
// Si des ROIs ont été sélectionnées, il faut s'assurer que
// la position sur la pile n'est pas enregistrée
//------------------------------------------------------------------
function cleanUpROIs(){
    //Si des ROIs ont été sélectionnées, il faut s'assurer que la position n'est pas enregistrée
    roiManager("Deselect");
    roiManager("Remove Channel Info");
    roiManager("Remove Slice Info");
    roiManager("Remove Frame Info");

    for(i=0; i<roiManager("Count"); i++){
        roiManager("Select", i);
        colorNb=i%colors.length;
        roiManager("Set Color", colors[colorNb]);
    roiManager("Rename", "ROI_"+(i+1));
    }
}

//------------------------------------------------------------------
// Création dynamique de la pile
// Quantification au cours du temps
//------------------------------------------------------------------
function formerLaPile(delai, timeOut){
    while(nImages==0) wait(delai);
    rename("Stack");
    quantifier(); //C'est le premier tour: on initie mesures
    roiManager("Show All"); //Active l'affichage de l'ensemble des ROIs

    //Déplace le stack pour éviter la superposition avec les images arrivant
    setLocation(screenWidth-getWidth, 0);

    nTimeOut=0;
    while(nTimeOut<timeOut){
        if(nImages>1){
            selectImage(nImages); //Sélectionner la dernière image ouverte
            run("Copy");
            close();

            selectWindow("Stack");
            run("Add Slice");
            run("Paste");

```

```
            quantifier();

            nTimeOut=0;
        }
        wait(delai);
        nTimeOut++;
    }
}

//------------------------------------------------------------------
// Quantification de l'intensité moyenne
// sur l'image entière et dans chaque ROI
//------------------------------------------------------------------
function quantifier(){
    //Stocke les infos sur toute l'image
    selectWindow("Stack");
    run("Select None");
    getStatistics(area, mean, min, max, std, histogram);
    setResult("Timepoint", nResults, nResults);
    setResult("Full_Image", nResults-1, mean);

    //Stocke les infos pour chaque région
    for(i=0; i<roiManager("Count"); i++){
        selectWindow("Stack");
        roiManager("Select", i);
        getStatistics(area, mean, min, max, std, histogram);
        setResult("ROI_"+(i+1), nResults-1, mean);
    }
    updateResults();
}
```

Etape 4: Créer et mettre à jour le graphique

But

Nous attaquons la partie finale de la macro: représenter graphiquement les données à mesure que les images sont acquises et transférées vers ImageJ. Nous allons nous heurter à quelques difficultés: * Comment faire en sorte que le graphique soit tracé ? * Comment le mettre à jour ? * Comment faire si la largeur du graphique ne permet pas d'afficher toutes les données ? * Comment mettre à l'échelle l'axe des intensités ?

Nous devrons répondre à toutes ces questions en plusieurs étapes, en adoptant la méthodologie suivante: 1. Etudier les possibilités de tracer un graphique avec ImageJ; 2. Ne pas négliger les interactions possibles avec l'utilisateur; 3. Ne pas négliger l'ergonomie de l'outil que nous finaliserons.

Quelques indications pour le développement

Tracer des graphiques avec ImageJ: les fonctions en `Plot.`

Le language macro embarque une série de fonctiones relatives au tracé de graphiques, toutes étant préfixées de `Plot.` . Tracer une courbe semble simple: il suffira de créer un système de repères avec la fonction `Plot.create("Titre du graphique", "Titre de l'axe x", "Titre de l'axe Y", donneesX, donneesY)` . Le graphique est affiché au moyen de l'instruction `Plot.show()` .

Pour une représentation statique, cette famille de fonctions est suffisante. Elle permet même une mise à jour des échelles x et y en fonction du contenu, grâce à la fonction `Plot.setLimitsToFit()` . Cette solution parait idéale... On repèrera rapidement les instructions `Plot.add(type, xValues, yValues)` , `Plot.setColor(color)` et `Plot.update()` qui devraient, respectivement, permettre d'ajouter une trace, de changer sa couleur et de mettre à jour le graphique.

Exercice *Testons donc les fonctions graphiques d'ImagesJ.* * Créer une nouvelle macro; * Créer trois tableaux de valeurs, xValues yValues et yValues2; * Créer un graphique au moyen de l'instruction `Plot.create` ; * Afficher le graphique en utilisant l'instruction `Plot.show` ;

Dans un second temps, ajouter à la macro les lignes correspondant * Modifier la couleur de tracé par défaut pour la prochaine courbe avec `Plot.setColor("red")` ; * Ajouter la seconde série de valeurs, sous la forme de lignes: `Plot.add("line"", xValues, yValues2)` ; * Mettre à jour le graphique.

Le code pourrait ressembler à ce qui suit: ```java //Partie 1 xValues=newArray(0, 1, 2, 3, 4, 5, 6); yValues=newArray(9, 4, 1, 0, 1, 4, 9); yValues2=newArray(3, 2, 1, 0, 1, 2, 3);

Plot.create("Graph test", "x", "y", xValues, yValues); Plot.show();

//Partie 2 Plot.setColor("red"); Plot.add("line", xValues, yValues2); plot.update(); ```

Malheureusement, si la partie 1 de la macro fonctionne, la partie 2 s'exécute avec une erreur, nous apprennant qu'un graphique, une fois affiché avec `Plot.show()` , ne peut être modifié. Les instructions sont pourtant valides: si on place le `Plot.show()` en toute fin de macro, le résultat attendu est bien retourné. La syntaxe du language a cependant sa propre "logique", à laquelle nous nous confrontons dans le cadre de cet exemple.

Lorsqu'on fait face à ce type de problème, on peut compter sur la force des exemples fournis dans la documentation des fonctions. Ainsi, on pourra s'inspirer de la macro ExamplePlots.txt, qui accompagne la section `Plot Functions` sur la page macro function du site d'ImageJ. on y apprend notamment que `Plot.update` permet d'afficher un nouveau graphique ou de le mettre à jour après que de nouvelles données y aient été ajoutées. Plus exactement, il permet de garder un même canevas, l'efface et y "ajoute" de

nouvelles données.

Exercice *Testons de nouveau les fonctions graphiques d'ImagesJ.* * Créer une nouvelle macro; * Créer trois tableaux de valeurs, xValues yValues et yValues2; * Créer une boucle qui permettra d'animer le graph; * Utiliser une fonction s'appliquant à un tableau, permettant d'en extraire une partie seulement pour générer les tableaux x, y et y1: ils contiendront la première valeur de chaque tableau original lors de la première itération, les deux premières lors de la seconde itération etc... * Créer un graphique au moyen de l'instruction `Plot.create` représentant y en fonction de x; * Changer la couleur au rouge; * Ajouter la courbe représentant y2 en fonction de x; * Repasser la couleur par défaut au noir; * Ajuster les échelles des axes pour que toutes les valeurs soient visibles; * Mettre à jour le graphique en utilisant l'instruction `Plot.update`; * Introduire un petit délai pour avoir le temps de visualiser l'évolution du graph (500msec devraient suffire).

Le code pourrait ressembler à ce qui suit: ```java xValues=newArray(0, 1, 2, 3, 4, 5, 6); yValues=newArray(9, 4, 1, 0, 1, 4, 9); yValue2s=newArray(3, 2, 1, 0, 1, 2, 3);

for(i=2; i<=xValues.length; i++){ x=Array.slice(xValues, 0, i); y=Array.slice(yValues, 0, i); y2=Array.slice(yValues2, 0, i);

```
Plot.create("Graph test", "x", "y", x, y);
Plot.setColor("Red");
Plot.add("line", x, y2);
Plot.setColor("black");
Plot.setLimitsToFit();
Plot.update();
wait(500);
```

}```

Nous avons bien fait de stocker toutes nos données dans un tableau de résultats ! Reste à en extraire les données pertinentes, en pointant la colonne dont on veut tracer les valeurs. Ne restera plus ensuite qu'à créer/mettre à jour le graph au moyen du tableau de valeurs extraites. Mais pour cette partie de l'implémentation, je vous laisse la concevoir par vous-mêmes.

Implémentation

INUTILE DE REGARDE LA SOLUTION, C'EST A VOUS DE CREER CETTE PARTIE DE LA MACRO !!!

Devoir

A vous d'implémenter les fonctionnalités suivants:

1. Créer une fonction `getColumnValues(title)`:
2. Son rôle est de générer une variable de type tableau, qu'elle remplie avec les valeurs trouvées dans la colonne "title" de la Results Table;
3. C'est une fonction particulière puisqu'elle **retourne** le tableau de valeurs: il suffira d'utiliser le mot-clé `return` pour transformer notre méthode en fonction;
4. L'appel à la fonction renvoyant un tableau, il doit être stocké dans une variable. L'attribution se fait simplement: `xValues=getColumnValues(title);`.
5. Créer une fonction `traceLeGraph()` qui... trace le graph:
6. Le graph correspondant à la colonne "Full_Image" doit apparaitre en noir, et être d'une épaisseur de 3 pixels;
7. Les graphs des données correspondant aux différentes ROIs doivent apparaitrent avec la même couleur que la ROI, et être d'une épaisseur de 1 pixel.
8. N'oubliez pas de faire les appels à ces fonctions: la macro globale doit être fonctionnelle.
9. Bonus: au final, la macro n'a que 2 paramètres: le délai et le timeout. On pourrait envisager de demander à l'utilisateur d'entrer ces paramètres au moyen d'un interface graphique. Pourquoi ne pas explorer la section `Dialog.` des fonctions macro et définir une fonction personalisée "GUI" ? Cette partie du devoir n'est pas obligatoire.

Barême/méthode de notation:

	Critère	Barême
+	La macro se lance et fonctionne	/5pts
+	Le code est commenté	/3pts
+	Le code est organisé en groupe fonctionnels	/2pts
+	Les fonctionnalités demandées ont été implémentées	5pts/par fonction
+	La fonction bonus a été implémentée. Elle est même commentée !	/2
-	Des modifications ont été nécessaires pour faire tourner la macro	-2pts/modification
-	Le code ressemble étrangement au code d'un autre étudiant et les commentaires de la macro n'indiquent pas que le travail ait été fait en "pair-coding" (qui est pourtant encouragé, pourvu qu'on dise qui bosse avec qui en entête de macro)	Aucune note ne sera attribuée
-	La macro est arrivée après la date limite	Aucune note ne sera attribuée