Optimisation mémoire... Eviter de ramer

Nombreux sont ceux qui parlent de ce terrible fléau que l'on appelle "L'appli qui rame".

Et oui ! Qui n'a pas déjà été déçu en voyant son travail saccader lors de la lecture de la projection ?... Tout particulièrement lorsque celle-ci passe par CD.

En effet, vous savez que votre appli utilise la RAM de l'ordinateur pour fonctionner. C'est à dire qu'à chaque fois que vous provoquer un événement sur votre appli (Appel d'un member, d'un son, d'une vidéo, d'une transition, voire le clic sur un bouton...), les ressources en RAM sont puisées.

Cela signifie que si l'ordinateur manque de RAM, Et bien... Ca rame...

Pourquoi ?

Pour une simple et bonne raison : Lorsque la mémoire vive (RAM) est saturée, la mémoire de votre Disque Dur (qui est beaucoup plus lente) prend le relais pour subvenir aux besoins de l'appli. D'où les saccades. N'oublions pas, qui plus est, que la RAM de votre ordinateur est déjà mis à contribution par d'autres programmes mais aussi par votre système d'exploitation. Imaginez donc, une machine avec windows XP et seulement 128 de RAM avec plusieurs programmes différents !!!

Bien sûr, la puissance du processeur, la vitesse du lecteur CD (quand l'appli est lue directement sur le CD) sont des facteurs très importants également qui vont faire la différence. Mais nous nous pencherons uniquement sur ce problème de mémoire.

L'inspecteur de mémoire :

Pour gérer au mieux l'utilisation de votre mémoire, il faut, avant tout, pouvoir observer les ressources puisées par votre appli durant son utilisation.

Pour cela, il existe un inspecteur sur Director qui vous permet de visualiser les variations de vos ressources.

En cliquant sur "Fenêtre/inspecteurs/mémoire", il apparaitra alors.

Vous pouvez alors voir une jauge qui prend en compte les ressources puisées par votre "movie", mais également celles utilisées par l'ensemble des autres programmes... Utiles, vous en conviendrez.

Attardons nous un peu sur cet inspecteur de mémoire.

Vous pouvez voir les indications suivantes :

- **Afficher la mémoire système totale disponible** : c'est à dire la RAM de votre ordinateur et la mémoire virtuelle disponible.
- **Mémoire physique** : (pour les utilisateurs de Windows) présente la quantité de mémoire RAM installée sur le système.
- **Taille partition** : (pour Mac) affiche la quantité de mémoire affectée à Director dans la case "Lire les informations" de votre OS (que vous pouvez modifier à volonté... L'un des points forts du Mac). Cette option n'est disponible que si l'option de mémoire temporaire est activée.

- **Total utilisé** : indique la quantité de mémoire RAM utilisée pour une animation, celle ci va varier selon les différentes manipulations de votre animation lors de la lecture. c'est aussi sur celle-ci que nous allons jouer pour réduire les "coûts" en mémoire.

- **Mém. disponible** : indique la quantité de mémoire encore disponible sur votre système (Donc votre RAM puis les ressources virtuelles).

- **Autre mémoire** : indique la quantité de mémoire occupée par les autres applications installées sur votre ordinateur. Nous ne pourrons donc pas jouer dessus. Cependant, il faut garder ce point en tête. N'oublions pas que les utilisateurs de votre appli se servent peut être de logiciels gourmands.

- **Programme** : indique la quantité de mémoire utilisée par Director, hormis l'espace occupé par le fichier d'application Director lui-même. Celle-ci est incompressible. Qui plus est, elle ne sera pas présente sur toutes les machines bien entendu (Le/la client(e) n'ayant pas forcément Director).

- **Distr. et scénario** : indique la quantité de mémoire utilisée par les acteurs dans la fenêtre Distribution ainsi que la notation dans la fenêtre "Scénario".
  Les acteurs comprennent tous les graphiques de la fenêtre Dessin, tous le texte de la fenêtre Texte, les acteurs qui utilisent l'encre Dessin seul dans le scénario, les miniatures de la fenêtre Distribution ainsi que tous les sons, palettes, boutons, animations vidéo numérique ou fichiers liés importés dans la distribution et actuellement chargés en mémoire.
  En bref, cela correspond à tous vos objets en mémoire dans vos distributions. Donc petite astuce : n'utilisez pas le "PSD" ni le "WAV" par exemple (format de bonne qualité, mais beaucoup trop lourd). Privilégiez le "PNG" et le "JPEG" si possible (quand vous n'avez pas besoin de jouer sur les alpha) et sur un format type "MP3" pour le son.

- **Tampon d'écran** : indique la quantité de mémoire que Director réserve pour un espace de travail pendant la lecture de certains éléments sur la scène.

Comme dit précédemment, nous allons donc jouer sur la partie "Total utilisé" pour optimiser votre mémoire.

Son utilisation :

Maintenant, activez votre animation et garder l'oeil sur votre inspecteur de mémoire. En effectuant diverses manipulations, vous allez vite vous rendre compte que le **"Total utilisé"** varie à la hausse. Parfois... Ca fait peur. A un moment, je n'osais même plus appuyer sur les boutons. 😉

Il s'agit à présent de limiter sa constante augmentation, la stabiliser à une valeur acceptable. Pour cela, plusieurs choses à faire.

Les phénomènes de purges :

Comme nous le savons, chaque acteur, quelqu'il soit, puise de la RAM. Il faut donc les enlever de la mémoire. Pour cela, cliquez sur l'inspecteur de propriété de votre acteur (le "i" entouré de bleu) puis sur l'onglet "acteur". Vous voyez apparaître une zone "Priorité de purges" avec quatre niveaux différents.

Comment ça marche ?

Chaque niveau est une méthode de gestion de la mémoire de votre objet. Je m'explique. Selon votre choix, votre acteur sera immédiatement purgé de la mémoire RAM lorsque celle ci sera saturée ou bien elle sera gardé pendant un certain laps de temps. Par défaut, la priorité de purge est toujours sur 3. C'est à dire que l'objet est balayé de la mémoire quasimment tout de suite lorsque Director se sentira pris à la gorge par le manque de ressource.

Les quatres niveaux sont :

- **–>** Dès que l'ordinateur manque de mémoire, l'acteur est purgé.

- **2 - Suivant** –> Si les ressources ne sont toujours pas suffisantes, les acteurs ayant cette priorité seront également "effacés"

- **1 - En dernier** –> L'acteur sera le dernier à être purgé (En bref, c'est l'extrême limite)

- **0 - Jamais** –> quelque soit les besoins de la machine, les acteurs restent en mémoire (Genre : Ma machine craint personne. Et puis celle des autres... Faudra qu'elles suivent !!!)

Il faut donc utiliser ces priorités à bon escient. Pour un objet que vous utilisez très fréquemment, il est intéressant de lui mettre un ordre de priorité moindre afin que celui ci ne soit pas effacé dès que la machine commence à souffler. Je vous donne un exemple : Un bouton qui mène vers des crédits est généralement utilisé une seule fois lors de l'utilisation. Par contre, votre bouton Menu doit rester en mémoire tout le temps pour que l'ordinateur n'ai pas à le recharger à tout bout de champ... Il vaut mieux que ce soit votre crédit qui s'efface temporairement que votre menu. En bref, tous vos acteurs les plus fréquemment utilisé méritent une purge de valeur 1 à 2 pour éviter d'être supplanté par des commandes subsidiaires.

Pour utiliser ces phénomènes de purges, vous pouvez utiliser le code Lingo également.

commande : **purgepriority**

L'instruction suivante affecte la valeur 3 à la priorité de purge de l'acteur Arrière-plan, ce qui signifie qu'il sera l'un des premiers acteurs à être purgés lorsque Director aura besoin de mémoire :

```
member("Arrière-plan").purgePriority = 3
```

Et pour les amoureux du code "verbose"

```
the purgepriority of member "Arrière-plan" to 3
```

Purger soi même

Il arrive pourtant que l'on désire, pour économiser la mémoire, purger un acteur lorsque un autre, par exemple le remplace.

Supposons que vous ayez réalisé un lecteur Audio avec plusieurs morceaux de votre choix.

Lorsque vous séléctionnez un nouveau morceau, le précédent, n'est pas pour autant purger de la mémoire. Cela prend donc beaucoup plus de mémoire. Pour remédier à ce problème, certaines commandes existent :

UNLOAD : Il permet de purger les acteurs qui se trouvent sur les images choisies :

Par exemple :

L'instruction suivante supprime de la mémoire les acteurs utilisés dans l'image 10 :

```
unLoad 10
```

Mais vous pouvez également purger une fourchette d'images. Ceci est très utile pour enlever les acteurs d'une animation par interpolation devenus inutiles.

L'instruction suivante supprime de la mémoire les acteurs utilisés dans les images 1 à 50 (inclus) :

```
unLoad 1, 54
```

Vous pouvez également utiliser les marqueurs comme repère.

```
unLoad "première", "dernière"
```

Enfin, vous pouvez aussi ne pas mettre d'argument. Dans ce cas là, Toutes les images d'une anim sont purgées.

UNLOADMEMBER

Cette commande a la même fonction, si ce n'est que l'on ne spécifie pas par image mais par acteur. Elle ne se limite donc pas à votre scénario. La purge affecte la distribution également.

Par exemple :

```
unloadmember member "écran"
```

ou

```
member "écran".unload()
```

de même qu'avec **"unLoad"**, vous pouvez choisir une fourchette d'acteurs en utilisant deux arguments

Par exemple :

L'instruction suivante purge de la mémoire tous les acteurs compris entre l'acteur 1 et l'acteur écran :

```
unLoadMember 1, member "écran"
```

ou

```
member(1).unload("écran")
```

Enfin, si vous n'utilisez pas d'argument, tous vos acteurs seront purgés de votre animation.

revenons donc à notre lecteur audio :

Soit deux morceaux de musique : member "Radiohead" et member "MassiveAttack"

Sans la commande unloadmember, les deux morceaux sont gardés en mémoire.

Par contre, dans le cas suivant :

```
on mousedown me

unloadmember member "Radiohead"

puppetsound 1, member "MassiveAttack"

end
```

Au clic de souris, Le member "Radiohead" sera purgé alors que le member "MassiveAttack" est rendu actif.

Ces deux commandes vous permettrons déjà de stabiliser votre inspecteur de mémoire. Faites un test avec l'exemple précédent en enlevant le unloadmember. Observez votre inspecteur. Puis recommencez avec la commande. vous allez voir, votre mémoire va vous remercier :wink:

Cependant, ces commandes doivent être utilisées selon le contexte. En effet, qui dit purge, dit également nécéssité de le recharger

lors d'un nouvel appel. cela peut prendre quelques instants selon la nature de l'acteur. En effet, soit vous gardez vos éléments en mémoire (pas besoin de charger) mais votre appli risque de ralentir car la mémoire sera saturée, soit vous purgez vos éléments pour stabiliser vos ressources, mais le rappel sera légèrement plus long.

Les préchargements

Pour gagner en fluidité, il est possible de précharger les acteurs dans la mémoire avec diverses commandes. Ces commandes sont très utiles pour les vidéos par exemple. Leur poids sont en effet souvent un rempart à la fluidité.

PRELOAD :

précharge en mémoire des acteurs de l'image ou de la plage d'images spécifiée et s'arrête lorsque la mémoire disponible est saturée ou que tous les acteurs spécifiés ont été préchargés. Il fonctionne de manière un peu analogue au Unload (Sauf qu'il charge en mémoire au lieu de purger :wink:) si ce n'est qu'il précharge à PARTIR de l'image choisie et ce jusqu'à la fin de l'animation.

Par exemple :

Les acteurs qui se situent entre l'image 10 et la fin de l'anim sont préchargés.

```
preload 10
```

Les acteurs qui se situent entre le marker "pouet" et la fin de l'anim sont préchargés.

```
preload marker (pouet)
```

Mais vous pouvez, une fois de plus, établir une fourchette d'images en mettant un deuxième argument :

Les acteurs qui se situent entr l'image 10 et l'image 50 sont préchargés.

```
preload 10, 50
```

Les acteurs qui se situent entr l'image 10 et le marker "pouet" sont préchargés.

```
preload 10, marker (pouet)
```

En l'absence d'argument, la commande précharge tous les acteurs utilisés depuis l'image courante jusqu'à la dernière image d'une animation.

PRELOADMEMBER :

précharge les acteurs et arrête le préchargement lorsque la mémoire est saturée ou que tous les acteurs spécifiés ont été préchargés.

Si l'argument "quelActeur" est fourni, preLoadMember ne précharge que cet acteur. Si "quelActeur" est un nombre entier, il n'est fait référence qu'à la première distribution. Si "quelActeur" est une chaîne, le premier acteur possédant cette chaîne comme nom est utilisé.

Exemple :

L'instruction suivante précharge l'acteur 20 de la première distribution (interne) :

```
member(20).preLoad()
```

Pour précharger un acteur particulier d'une distribution spécifique, utilisez la syntaxe suivante :

```
member("Shadowrun", "cyber").preLoad()
```

–> Shadowrun étant l'acteur et "cyber" le nom de la distribution

Si les deux arguments "deLacteur" et "aLacteur" sont fournis, la commande preLoadMember précharge tous les acteurs dans la plage spécifiée par les noms ou numéros des acteurs.

Exemple :

L'instruction suivante précharge dans la fenêtre Distribution l'acteur Temple et les 10 acteurs qui le suivent :

```
member("Temple").preLoad(member("Temple").number + 10)
```

En l'absence d'arguments, preLoadMember précharge tous les acteurs de l'animation.

Ces deux commandes sont généralement placés dans un script d'animation (movie script) pour plus d'efficacité. En effet, elles seront prises en compte durant toute l'animation.

PRELOADMOVIE :

précharge les données et les acteurs associés à la première image de l'animation spécifiée. Le préchargement d'une animation permet de la faire démarrer plus rapidement à la suite d'une commande go to movie ou play movie. Ainsi, lorsque vous lancer une autre animation, tous les éléments sont préchargés avant son lancement.

Par exemple :

L'instruction suivante précharge les objets de l'animation équipe qui est située dans le même dossier que l'animation courante :

```
on mouseup me
 preLoadMovie "équipe"
```

```
 go to movie "équipe.dir"
end
```

NB : Il existe ensuite d'autres commandes tel que le preloadnething pour internet, preloadRam... Mais elles seront vues dans un autre article. Nous verrons également le "queue" qui peut s'avérer très utile.

Eviter les trop grosses projections

En effet, si vous mettez tous vos ".dir" dans une même projection, tous les éléments de votre appli seront gardés en mémoire. Imaginez que vous ayez une quinzaine de movies avec intégration de son et de vidéo, plus vous avancerez dans votre programme, plus votre ordinateur en souffrira.

Préférez donc une projection de lancement avec les .dir qui gravitent autour. Ainsi, au lieu d'avoir une projection lourde, vous aurez plusieurs petites animations qui seront "tuées" lors de leur fermeture grâce au code suivant :

```
on closewindow
 -- opérations de maintenance standard
 forget the activewindow -- Libère la fenêtre courante de la mémoire
end
```

Cela dit vous pouvez utiliser cette commande avec un "on mouseup" par exemple :

```
on mouseup me
 forget the activewindow
 go to movie "lance.dir"
end
```

Ce petit executable qui va vous permettre de naviguer vers vos divers movies est appelé "Stub de Lancement" ou "Stub projector". Cette méthode offre quatre avantages :

Comme dit précédemment, la mémoire ne sera pas saturée car les données de chaque fenêtre seront tuées au fur et à mesure

- **La grande rapidité de démarrage :** En effet, plus votre éxécutable sera léger et moins l'ordinateur devra se lancer dans des calculs rocambolesques vous laissant le temps d'aller boire votre café Mamie Nova. Ayant moins d'informations à gérer au préalable, votre application apparaitra quasimment instantanément.
- **La rapidité de mise en projection :** Et oui, si votre éxécutable est léger, il ne faudra que quelques instants à Director pour en faire la projection. Vous pourrez constater le gain de temps considérable de cette méthode (Cette fois ci, vous irez boire le café Mamie Nova avec des amis sans penser à votre projection).
- **Mise en éxécution de tâches multiples :** Vous pouvez lancer vos movies, mais également placer vos préchargements de distributions, gérer toutes les détections dont vous avez besoin (quickTime, Lecteur Flash...), mais également déclarer toutes vos globales (Variables qui seront prises en compte dans tous vos movies).

Alooooors ??? Convaincu ??

Mais comment faire un stub de lancement ?

- Tout d'abord, Créer un dossier sur votre bureau. Vous pouvez l'appeler comme vous le désirez. Dans notre cas, nous le nommerons "Mon projet"

- A présent, ouvrez ce dossier et créer un autre dossier à l'intérieur que vous nommerez "Movies" Ensuite, copiez tous les éléments de votre appli dans ce dossier "Movies" (cad : ".dir", Distributions externes, vos dossiers de vidéos...[Sauf les Xtras])

- A la racine du dossier "Mon projet", créez un nouveau dossier appelé "Xtras". (Il offre la possibilité d'un démarrage plus rapide car vous aurez séléctionné les Xtras dont vous avez besoin. Cette partie, facultative cela dit, sera présentée plus tard)

- Ouvrez votre ".dir" qui servira de stub et créez votre projection en ajoutant aucun autre movie. Cette projection que vous créez DOIT se trouver à la racine du dossier "Mon projet" afin qu'elle soit visible en priorité.

Et les autres modes de projections dans tout ça ?

Il existe en effet d'autres modes de projection que le stub, je vais vous faire un rapide descriptif de ces modes avec un bref comparatif de vitesse. Vos relations entre votre stub de lancement et vos movie se fera alors automatiquement.

- **Projection Standard :**

Tous les Xtras sont internes (Cad que vous n'avez pas fait un dossier externe avec les Xtras dont vous vous servez). de même tous vos éléments sont généralement dans des distributions internes. Pour utiliser cette méthode lors de votre création de projection, faites option puis cliquez sur "Standard". Cette méthode créé la projection ayant la taille la plus importante. La vitesse dépendra de votre appli, de votre micro processeur.

- **Projection compressée :**

Dans le principe, elle dorrespond à votre projection standard avec l'option "compressed" à la place de "standard" (ce serait un peu le .zip d'un fichier par exemple) Sa taille est bien évidemment moindre qu'une projection standard. Son lancement sera plus lent qu'une projection standard du fait que l'ordinateur doit d'abord décompresser votre application.

- **Stub de lancement :**

Je ne reviens pas sur sa définition expliquée plus haut :wink:, si ce n'est que vous devez choisir l'option "Standard" En terme de taille, il serait le compromis entre les deux modes présentés ci dessus. En effet, il est plus léger que le "standard" du fait de sa segmentation, mais il n'est pas compressé. Il reste donc plus lourd que le mode "compressed". En terme de vitesse, ce stub est plus rapide que les deux modes précédents pour une simple raison : Il n'y a pas de décompression nécessaire malgré sa taille réduite.

- **Projection Shockwave :**

Les Xtras sont en interne mais l'option utilisée est "Shockwave". Ce mode est plus léger que les versions "Standard" et "Stub de lancement" Le mode shockwave offre également un lancement plus rapide que le "Standard"

- **Fast-Start Projector :**

Vos Xtras sont externalisés dans un dossier et utilise l'option Shockwave tout comme votre projection. ce mode détient la palme de la projection la plus légère. C'est avec ce mode que votre appli se lancera le plus rapidement.

```
"Attention:
Les modes shockwave offrent de nombreux avantages, mais il ne faut pas oublier que pour être lu correct
l'utilisateur doit posséder un lecteur shockwave contrairement aux autres modes qui ne requièrent
aucun équipement nécessaire."
```

Bien sûr, cet article ne présente pas tous les cas de figures mais constitue seulement une première approche de la gestion de mémoire. Pour les cas particuliers, je vous invite à taper frénétiquement sur votre touche F1 mais également à poser vos questions si vous désirez plus d'informations. Je réactualiserais cet article prochainement avec les différentes suggestions qui m'ont été faites.

tutoriaux/director/memory_optimize.txt · Dernière modification: 13/10/2005 22:23