La Matrice et sa représentation graphique.

Plan:

A) Basic:
- I) Ecran principal
  - 1) Théorie
  - 2) Pratique
- II) Ecran graphique
  - 1) Théorie
  - 2) Pratique

B) Basic étendu: Basic et ASM
- I) Ecran principal
  - 1) Théorie
  - 2) Pratique
- II) Ecran graphique
  - 1) Théorie
  - 2) Pratique

C) Lescique

D) Conclusion

A) Basic:

I) Ecran principal:

1) Théorie

La matrice* la plus simple d'utilisation est bien entendu celle qui gère des graphismes "ASCII*", car les éléments qui la composent se trouvent aux mêmes coordonnées que les caractères à l'écran. La matrice définit ce qui est à l'écran pour votre programme. Le système de base le plus souvent* utilisé est simplissime: 0 dans la matrice, et il n'y a rien aux mêmes coordonnées sur l'écran; 1 dans la matrice, et quelque chose est affiché à l'écran à ces coordonnées.
Parlons par exemple la pièce d'un RPG*, et admettons qu'un mur soit représenté par une croix (X) et un monstre par un M.

L'écran et la matrice seront alors comme ceci :

écran :

X X X X X
X M X
X X
X X X X X

matrice :

[[1,1,1,1,1]
[1,1,0,0,1]
[1,0,0,0,1]
[1,1,1,1,1]]

Au niveau des 0 dans la matrices, le personnage pourra se déplacer, mais il sera bloqué au niveau des 1. Le monstre et les murs sont tous deux des 1 : le personnage ne pourra donc pas les "recouvrir" à l'écran.
Comme cela a déjà été précisé, les coordonnées à l'écran sont les mêmes que dans la matrice : en (2,1) dans la matrice est un 1, et en (2,1) de l'écran se trouve un mur. Un simple test de la valeur de la matrice permettra de connaître ce qui est affiché.
Bien entendu, vous ne pourrez pas vous contenter de 1 et de 0 dans votre matrice. Si vous voulez qu'on puisse combattre le monstre, sa valeur sera différente de celle du mur, par exemple 2. Mais pour plus de facilité, choisissez une valeur "bloquante" ou de "passage" :

- Toujours avec l'exemple du RPG, on décide que le 1 est bloquant : ainsi, on teste la matrice pour savoir si le personnage peut avancer : si la matrice vaut 1, il ne pourra pas. Si elle vaut 0, 2 ou tout autre valeur différente de 1, il pourra. De cette manière, on attribue la valeur 3 aux coordonnées de la matrice correspondant à l'emplacement d'une flaque à l'écran. Le personnage pourra passer dessus, car elle est différente de 1 ; mais si on a besoin de savoir qu'il y a une flaque, il suffira de tester la matrice.
- Au contraire si on décide que le 0 est la valeur de

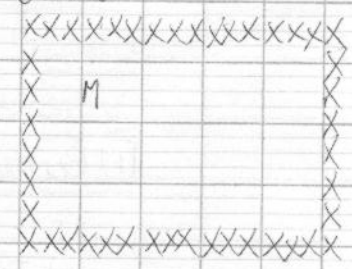
"passage", alors il suffira de tester la matrice pour savoir la valeur de ce qui est affiché à l'écran: si c'est 0, on passe, pour toute autre valeur, on reste sur place.
Pour un programme qui utilise une matrice peu complexe, il vaut mieux utiliser la valeur de "passage" qui est plus simple d'utilisation. Mais pour un programme utilisant une matrice plus compliquée, le choix d'une valeur "bloquante" s'impose, car elle offre plus de possibilité du programmeur.

2) pratique

matrice: ([A])

```
[[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1]
 [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1]
 [1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1]
 [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1]
                  ||
                  ||
                  ||
 [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1]]
(matrice dim (8x16))
```

écran:



Seulement deux valeurs dans la matrice: 1 est "Bloquant", 0 est de "passage".
les "x" sont des murs, M est un monstre.
Voici maintenant le programme permettant le déplacement du personnage:

```
2→A:2→B                  A et B seront les coordonnées du perso
Output(A,B,"O            affichage du perso
Repeat G=45              début de la boucle principale. Si CLEAR
                         est pressé, fin du programme.
```

```
Repeat G                    On attend l'appui d'une touche.
Getkey→G
End
Output(A,B," "              On efface le perso
A→D:B→E                     On sauve ses coordonnées
D+(G=34)-(G=25→D            déplacement des coordonnées Y
E+(G=26)-(G=24→E            déplacement des coordonnées X
If not([A](D,E              . Si la matrice renvoie 0 aux nouvelles coor-
Then:D→A:E→B                données, on les garde, sinon, on revient aux
End                         anciennes
Output(A,B,"O               On réaffiche le perso
End                         Fin de la boucle principale et du programme.
```

II Ecran graphique

1) Théorie

Le déplacement d'un sprite* sur l'écran graphique ne demandera pas forcément de matrice: si on utilise seulement deux valeurs, 0 et 1, dans une matrice, autant utiliser directement un Pixel-Test, en ayant choisi auparavant l'emplacement obligatoire d'un pixel pour chaque objet au même endroit.
On aura besoin d'une matrice à partir de trois valeurs. Mais cette matrice ne sera pas à la taille de l'écran (96×64), ce serait trop lourd et complètement inutilisable.
On divisera donc l'écran en rectangles ou carrés pour chaque sprites, de manière à ce que tous les sprites soient de la même taille. En Basic, seuls 95×63 pixels sont programmables. Il faut donc savoir qu'un écran divisé en sprites ne sera pas entièrement rempli: il restera une bande inutilisée sur les bords.

Une autre manière de dessiner sur l'écran graphique peut être de se servir de la commande Test(. L'affichage est plus rapide, mais à cause du nombre de caractères limité, cette méthode est rarement utilisée. Ce sera à vous de choisir entre vitesse-taille et beauté des graphismes.

Pour le déplacement du sprite principal, on utilisera les coordonnées de la matrice, et on fera un rapide calcul pour l'afficher.
Si le commencement de la mosaïque de sprites* ne débute pas au point (0,0) à l'écran, on appellera X le décalage en colonnes et Y celui en lignes.
R sera la taille en colonnes du sprite, S la taille en ligne.
A sera la coordonnée X dans la matrice, B la Y.
Pour le déplacement du sprite principal, on fera donc la même chose en substance que pour l'écran principal.

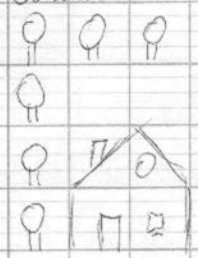
- effacement du sprite
- sauvegarde des coordonnées
- le déplacement du sprite
- test de la matrice
- suivant le résultat, on garde les nouvelles coordonnées ou on reprend les anciennes
- on réaffiche le sprite.

Maintenant la formule qui servira à l'effacement et à l'affichage du sprite sera:
en X: . $R(A-1)+X$
en Y: . $S(B-1)+Y$
Ne pas oublier que pour le test de la matrice, X et Y sont inversés, et on n'écrira pas: If [A](A,B... mais If [A](B,A...

2) Pratique

On utilise des sprites 10×10 échelle : 10 (10 ↔, 10 ↕) en pixels.

écran :



matrice :

[[1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
[1, 0, 0, 0, 0, 0, 0, 0, 0, 1]
[1, 1, 1, 0, 0, 0, 0, 0, 0, 1]
[1, 1, 1, 0, 0, 0, 0, 0, 0, 1]
[1, 0, 0, 0, 0, 0, 0, 0, 0, 1]
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1]]

Pour centrer l'écran, le décalage en X est X=2 et celui en Y est Y=2. Pour la taille du sprite, R=S=10.
Le sprite est contenu dans le programme prgm PERSO, créé avec Sprites Converter * et qui XOR * un personnage de votre choix en (A=Y ; B=X).
Les coordonnées de la matrice sont : D et E.
Voici le programme :

```
2→D : 2→E                  coordonnées de la matrice
10(D-1)+2→A                | calcul des coordonnées où on
10(E-1)+2→B                | affichera le sprite
prgm PERSO                 on affiche le sprite
Repeat G=45                début de la boucle principale
 Repeat G                  | recherche si une touche a
 getKey→G                  | été pressée...
 End
 10(D-1)+2→A               | calcul des coordonnées et effaçage
 10(E-1)+2→B               | du sprite
 prgm PERSO
 D→H : E→I                 sauvegarde des coordonnées
```

```
H+(G=34)-(G=25→H   | déplacement des coordonnées suivant la touche
I+(G=26)-(G=24→I   | pressée
If not([A](H,I     | test de la matrice
Then: H→D          | si elle vaut 0, on garde les nouvelles coordonnées
I→E: End
10(D-1)+2→A        | réaffichage
10(E-1)+2→B        | du
prgm PERSO         | sprite
End                | Fin de la boucle et du programme.
```

B) Basic étendu* : Basic et ASM*

I) Ecran principal

1) Théorie

Vous vous demandez sûrement où se situe l'intérêt d'un programme en basic étendu utilisant des matrices pour l'écran principal. Outre une diminution du poids et une augmentation de la vitesse, l'important est le nombre de caractères inutilisables en Basic pur*.

*en basic

Si on sait programmer les matrices pour l'écran principal, on aura pas de problème en Basic étendu. Le seul changement notable est que en ASM, le premier caractère de l'écran est en (0;0) et pas en (1;1), on décalera donc les X d'un vers la gauche et les Y d'un vers le haut.

Si A et B sont les coordonnées de la matrice, A-1 et B-1 seront ceux de l'écran.

2) pratique

On reprends l'exemple du A) I) 2) et le programme ASM prgm PERSO demande en Ans* une liste {X, Y, Z}; avec X et Y coordonnées du personnage et Z égal à 0 pour l'effacer, et égal à 1 pour l'afficher.
On ne changera alors dans le programme que deux lignes:

- Output(A,B,"↵.
  deviendra:
  {B,A,0:Asm(prgm PERSO
- Output(A,B,"0
  deviendra:
  {B,A,1:Asm(prgm PERSO

II) Ecran graphique

1) théorie

C'est pour un programme représentant une matrice dans l'écran graphique que le mélange Basic et ASM devient réellement intéressant, et ce pour plusieurs raisons:

- la place: par rapport à un affichage de sprites en basic pur, la création du même programme en ASM permet de diviser la taille du programme par 3, voire par 4.
- la vitesse: un affichage en ASM deux fois plus rapide qu'en Basic sur une 84+*, et comme la vitesse en ASM est la même sur 84+ que sur 83+*, on gagne jusqu'à 5 fois plus de vitesse si vous possédez une 84+

La principale difficulté du Basic étendu est que si les coordonnées de la matrice sont comptées de haut en bas et de gauche à droite, ceux des pixels en ASM vont de ~~haut en bas~~ X (jusque là, pas de problème), et de bas en haut.

X gauche à droite

Le calcul que nous avons vu en A)III)1) va être encore compliqué. Mais grâce à la possibilité de l'assembleur d'afficher les pixels aux coordonnées (63;95) contrairement au basic Pur, on supprimera donc le décalage de la formule du A)II)1).
On a donc: . R : taille en colonne du sprite
. S : taille en ligne du sprite
. A : taille en colonne de la matrice
. B : taille en ligne de la matrice
La formule qui servira à l'effacement et à l'affichage du sprite sera:
en X : $-RA+71$
en Y : $SB-9$
On n'oubliera pas que pour le test de la matrice, X et Y sont inversés, on écrira donc : If not ([A](B,A...

2) Pratique

On reprend l'exemple du A)II)2), et on crée le programme ASM prgm PERSO qui nécessite en Ans une liste {X,Y}, avec X et Y coordonnées du sprite qui se XOR.
On changera alors les lignes:
- $10(D-1)+2 \to A$
$10(E-1)+2 \to B$
prgm PERSO
deviendra:

{#10B-7,-10A+73 : Asm(prgmPERSO
_et de même pour le réaffichage

Xpas Notons dans cet exemple que je n'ai écrit (10B-7,-10+71); car j'ai repris le décalage de deux pixels de l'exemple du [illegible] 2).
Il n'y aura pas ce décalage si le sprite fait 8×8, 16×16 ou 32×32 pixels car ce sont des multiples du nombre de pixels de l'écran (96×64 pixels)

c) lexique

Ans: Dernière ligne de commande mémorisée. Elle peut être un nombre, une liste, une matrice, un string...
ASCII: (table de caractères contenant les 256 charactères disponibles sur la TI*.
ASM: diminutif de Assembleur, l'un des deux langages (avec le Basic) fourni avec les TIs, le plus rapide des langages, le plus proche de la machine, car compilé avant l'exécution.
Basic étendu: mélange de deux programmes, l'un en Basic et l'autre en ASM. Le programme Basic exécutera le programme ASM, et très rarement le contraire.
Basic Pur: langage programmable on-calc*, simple d'utilisation, compilé pendant l'exécution.
Bloquante(valeur): valeur choisie pour être celle qui bloquera la progression durant la représentation d'une matrice.
Matrice: Ensemble de nombres à n dimension. À 1 dimension, ce sera une liste. les matrices des TIs sont à 2 dimensions et sont représentées par des tableaux.
On-calc: (langage programmable). Langage développable directement sur la calculatrice.

De passage (valeur): valeur choisie pour être celle qui laissera passer le sprite durant la représentation d'une matrice.

Sprite: dessin de dimension variable mais généralement petit, fait pour être déplacé à l'écran.

Sprites Converter: programme crée par Sylo-Dias permettant de convertir un dessin en sprite exécutable en basic et s'affichant aux coordonnées (A;B)

TI: sigle de la firme Texas Instruments. Désigne généralement une calculatrice.

XOR: mode d'affichage des pixels, changeant leur état.

83+: TI possédant 180 kotets de ROM et tournant à 6 MHz, possédant un moteur zilog80

84+: TI possédant 480 ko de ROM, tournant à 15 MHz, et possédant un moteur zilog80

D) Conclusion

Avec ce tutorial, vous êtes maintenant en mesure de programmer X autres que d'arcade. A mon avis, la meilleure méthode de vous entraîner est de programmer un petit jeu utilisant les matrices. Cette leçon deviendra indispensable par la suite, car vous utiliserez souvent les matrices. Elles sont le moteur de ma série de RPG (Fight and Magic) et de quelques uns de mes autres jeux. En espérant que ce tutorial aura été suffisamment clair et qu'il vous servira.

X des jeux

Sylo-Dias