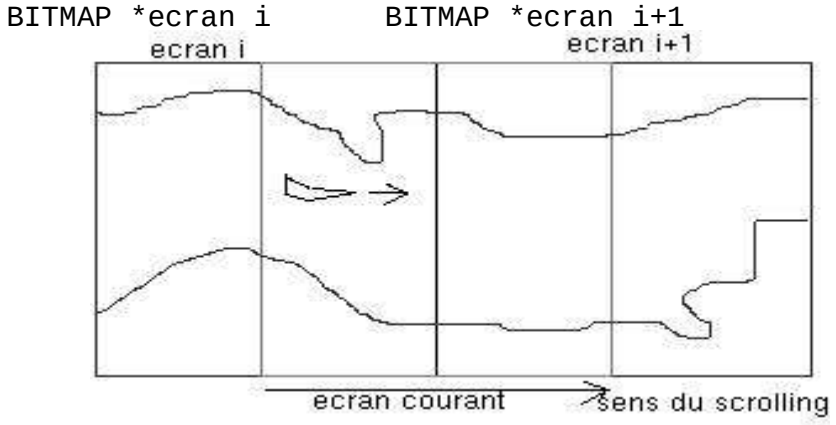
Mathieu Deschamps
20/02/2002

Conception
&
Structure de données

STOCKAGE DES GRAPHISMES ET ETATS LOGIQUES EN MEMOIRE VIVE

Un niveau de jeu se compose en plusieurs écrans. Nous avons besoin pour réaliser le scrolling de connaître l'état mémoire de l'écran courant i à afficher et de l'écran suivant i+1.

Il faut donc créer une structure de donnée de type BITMAP* car ce type est compatible avec toutes les fonctions d'Allegro. Cette structure representera d'abord toutes les images a afficher sur un écran de niveau. Pendant le cycle principal, seulement deux de ces structures seront chargés en mémoire vive pour effectuer le défilement.



La structure d'image graphique BITMAP qu'offre Allegro est la suivante :

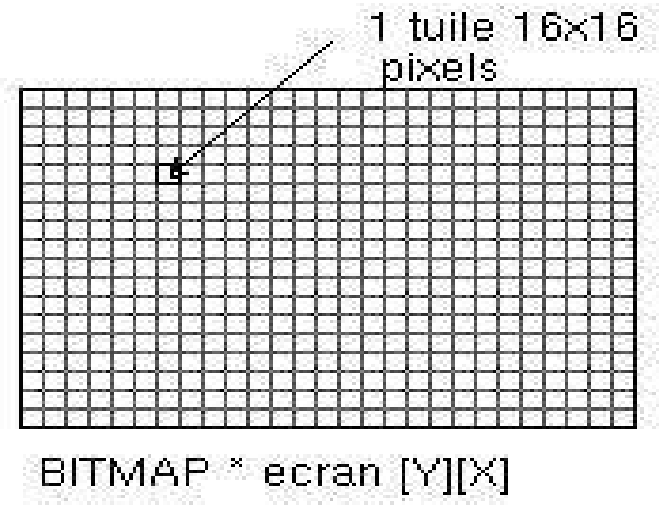
```
typedef struct BITMAP
{
 int w,h                    //longueur et largeur de l'image en pixels
 int clip                   //booléen d'activation du mode clipping
 int cl,cr,ct,cb            //coordonnées du rectangle »clip »
 int seg                    //segment mémoire
 unsigned char* line[]      //ligne de pixel de l'image
}
```

Nous pourrions utiliser alors une BITMAP* ecran i et une BITMAP* ecran i+1 puis réaliser le scrolling avec ces deux BITMAP de dimension de l'écran (ex:640x480). Mais cela a un inconvénient majeur, il faudrait redessiner BITMAP* ecran i de 640x480 à chaque modification de l'image, par example, suite à l'explosion d'une petite partie du décor.
Cette conception n'est même pas envisageable, nous voulons pouvoir redessinner toutes les tuiles de l'écran s'il le faut en un moment donné sans subir de ralentissement.

Aprés analyse de code source de jeu existant, nous avons remarqué que la solution existe : une conception par tuiles. L'écran est composé en fait d'un quadrillage de petite tuiles de taille standard 16x16

pixels. L'objet tuile etant aussi un BITMAP*, chaque tuile peut stocker donc une image.

Ainsi notre probléme est résolu : il suffit simplement de redessinner la tuile modifiée par l'explosion.



Une tuile ou tile en anglais sera notre élément graphique de base. Le schéma ci-dessus représente un écran graphique par exemple en 640x480 composé de tuile de 16x16.
X= 640 / 16 = 40
Y= 480 / 16 = 30
ecran=XxY=40X30=1200 tuiles.

De plus, il faut distinguer les éléments de nature différente qui compose une image d'écran de jeu de X-JAC. En effet,nous aimerions qu'une méthode unique soit compatible avec chaque type d'éléments.
Ainsi, nous fixons que les éléments graphiques d'un certain type appartiennent à une couche. Nous distinguons cinq types de couches au total par écran graphique.

Par la suite, toutes les couches seront appliquer les unes sur les autres en utilisant la méthodes de « blitage » suivantes:
copier/coller dessus : blit()
copier/coller masqué : masked_blit()

Ces opérations se feront sur des BITMAP* internes de travail, et des BITMAP* intermédiaires ou buffer. Ces buffers seront ensuite blités à l'écran extern BITMAP* screen d'Allegro.

L'affichage final n'est donc qu'un empilement de couches blité à l'écran. Ceci est rendu possible grâce au fait qu'un bitmap lors de sa création mémoire est initialisé par défaut à la couleur blanche {rouge=0,vert=0,bleu=0}, l'élément neutre qui permet les actions de « blitage ».

En ajoutant 3 dimensions a notre tableau de BITMAP*, voici notre structure de données en couche qui a elle seule représente tout les éléments graphiques d'un écran de jeu:

BITMAP* ecran [Nombre_de_couches : 5] [Y] [X]

- couche 0 : Objets
  Les Objets sont tous les éléments graphiques qui sont immobiles à l'écran et dont nous n'aurons donc pas à calculer les déplacements. Cela implique aussi qu'ils ne seront pas redessinés en général.

- couche 1 : Sprites
  Les Sprites sont tous les éléments graphiques qui sont mobiles à l'écran et dont il faudra calculer les déplacements et par conséquent seront redessinés plus souvent.

- couche 2 : Propriétés

  Cette couche n'est pas une couche graphique. Elle paraît est un peu intruse dans cette structure mais, c'est une couche logique qui permet de coder des états logique pour chaque tuile. Les états logiques seront codés grâce à la couleur de certains pixels et à leur emplacement dans une tuile. Le but est de coder des etats logiques que l'on peut de changer à la survenu d'un ou plusieurs
  évenements.

  Pour chaque couche, on a pour l'instant besoin de est_DESTRUCTIBLE,est_PASSABLE en lecture seule, et de a_DESSINER,est_DEPASSE en lecture/ecriture.

  Exemples d'utilisation:

  IMPLEMENTER LES REGLES/CONTRAINTES DE JEU
  - La tuile de la couche objet est elle passable par des sprites, si non il y aura collision.
  - La tuile de la couche objet est elle destructible par des sprites, si oui penser à la redessinner.
  - La tuile de la couche objet est elle a dessiner a l'écran.
  - etc, etc...

  CONTROLE EVENMENTIEL
  Les déclencheurs permettent de spécifier une déclenchement d'action du jeu lors de sa traversée.

  - La tuile représente le point de départ/fin du niveau.
  - La tuile a la propriété d'accélerer la vitesse d'un sprite la traversant
  - etc, etc...

  Le nombre de propriétés codables se calcule pour une tuile de 16x16 se composant de 256 pixels. Mettons que l'on fixe que chaque lignes de pixels de cette tuile code les propriétés des tuiles de chaque couche, cela donne donc 16 pixels pourcoder 16 proprités différentes. De plus, si on code la propriété selon la couleur de ce pixel, par exemple avec une profondeur de couleur de 8 bits, on obtient déjà 128 propriétés différentes !

- couche 3 : Fond
  Cette couche represente le fond ou l'univers graphique dans lequel les objets et sprites évoluent par exemple dans l'espace.

- couche 4 : Fond2
  Cette couche faculatative est reservée pour un usage ultérieur.