Tutorial librairie multimédia
ALLEGRO 4.2.3

Frédéric Drouillon
Professeur d'informatique

Partie 1 :
Tutorial Allegro

Table des programmes d'illustration

C1 - Démarrer avec Allegro

C2 - Couleur, image Bitmap

C3 – Concevoir une animation

Partie 2 :
Eléments algorithmiques pour les jeux

Les contenus C4 et C5 sont donnés sur le site http://fdrouillon.free.fr à travers différents documents mais ils ne seront pas intégrés dans le tutorial cette année.

C4 – Elaborer un monde

1. Avoir un décor

2. Déplacer des personnages dans un décor

3. Diviser son programme en plusieurs modules

C5 – Lignes et chemins

1. Récupérer une ligne, un contour

2. Disposer de trajets

3. Rechercher un chemin, bases

4. Recherche de chemins, l'algorithme A-star (A*)

C1 - Démarrer avec Allegro

Introduction : notions d'environnement de développement

La notion d'environnement de développement est assez large. Elle comprend généralement compilateur, différentes sortes de programmes utilitaires et des librairies. Les librairies sont souvent indépendantes, prévues pour être utilisées dans différents environnements, sous plusieurs systèmes d'exploitation avec différents compilateurs. Notons également que le développement de programmes suscite parfois d'autres activités que la programmation. Par exemple un peu de dessin pour avoir des bitmaps, de la modélisation 3D, du traitement d'échantillons sonores pour les sons utilisés etc. Chacune de ces activités suppose l'utilisation de logiciels spécialisés.

Essentiellement l'environnement de programmation est un compilateur, un IDE (pour « Integrated Development Environment » et des librairies. Les librairies sont des ensembles de codes sources préétablis, pensés et rassemblés en fonction de projets spécifiques. Outre les librairies standards du C (stdlib.h, stdio.h etc) il y a par exemple les Microsoft Fondation Classes (MFC) pour des travaux de bureautique associés à Windows, le sound manager (Macintosh) ou Fmod (linux,windows) pour ce qui est son, pthread pour les threads et bien d'autres, probablement des milliers. Les librairies peuvent être compilée et prendre plusieurs aspects : point C, point h (headers) point lib (Visual C++) point a (CodeBlock), DLL (Dynamique Linked, Librarie).

En particulier lorsqu'une librairie est introduite dans un projet elle fait l'objet d'un lien à la compilation ce qui nécessite une opération de « linkage », une opération qui permet de rendre accessible la librairie au compilateur. Le plus souvent la librairie utilisée nécessite également une inclusion de « header » du type #include « la_librairie.h » par exemple.

Concept de la librairie Allegro, communauté de développement, add-on

Allegro est une librairie de développement pour les jeux vidéo et autres types de programmes multimédia en C/C++. Elle est distribuée gratuitement et supporte les plates-formes DOS, Unix (Linux, FreeBSD, Irix, Solaris, Darwin), Windows, QNX, BeOS et MacOS X. Allegro fournit de nombreuses fonctions graphiques et sonores, gère le clavier, la souris, le joystick et des timers haute résolution. Elle dispose également de fonctions mathématiques 3d et en point fixe, de fonctions de gestion de fichiers de données compressés et d'une interface graphique. Vous aurez plus de détails sur la page d'introduction du site : http://alleg.sourceforge.net

A l'origine Allegro est écrite par Shawn Hargreaves pour Atari puis pour le compilateur DJGPP sous DOS dans un mélange de C et d'assembleur. Allegro en italien signifie «rapide, vivant, vif». Et fournit un acronyme récursif qui correspond à «**A**llegro **L**ow **LE**vel **G**ame **RO**utines» (routines de bas niveau pour les jeux).

Actuellement, environ 200 personnes participent activement à son développement http://alleg.sourceforge.net/thanks.fr.html De très nombreuses autres, utilisateurs attentifs, permettent de remonter des bugs afin qu'ils soient corrigés, ou contribuent à l'apport de nouvelles idées, ou laissent libre et ouvert le code source des applications qu'ils réalisent avec la librairie, ou encore écrivent quelques modules complémentaires à la librairie, des « add-on » que vous trouverez en grande partie sur le site http://www.allegro.cc.

Mailing listes, forums, wiki, documentation, exemples

Les mailings listes semblent de plus en plus désertées au profit des forums. Toutefois il y a plusieurs mailings listes toujours actives associées à la librairie : http://alleg.sourceforge.net/maillist.fr.html
La plus fréquemment utilisée était : [AL] Allegro

Cette liste est pour tout type de discussion sur Allegro : questions sur Allegro, rapports de problèmes avec Allegro, suggestions de nouvelles fonctionnalités, annonces de programmes écrits avec Allegro, etc. En règle générale si la question à quelque chose à voir avec Allegro, c'est la bonne liste.

Le forum est à l'adresse : http://www.allegro.cc/forums/. Il est divisé en trois parties essentielles. Ce qui concerne la librairie Allegro (rapports de bugs, questions techniques relatives à la librairie), ce qui concerne le développement de jeux (conception et architecture, questions de programmation) et un espace de discussion pour d'autres sujets toujours en relation.
D'autre part, une documentation claire est un aspect important d'un environnement de développement. Dans le dossier « doc » de la librairie une documentation est fournie et plusieurs formats sont disponibles (html, .doc etc.). Cette documentation existe aussi en ligne http://alleg.sourceforge.net/api.fr.html et http://fdrouillon.free.fr à la rubrique "tutorial". Vous pouvez également trouver un grand nombre d'articles concernant Allegro et son utilisation sur le Wiki Allegro à http://wiki.allegro.cc/.

Autre aspect très important d'un environnement de développement : les exemples de code. La librairie fournit un grand nombre d'exemples de code. Ils sont dans le dossier « example » de la librairie et illustrent le fonctionnement de la plupart des fonctions de la librairie.

Centralisation des ressources : http://fdrouillon.free.fr

Globalement, toutes les ressources nécessaires pour installer la librairie et l'utiliser à savoir compilateur, librairie compilée, exemples de code, documentation etc., ainsi que tous les exemples de code du présent tutorial sont regroupés sur le site http://fdrouillon.free.fr

Choisir son compilateur et son environnement de développement

Système d'exploitation et compilateur

Allegro est une librairie multi plates-formes et selon le système d'exploitation choisi différents compilateurs peuvent être utilisés ; la documentation Allegro en recense un certain nombre. Pour avoir des informations sur chacun des compilateurs les plus couramment utilisés avec Allegro suivre dans la documentation le lien « A general introduction to Allegro » et ensuite choisir son compilateur :

DOS/djgpp	- cf docs/build/djgpp.txt
DOS/Watcom	- cf docs/build/watcom.txt
Windows/MSVC	- cf docs/build/msvc.txt
Windows/MinGW	- cf docs/build/mingw32.txt
Windows/Cygwin	- cf docs/build/mingw32.txt
Windows/Borland	- cf docs/build/bcc32.txt
Linux (console)	- cf docs/build/linux.txt
Unix (X11)	- cf docs/build/unix.txt
Darwin (X11)	- cf docs/build/darwin.txt
BeOS	- cf docs/build/beos.txt
QNX	- cf docs/build/qnx.txt
MacOS X	- cf docs/build/macosx.txt

Dans ce tutorial nous avons choisi Windows comme système d'exploitation et les deux compilateurs MinGW et Visual C++ 8.

Environnement de développement

Pour être utilisable le compilateur est complété par un environnement de développement : EDI pour Environnement de Développement Intégré ou en anglais IDE pour Integrated Development Environment. C'est un programme qui regroupe un, éditeur de texte le compilateur, éventuellement des outils automatiques de fabrication, et souvent un débogueur
Visual C++ 8 a son propre IDE assez complexe et comprenant notamment des fonctions de recherche très puissantes et un excellent débogueur. Microsoft met à disposition gratuitement les versions

Express de Visual C++, C#, J#, Basic et SQL server. Elles sont téléchargeables ici :
http://msdn.microsoft.com/vstudio/express/downloads/default.aspx

Elles sont également distribuées aux étudiants et professeurs de l'ECE qui le souhaitent via ce lien :
http://elms.ece.fr ou
https://msdn20.e-academy.com/elms/Security/Login.aspx?campus=ece_si-rset

CodeBlock est un IDE distribué gratuitement avec le compilateur MinGW32 par défaut. Il est très simple d'utilisation et très complet. Il est téléchargeable sur http://fdrouillon.free.fr et sur son site de référence : http://www.codeblocks.org

Pour ce tutorial nous nous sommes appuyés sur codeBlock avec le compilateur MingW32. Cependant vous trouverez en annexe comment utiliser Visual Studio.

1. Démarrage

1.1 Installer la librairie

Pour installer la librairie dans codeBlock il y a deux possibilités, l'une, automatique est sous forme de "package Dev C++ ". Pour ce faire il faut utiliser le plugin "Dev-C++ DevPack updater/installer" accessible dans le menu "Plugins".

C'est toutefois l'autre méthode, manuelle, que nous allons détailler. Elle a l'avantage de fonctionner avec beaucoup de compilateurs différents. Elle consiste uniquement à recopier des fichiers des dossiers include et lib du répertoire de la librairie vers les dossiers include et lib du répertoire du compilateur.

Installer allegro manuellement à partir des fichiers

Sur le site http://www.allegro.cc/files/ à la rubrique « Binary » télécharger :
- le zip prévu pour MinGW et le zip Tools & examples .

Ou sur le site http://fdrouillon.free.fr à la rubrique « ressources » télécharger le package maison pour codeBlock. Ce package contient tout ce qui est nécessaire : exemples, doc, include, lib, tools et tests.

Ensuite il suffit :
1) De copier le contenu du dossier 'include' de la librairie dans le dossier 'include' du compilateur MinGW. MinGW se trouve dans le répertoire codeBlock (en principe le chemin complet est C:/Program Files/CodeBlocks/MinGW.

2) Dans le dossier 'lib' de la librairie allegro prendre les fichiers libaldat.a, liballd.a, liballd_s.a, liballeg.a, liballeg_s.a, liballp.a, liballp_s.a et les copier dans le dossier 'lib' de CodeBlocks/MinGW.

3) Copier la ou les DLLs (alleg42.dll, alld42.dll, allp42.dll) dans le dossier WINDOWS/system ou WINDOWS/system32 (pour XP, Vista). Eventuellement on peut préférer copier la DLL choisie dans le répertoire où se trouve le programme qui en a besoin (voir chapitre linkage)

Remarque :
Cette installation permet d'utiliser la librairie mais ne permet pas de compiler soi-même la librairie. Compiler soi-même la librairie est intéressant lorsque l'on souhaite suivre les étapes intermédiaires du développement de la librairie. Pour pouvoir compiler soi-même la librairie il faut encore ajouter dans les dossiers include et lib de CodeBlocks/MinGW le contenu des dossiers include et lib du SDK de directX pour Dev C++. Ces fichiers sont à télécharger sur www.allegro.cc et lors de l'installation ils doivent éventuellement écraser les fichiers du même nom des versions existantes déjà à l'installation de codeBlock.

1.2 Faire un projet

Pour faire un nouveau projet dans codeBlock aller dans le menu « fichier » sélectionner nouveau et ensuite nouveau projet, une fenêtre s'ouvre (figure 1) . Nous pouvons choisir l'un de ces trois types de projet :

- Console Application ou Empty Project
- Win32 GUI project

Les projets « Console application » et Empty project permettent d'avoir une fenêtre console en parallèle du fonctionnement du programme. Cette fenêtre est visible tant que le mode graphique d'allegro n'est pas actionné mais aussi si le programme est paramétré en mode fenêtre (ce point est abordé avec l'utilisation de la fonction set_gfx_mode). Cette fenêtre console peut être utilisée au début ou à la fin du programme par exemple, pour des informations sur le fonctionnement. « Empty project » est un projet console par défaut mais sans aucun fichier source fourni automatiquement au départ. C'est pratique lorsque l'on fait un programme à partir d'un fichier source déjà existant, avec son main() etc. Après la création du projet vide il suffit d'ajouter le fichier source, de linker (voir plus bas) et de compiler.

Si l'on choisit un projet « Win32 GUI project » il faut effacer entièrement la page de code généré automatiquement par codeBlock et remplacer ce code avec un main() spécifique au fonctionnement d'allegro (voir section « Premier programme »)

Une fois le type de projet spécifié cliquer sur "go".
L'étape suivante demande de choisir entre C et C++.
La troisième étape consiste à entrer un nom pour le projet et indiquer dans quel répertoire il doit être mis.

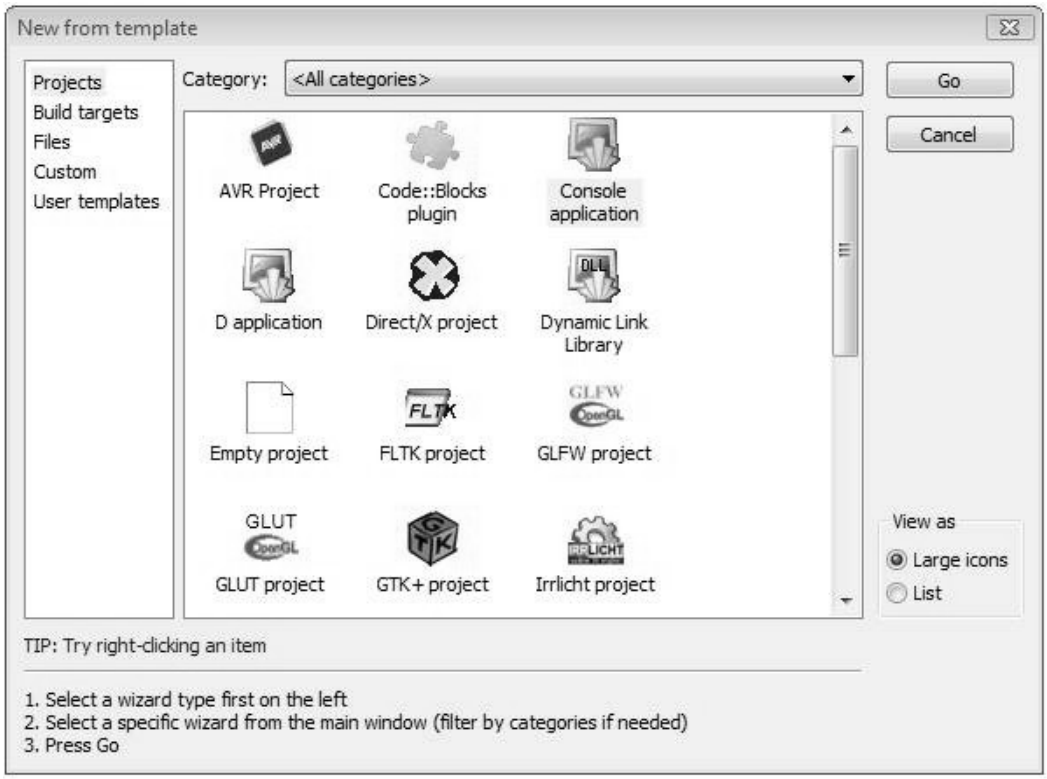


Fig 1 : faire un projet

Vient ensuite une quatrième et dernière étape (figure 2). Il s'agit de spécifier quel compilateur codeBlock va utiliser, si le projet comporte un mode "debug" et un mode "release". Le mode debug incorpore à la compilation du programme des informations qui permettent précisément de déboguer. C'est le mode utilisé pour le déboguage. Le mieux est de cocher les deux modes afin de pouvoir compiler le programme dans les deux modes ensuite.

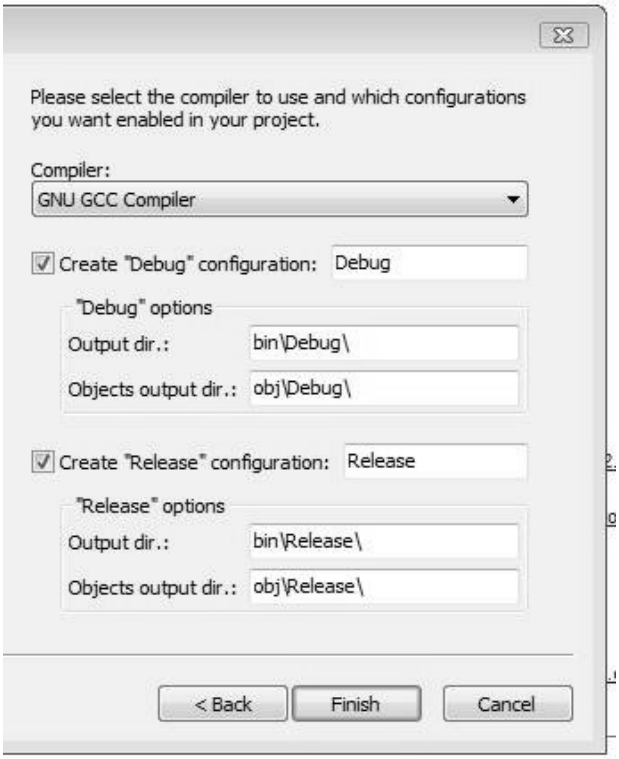


Fig 2 : configuration debug et release

1.3 Configurer le projet (linkage)

La traduction d'un programme source en exécutable passe par quatre grandes étapes.
1) Les algorithmes sont premièrement traduits en langage C ce qui se fait à avec un éditeur de texte. 2) Au moment de la compilation le programme source est d'abord traité par le préprocesseur qui remplace toutes les directives (#define, #include etc.) par leur contenu effectif de code source. 3) Le source d'origine ainsi retravaillé est alors présenté au compilateur. C'est une phase d'examination de chaque ligne du source avec vérification de la syntaxe et finalement la génération d'un code intermédiaire. 4) Lorsqu'il y a plusieurs fichiers source dans le programme les codes intermédiaires sont rassemblés et il reste à les LIER AUX LIBRAIRIES UTILISEES. C'est le rôle de l'éditeur de liens qui au final permet de produire le code exécutable complet du programme.

Les librairies, contenues dans le dossier lib du compilateur sont en réalité du code compilé et ce code compilé est en général complété par un ou plusieurs "headers" à savoir un ou plusieurs fichier.h qui contiennent toutes les déclarations des fonctions et définitions de type associés à la librairie. Ces fichiers, nécessaires pour pouvoir utiliser le code de la librairie, devront être inclus dans le code du projet.

Dans un projet l'opération de linkage va consister à signaler à l'éditeur de lien quelles sont les diverses librairies utilisées. C'est une opération en général très simple qui dépend de l'interface prévue dans le compilateur. Dans le cas de la librairie allegro il y a deux linkages possibles selon que l'on souhaite ou pas utiliser une DLL avec le programme. Le linkage fondé sur l'utilisation d'une DLL est dit « dynamique » (Dynamic, Linkage, Librairie) et sinon il est dit « statique ».

Link dynamique (avec DLL)

Le linkage dynamique suppose que le programme exécutable résultant va utiliser une DLL. Cette DLL devra être présente soit dans le dossier system32 de Windows XP/Vista, soit au même niveau que l'exécutable dans le même répertoire (le même dossier). Essentiellement la dll requise pour allegro version 4.2 stable est : "alleg42.dll", (elle se trouve dans le dossier lib du dossier de la librairie une fois compilée). L'avantage d'une dll est que plusieurs programmes peuvent utiliser la même dll ce qui

permet de diminuer la taille de chaque exécutable de la taille de la dll.

Pour obtenir la fenêtre de l'éditeur de lien dans codeBlock, aller au menu "project/built option". Une fenêtre s'ouvre (figure 3).

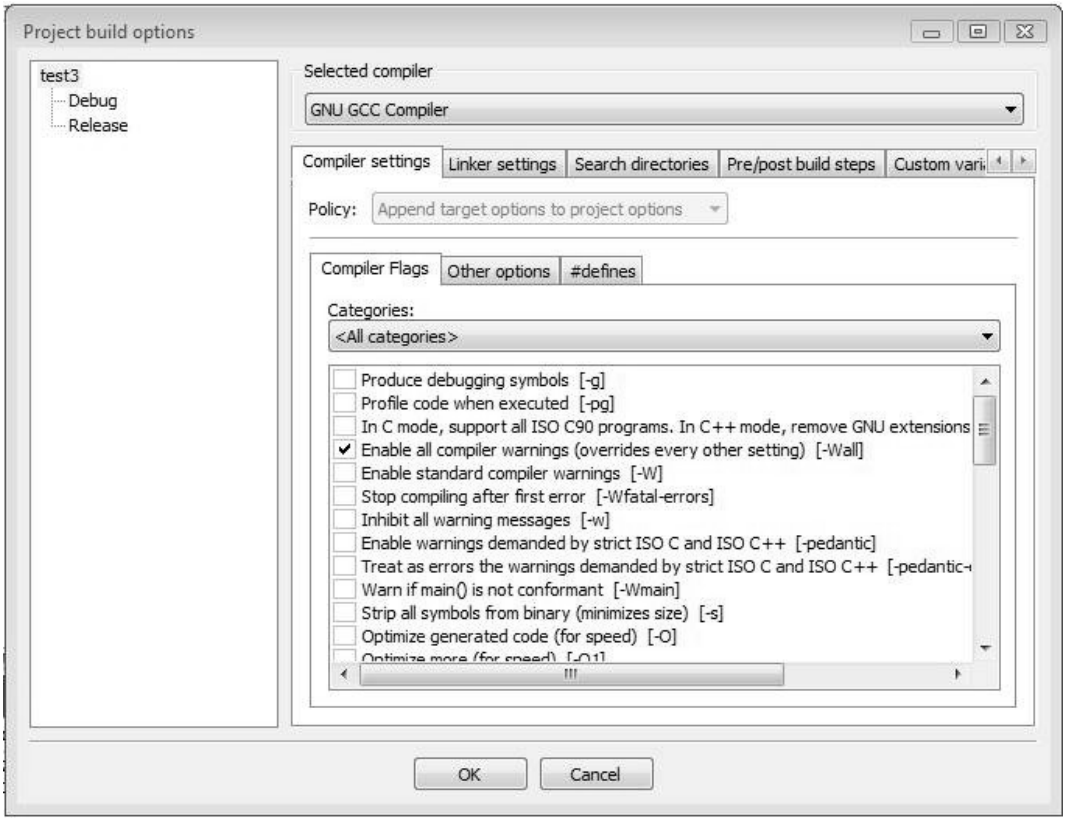


Figure 3 : fenêtre pour l'édition des options du compilateur

Selectionnez ensuite l'onglet "Linker settings" qui est à la droite de l'onglet actif "Compiler settings". La fenêtre de la figure 4 apparaît.

Dans la colonne de gauche sélectionnez le mode qui sera le sujet de linkage spécifié. Il y a trois possibilités : soit pour tout le projet (sans distinction release-debug) soit pour le mode debug, soit pour le mode release. En effet un linkage différent peut être spécifié selon le mode. Pour commencer nous utiliserons la première solution sans distinction entre les modes.

1) Sélectionnez le nom du projet dans la colonne de gauche ("test3" sur la figure 4).

2) Utilisez le bouton "add" au dessous de la colonne du centre nommée "Link libraries" pour entrer le nom des librairies nécessaires au projet. Pour un linkage dynamique d'allegro il suffit d'entrer **alleg**. Ce nom correspond au fichier "liballeg.a" qui doit se trouver dans le dossier lib du compilateur. Le préfixe "lib" à disparu ainsi que le suffixe ".a".

3) Cliquez sur "ok".

Remarque :
le fichier "liballeg.a" est celui utilisé pour le mode "release". Pour le mode "debug" il y a le fichier "liballd.a" prévu par la librairie. Pour dissocier les deux modes il suffit de sélectionner "release" dans la colonne de gauche et d'entrer "alleg" dans la colonne du centre, puis de sélectionner "debug" à gauche et d'entrer "alld" au centre.

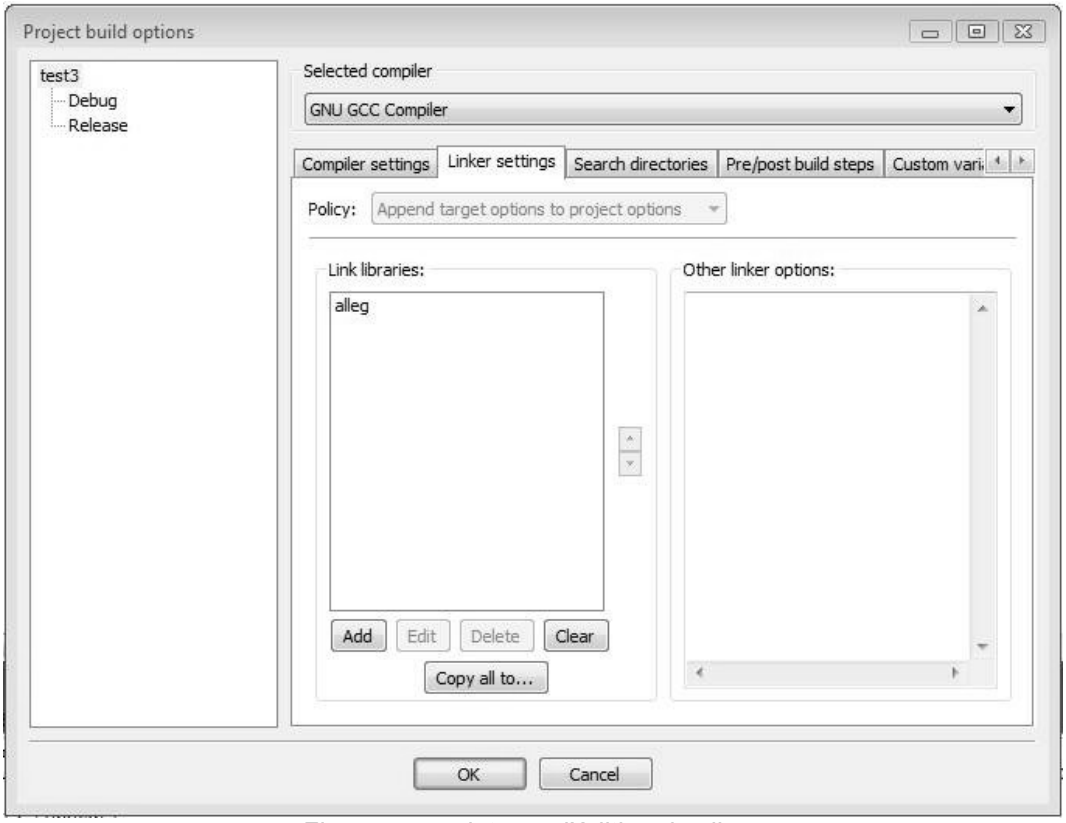


Figure 4 : onglet pour l'édition des liens

Link static (sans DLL)

A la différence du link dynamique la partie compilée de la librairie est incorporée dans le programme à la compilation. L'exécutable n'aura pas besoin d'une dll pour fonctionner. Inconvénient il ajoute à son propre poids de mémoire le poids de la librairie (Environ 500 k° pour allegro)

Méthode pour codeBlock :
Comme précédemment aller dans le menu "project/built option". Dans la première fenêtre, celle de la figure 3, selectionnez premièrement l'onglet "#define".

Dans le cadre de droite de cet onglet taper ALLEGRO_STATICLINK (figure 5)

Ensuite, comme précédemment sélectionnez l'onglet "Linker settings" et dans la colonne du centre entrer la liste suivante des librairies nécessaires :

alleg_s, gdi32, winmm, ole32, dxguid, dinput, draw, dsound,

Pour finir cliquer "OK"

Remarque :
Si ALLEGRO_STATICLINK n'est pas reporté dans la colonne "#define"de l'éditeur de liens il faut alors rajouter #define ALLEGRO_STATICLINK dans le source du programme avant l'inclusion de la librairie, y compris si l'inclusion de la librairie Allegro se fait dans une librairie personnelle,
comme ceci :

```
#define ALLEGRO_STATICLINK
#include <allegro.h>
```

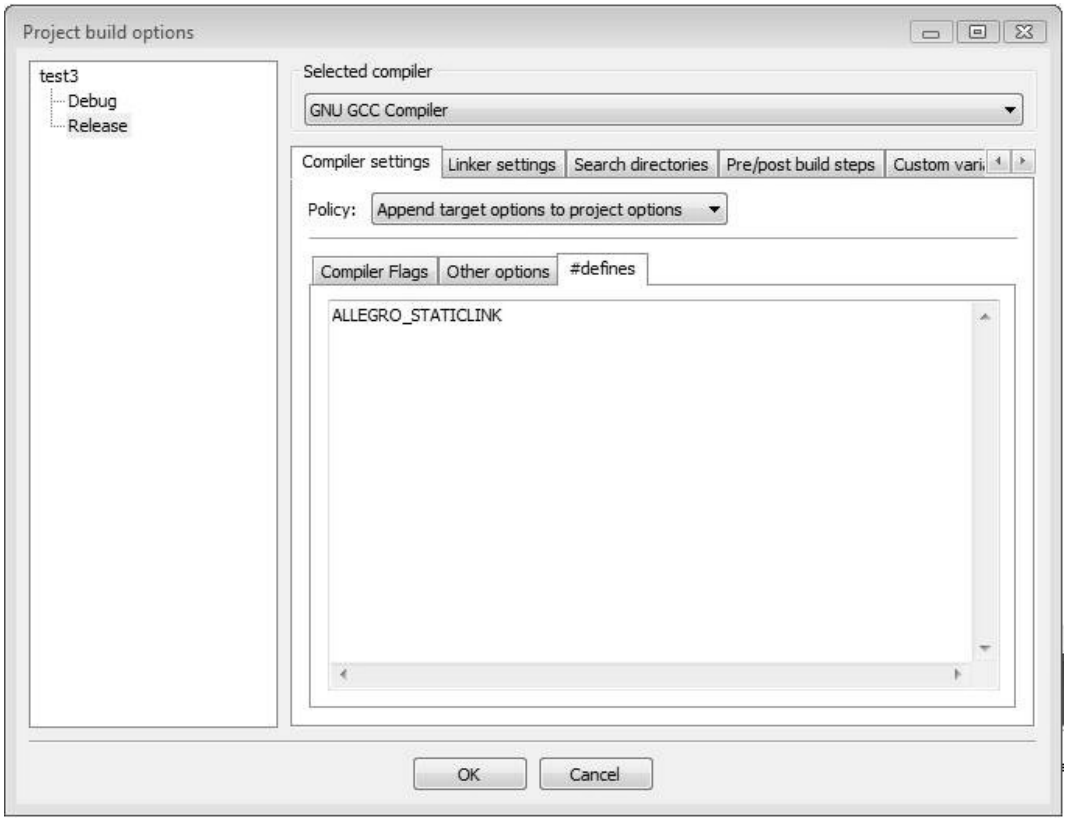


Figure 5 : spécifier un #define pour le linkage statique,

1.4 Premier programme

Test 1.1 : premier programme

```
    int main(void)
    {
(1)   allegro_init();
(2)   allegro_message(« OK ça marche ») ;
(3)   return 0;
    }
(4) END_OF_MAIN() ;
```

(1) Initialisation d'allegro obligatoire au début du programme

int allegro_init();
Comme on le voit le programme commence avec un appel à la fonction allegro_init(). En fait il s'agit d'une autre macro qui initialise la librairie (des variables sont créées, des infos sont récupérées etc.). C'est l'équivalent simplifié de l'appel :

```
install_allegro(SYSTEM_AUTODETECT, &errno, atexit);
```

C'est nécessaire d'appeler allegro_init() au début du programme, avant de faire quoique ce soit concernant des fonctions de la librairie. Seule exception, le cas où l'on souhaite modifier le codage du texte avec la fonction set_uformat() cette fonction doit alors être appelée avant allegro_init() (voir plus loin ou « unicode routines » dans la documentation allegro)

(2) Fenêtre avec message

void allegro_message(const char *msg, ...);
Cette fonction sort un message qui utilise une chaîne de caractères formatée comme printf. Si le système a une console, le message sera affiché sur la console. Dans un système fenêtré comme Windows, le message sera donné dans une fenêtre de dialogue.

ATTENTION :
Cette fonction entraîne une réinitialisation du mode graphique en mode texte. De ce fait, elle peut être utilisée soit avant un appel à set_gfx_mode() c'est-à-dire avant la sélection d'un mode graphique ou alors en mode texte après un appel à : set_gfx_mode(GFX_TEXT, …) , sinon il faut repasser en mode graphique après son utilisation (voir la section « Initialiser le mode graphique »).

(3) Fin habituelle d'un programme (non spécifique d'allegro)

Retour normal d'un programme qui se termine sans erreur.

(4) Macro END_OF_MAIN() obligatoire après le main

La compatibilité du main entre les différentes plateformes supportées par Allegro se traduit par l'utilisation d'une macro qui doit être placée tout à la fin de la fonction main, juste après la fermeture du bloc. Pour comprendre cette macro le mieux est d'en consulter le source dans la librairie. En fait la macro est remplacée par le main adéquat en fonction de la plateforme (windows et linux en ont besoin). En quelque sorte cette macro est le véritable main() du pogramme.

2. Entrer dans le mode graphique

2.1 Choisir et initialiser le mode graphique

L'initialisation du mode graphique s'occupe de sélectionner un driver pour la carte graphique avec le choix entre les modes plein écran ou fenêtre et le choix de la résolution. C'est un appel à la fonction :

int set_gfx_mode(int card, int w, int h, int v_w, int v_h);
1) Le paramètre "card" permet de sélectionner un driver graphique.
Les principaux drivers sont représentés par les macros suivantes :

GFX_AUTODETECT
Laisse allegro sélectionner un driver graphique approprié.

GFX_AUTODETECT_FULLSCREEN
Détecte automatiquement un driver graphique mais uniquement pour un plein écran, erreur s'il n'y en a pas d'utilisable pour le système.

GFX_AUTODETECT_WINDOWED
Idem précédent mais pour le mode fenêtre

GFX_SAFE
Driver spécial : allegro garantit que ce mode sera toujours installé correctement avec une résolution d'écran et un mode de couleur (8, 15, 16, 24, 32 bits) qui fonctionne. Au cas ou rien n'est possible sur la machine un nombre négatif est renvoyé. Si l'appel à set_gfx_mode() fonctionne avec ce driver on ne peut être sûr de la résolution de l'écran et du mode couleur, le code doit en tenir compte ensuite.

GFX_TEXT
Ce driver ferme tout éventuel mode graphique installé précédemment avec un appel de set_gfx_mode(), ce qui rend impossible l'utilisation de la variable globale "screen" (voir ci-dessous). Dans cet environnement non grahique les paramètres pour la taille de la bitmap screen sont inutiles,

on peut les mettre tous à 0 ce qui donne :

```
set_gfx_mode(GFX_TEXT,0,0,0,0) ;
```

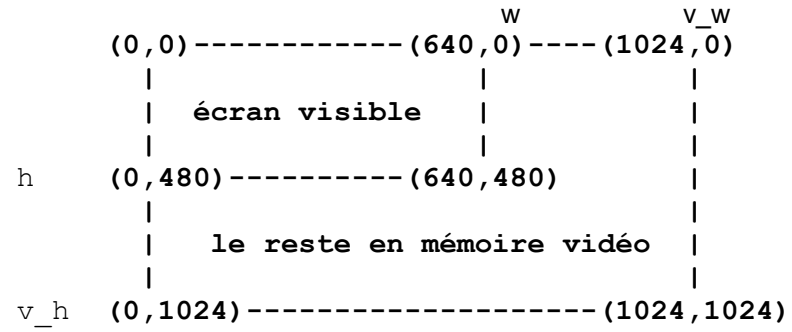
Cet appel, qui ferme le mode graphique, précède en général un appel à la fonction void allegro_message(const char *msg, ...);

GFX_GDI
Windows uniquement, c'est un mode fenêtre extrêmement lent mais qui garantit de fonctionner sur n'importe quel hardware. Utile dans des situations où l'on a besoin d'une fenêtre sans beaucoup de performance graphique.

D'autres possibilités qui dépendent de plateformes spécifiques sont indiquées dans la documentation à "platform specific"

2) Les paramètres w et h donnent la résolution de l'écran (tailles 640-480, 800-600, 1024-768 etc.).
Les paramètres v_w et v_h permettent de définir un écran virtuel plus large avec une taille supérieure à la résolution choisie. Ceci afin de mettre en place un scroll ou le principe de "page flipping" pour des animations. Si ces paramètres sont à 0 il n'y a pas d'écran virtuel. Toutes les cartes graphiques n'accepte pas d'écran virtuel. A noter également que les tailles horizontales et verticales de l'écran virtuel doivent toujours correspondre à des puissances de deux (512, 1024, 2048 etc.).

Exemple taille écran physique (w, h) allongée écran virtuel (v_w, v_h) :



3) La fonction initialise 5 variables globales :

screen : est un pointeur de type BITMAP* sur une bitmap de mémoire vidéo. Tout affichage écran se fait avec ce pointeur
SCREEN_W et **SCREEN_H** : correspondent aux valeurs de w et h pour la résolution de l'écran physique
VIRTUAL_W et **VIRTUAL_H** : correspondent aux valeurs de v_w et v_h pour la taille de l'écran virtuel

4) Si allegro ne peut pas sélectionner un mode graphique approprié, set_gfx_mode() retourne un nombre négatif et stocke une description du problème dans la chaine allegro_error (voir doc "using allegro"). Si ça marche set_gfx_mode() retourne 0

Contrôle d'erreur

Il est important lorsque l'on sélectionne un mode graphique de faire un contrôle d'erreur. Si la valeur de retour est différente de 0 la carte graphique ne supporte pas l'initialisation demandée et le programme ne peut pas fonctionner. Afin d'alléger le code nous allons nous écrire une macro de contrôle d'erreur et nous l'emploierons dans la suite du cours :

```
#define ERREUR(msg){\
   set_gfx_mode(GFX_TEXT,0,0,0,0);\
   allegro_message("err %s\nligne %d\nfile %s\n",msg,__LINE__,__FILE__);\
   allegro_exit();\
   return 1;\
}
```

Chaque ligne d'une macro doit être terminée par un \ et chaque ligne doit être complète, il n'est pas possible d'écrire une ligne sur deux lignes par exemple :

```
allegro_message(""err %s\nligne %d\nfile %s\n",msg,__LINE__,\
                                                __FILE__);\
```

ne fonctionne pas.
Une fois la macro ERREUR définie correctement, à chaque fois que nous écrirons ERREUR( ) en passant une chaine de caractère entre les parenthèses dans le programme, le code correspondant lui sera automatiquement substitué lors de la première étape de la compilation et la chaine de caractère sera affichée dans une dialogue boxe, avec le message passé, le numéro de la ligne au moment de l'appel et le fichier de code concerné.

Certaine fonction, comme set_gfx_mode, stocke une information sur une erreur arrivée dans la chaine de caractères allegro_error, dans ce cas il est utile de l'employer sinon rédiger son propre message en fonction de la situation rencontrée.

Test 1.2 : initialisation du mode graphique

```
#include <allegro.h>

// la macro pour le contrôle d'erreur
#define ERREUR(msg){\
   set_gfx_mode(GFX_TEXT,0,0,0,0);\
   allegro_message("err %s\nligne %d\nfile %s\n",msg,__LINE__,__FILE__);\
   allegro_exit();\
   return 1;\
}

int main()
{
   // initialisation allegro obligatoire
   allegro_init();

   // pour disposer du clavier (voir section clavier)
   install_keyboard();

   // définir un mode graphique
   if (set_gfx_mode(GFX_AUTODETECT_WINDOWED,800,600,0,0)!=0)
      // contrôler si le mode graphique fonctionne
      ERREUR(allegro_error);

   // attend une touche pour quitter (similaire getch() de
   // conio.h, voir section clavier, approche buffer)
   readkey();

   return 0;
}
END_OF_MAIN();  //attention ne pas oublier à la fin du bloc main
```

Dans cet exemple le programme initialise un mode graphique, une fenêtre Windows de 800 par 600 pixels, et attend ensuite qu'une touche du clavier soit pressée pour quitter (appel de la fonction readkey()). La fonction set_gfx_mode() retourne une valeur égale à 0 si l'opération a fonctionné mais si la valeur de retour est différente de 0 alors il y a un problème, les instruction du bloc du if , sous la forme de la macro ERREUR() sont exécutées : passage en mode texte, envoie d'une dialoque boxe avec un message pour l'utilisateur, allegro est quittée et le programme est forcé de terminer avec une indication d'erreur au système.

2.2 Avoir une couleur RGB

La couleur RGB résulte d'un mélange de rouge de vert et de bleu et il y a essentiellement deux cas pour opérer ce mélange. Le mode « vraie couleur » c'est-à-dire 15, 16, 24 ou 32 bits. Le mode 8 bits, 256 couleurs, géré avec une « palette ». Dans les deux cas les proportions de rouge, de vert et de bleu sont déterminées par des valeurs numériques et la couleur résultante est codée sur un entier (voir partie 2 sur la couleur)

Pour avoir facilement une couleur RGB utiliser la fonction :

int makecol(int r, int g, int b);
Cette fonction donne une couleur selon le mode couleur courant, 8, 15, 16, 24 ou 32 bits (par défaut 8 bits). Elle prend des valeurs de rouge, vert et bleu comprises dans la fourchette de 0 à 255.

2.3 Afficher du texte

Afficher du texte renvoie à quelques précisions :
- le format unicode
- les fontes
- affichage des lettres sur fond opaque ou transparent (texte mode)
- les fonctions d'affichage de chaînes non formatées
- les fonctions d'affichage de chaînes formatées du type printf
- avoir la taille d'une fonte

2.3.1 Le format UNICODE (doc : "Unicode routines")

Allegro permet des caractères de toutes les langues selon plusieurs options de codage à définir au lancement du programme par un appel à la fonction :

void set_uformat(int type);
Cette fonction détermine le format pour le codage du texte pendant le déroulement du programme. Ce format va permettre ou non les accents et différents types d'alphabets (grecs, chinois, arabe etc.). Par défaut Allegro utilise le format UTF_8 qui limite l'ascii aux 7 premiers bits du codage : il n'y a que les lettres latines et il n'y a pas d'accent.
Toutes les chaînes de caractères seront interprétées dans le type de format choisit y compris les chemins d'accès à d'éventuels fichiers dans le programme. Pour le français et ses accents :

U_ASCII	- 8 bits ASCII, ascii normal est idéal et suffisant

Autres possibilités :

U_ASCII_CP	- 8 bits codepage (voir set_ucodepage())
U_UNICODE	- 16 bits, caractères Unicodes
U_UTF8	- taille variable, format UTF-8 caractères Unicodes

Si cette fonction est utilisée il faut de préférence que ce soit avant l'appel de allegro_init() au début du programme. Il est possible de changer ce format à la volée dans le programme mais c'est déconseillé car des informations sous forme de chaîne de caractères sont initialisées à l'appel de allegro_init() au début du programme, par exemple en ce qui concerne des drivers du hardware, de plus le chemin d'accès aux fichiers peut également devenir problématique. Changer le format d'encodage du texte dans le programme risque de produire des erreurs.

2.3.2 Les fontes (doc : « Text output »)

Allegro fournit des fonctions d'affichage de texte qui supportent fontes monochrome et couleur et n'importe quel format unicode. Le programme « Grabber » qui est un utilitaire de la librairie, permet de créer des fontes à partir de caractères dessinés dans un fichier bitmap (voir grabber.txt). Il peut aussi importer des fichiers de fontes aux formats GRX et BIOS.

extern FONT *font;
Allegro fournit une fonte simple de taille 8x8 pixels (le mode 13h BIOS). Cette fonte contient les caractères ASCII standards (U+20 to U+7F), Latin-1 (U+A1 to U+FF), et Latin étendu-A (U+0100 to U+017F). C'est la fonte qui est utilisée dans tous les exemples du cours. Cependant le pointeur « font » peut être réaffecté à une fonte personnelle qui sera dans ce cas utilisée par Allegro notamment pour certaines fonctions relatives à l'interface (GUI routines).

2.3.3 Affichage de chaînes non formatées

int bg : détermine le mode d'affichage du texte
Pour toutes les fonctions d'affichage de texte le paramètre int bg détermine le mode d'affichage du texte. Si bg est supérieur ou égal à 0, le texte est opaque et le fond du texte prend la couleur donnée par bg. Si bg est négatif le texte est affiché et le fond est transparent. Par défaut bg est à 0 ce qui donne en général un fond noir pour le texte (en mode 8 bits palette 0 peut éventuellement correspondre à une autre couleur).

void textout_ex(BITMAP *bmp, const FONT *f, const char *s, int x, y, int color, int bg);
Cette fonction affiche dans la bitmap « bmp » avec la fonte « f » une chaîne de caractères non formatée « s » à partir de la position « x,y » et avec la couleur « color » spécifiées. Si la couleur est -1 et qu'une fonte colorée est utilisée (une fonte récupérée avec le grabber) ce sont les couleurs originales de la fonte qui seront utilisées (ce qui un moyen d'avoir du texte multicolore)

void textout_centre_ex(BITMAP *bmp, const FONT *f, const char *s, int x, y, color, int bg);
Idem textout() mais la coordonnée horizontale est interprétées comme le centre le la chaîne et non comme le bord gauche .

void textout_right_ex(BITMAP *bmp, const FONT *f, const char *s, int x, y, color, int bg);
Idem textout(), mais la coordonnée horizontale x correspond à la marge de droite et non celle de gauche.

void textout_justify_ex(BITMAP *bmp, const FONT *f, const char *s, int x1, int x2, int y, int diff, int color, int bg);
Affiche le texte justifié entre x1 et x2. Si la différence entre cet espace (x2-x1) et l'espace utilisé par les caractères (nombres de caractères multiplié par taille des lettres est inférieure à la valeur passée à l'argument diff l'affichage se fait simplement aligné à gauche.

2.3.4 Afficher un texte formaté

void textprintf_ex(BITMAP *bmp, const FONT *f, int x, y, color, int bg, const char *fmt, ...);
Il s'agit d'une fonction d'affichage de texte dans le style de printf() et qui use le même formatage des chaînes de caractères (%d etc.). La fonction affiche dans la bitmap « bmp », selon la fonte « f », à la position « x,y » et avec la couleur « color » la chaîne formatée « fmt ». Remarque : le caractère '\n ' de retour chariot pour ligne suivante est inopérant en mode graphique. Pour changer de ligne il faut découper son texte en ligne et jouer sur les positions x et y de plusieurs appels à textprintf()

void textprintf_centre_ex(BITMAP *bmp, const FONT *f, int x, y, color, int bg, const char *fmt, ...);
Idem textprintf(), mais la coordonnée horizontale est interprétées comme le centre le la chaîne et non comme le bord gauche.

void textprintf_right_ex(BITMAP *bmp, const FONT *f, int x, y, color, int bg, const char *fmt, ...);
Idem textprintf(), mais la coordonnée horizontale x correspond à la marge de droite et non celle de gauche.

void textprintf_justify_ex(BITMAP *bmp, const FONT *f, int x1, int x2, int y, int diff, int color, int bg, const char *fmt, ...);
Idem textout_justify, mais utilise le formatage des chaînes de caractères du type printf().

2.3.5 Longueur d'une chaîne et hauteur d'un caractère dans une fonte donnée

int text_length(const FONT *f, const char *str);
Retourne la longueur en pixel d'une chaîne de caractère et selon la fonte spécifiée.
int text_height(const FONT *f)
Retourne la hauteur en pixels de la fonte spécifiée.

2.4 Primitives de dessin

Toujours dans le domaine de l'affichage Allegro propose un ensemble de fonctions de dessin. Exemples :

void putpixel(BITMAP *bmp, int x, int y, int color);
Ecrit la couleur « color » d'un pixel (x,y) d'une image bitmap "bmp" dans le mode couleur courant du programme et à l'intérieur du clip de l'image.

void vline(BITMAP *bmp, int x, int y1, int y2, int color);
Trace sur "bmp" une ligne verticale de (x, y1) à (x, y2) de la couleur color.

void hline(BITMAP *bmp, int x1, int y, int x2, int color);
Trace sur "bmp" une ligne horizontale de (x1, y) à (x2, y) de la couleur color.

void line(BITMAP *bmp, int x1, int y1, int x2, int y2, int color);
Trace sur "bmp" une ligne du point (x1, y1) au point (x2, y2) de la couleur color.

void rect(BITMAP *bmp, int x1, int y1, int x2, int y2, int color);
Trace sur « bmp » le tour d'un rectangle à partir des points des deux coins opposés haut-gauche et bas-droite de la couleur color.

void rectfill(BITMAP *bmp, int x1, int y1, int x2, int y2, int color);
Rempli sur « bmp » le rectangle construit à partir des points des deux coins opposés haut-gauche et bas-droite de la couleur "color".

void triangle(BITMAP *bmp, int x1, y1, x2, y2, x3, y3, int color);
Trace et colore sur « bmp » un triangle entre les trois points de la couleur color.

void polygon(BITMAP *bmp, int vertices, const int *points, int color);
Trace et colore sur « bmp » un polygone de "vertices" sommets. Chaque sommets est un point de coordonnées (x,y). Ils sont stocké dans le tableau "int *points" dont la taille sera vertices*2.

void floodfill(BITMAP *bmp, int x, int y, int color);
Sur « bmp » colore une zone fermée à partir d'un point (x,y) dedans avec la couleur spécifiée

Etc. D'autres fonctions de dessin sont disponibles pour les cercles, les arcs, les splines (courbes de Bézier). Certaines emploient des pointeurs de fonction (doc : "Drawing primitives).

Test 1.3 : fonctions texte et primitives de dessin

```
#include <allegro.h>

#define ERREUR(msg){\
   set_gfx_mode(GFX_TEXT,0,0,0,0);\
   allegro_message("err %s\nligne %d\nfile %s\n",msg,__LINE__,__FILE__);\
   allegro_exit();\
   return 1;\
}

int main(int argc, char *argv[])
{
```

```
int c,x,y,tx,ty;

   // pour disposer du mode ascii complet (avec accents)
   set_uformat(U_ASCII);
   allegro_init();
   install_keyboard();

   if (set_gfx_mode(GFX_AUTODETECT_WINDOWED,800,600,0,0)!=0)
      ERREUR(allegro_error);

   // Avoir une couleur (ici du blanc)
   c=makecol(255,255,255);

   // afficher du texte
   textout_ex(screen,font,"texte non formate",10,50,c,-1);
   textout_ex(screen,font," éèàùïôê accents ? oui si",10,70,c,-1);
   textout_ex(screen,font,"appel de set_uformat() au début",10,85, c,-1);
   textprintf_ex(screen,font,40,100,c,-1,"format:%d %d",SCREEN_W,SCREEN_H);

   // afficher des formes : primitives de dessin
   c=makecol(255,255,255);
   rectfill(screen, 50,110,50+200,110+70,c);
   readkey();

   // test rect et cercle, remplis ou contours
   while (!keypressed()){
      y=250+rand()%(SCREEN_H-400);
      x=rand()%SCREEN_W;
      tx=rand()%100;
      ty=rand()%100;
      c=makecol(rand()%256, rand()%128,rand()%64);
      switch(rand()%4){
         case 0 : rect(screen, x,y,x+tx,y+ty,c);      break;
         case 1 : rectfill(screen, x,y,x+tx,y+ty,c);  break;
         case 2 : circle(screen, x,y,tx,c);           break;
         case 3 : circlefill(screen, x,y,tx/3,c);     break;
      }
      rest(100);
   }
   return 0;
}
END_OF_MAIN();
```

3. Enter dans l'événementiel

La boucle d'événements est souvent la boucle principale du programme qui permet de capturer les actions de l'utilisateur (les événements) ou de réitérer des traitements en attendant un arrêt du programme par l'utilisateur. Les événements fondamentaux sont en général les entrées souris et clavier. Le programme détecte chaque mouvement de a souris et chaque touche pressée du clavier. D'autres type d'entrées existent en particulier allegro permet d'utiliser des joysticks et il est possible de se relier à des capteurs. Mais pour commencer il est indispensable de maîtriser les entrées souris et clavier afin d'actionner des commandes.

3.1 Contrôle de la Souris

Nous présentons ici les caractéristiques principales de la souris : installation, position, clics. D'autres

possibilités sont présentées dans la documentation « Mouse routines » d'Allegro.

int install_mouse();
Installe le "Allegro mouse handler", boucle parallèle pour une mise à jour permanente de l'état de la souris. L'appel de cette fonction est obligatoire pour pouvoir utiliser des fonctions pour la souris. Cette fonction renvoie -1 en cas d'échec et sinon le nombre de bouton de la souris.

Cinq variables globales sont générées par allegro. Réactualisées en permanence de façon asynchrone elles contiennent toutes les infos utiles pour l'utilisation de la souris :

extern volatile int mouse_x;
extern volatile int mouse_y;
mouse_x et mouse_y indique la position à l'écran comprise entre 0 et, selon la résolution, le coin en bas à droite.

extern volatile int mouse_z;
mouse_z indique la position de la roue ("wheel") s'il y en a une.

extern volatile int mouse_b;
mouse_b est un champ de bits qui indique l'état de chaque bouton. Le bit 0 pour le bouton gauche, le bit 1 pour le bouton droit, le bit 2 pour le bouton du milieu.

Pour connaître la valeur d'un bit on utilise l'opérateur bit à bit & ("et"). Cet opérateur permet de faire des tests sur l'état de chaque bit d'une variable de type char, short, int, long, double. Par exemple, soit un char c constitué de 8 bits : b7b6b5b4b3b2b1b0 chacun a une valeur de 0 ou 1 c.a.d que bi=0 ou bi=1. Le fonctionnement de l'opérateur est :

bi & 0 = 0 // quelque soit la valeur initiale de bi (si bi vaut 1 ça fait 0 et si bi vaut 0 ça fait 0)
bi & 1 = bi // quelque soit la valeur initiale de bi (si bi vaut 1 ça fait 1 et si bi vaut 0 ça fait 0)

Le "et" est effectué bit à bit sur tous les bits de c à partir d'une valeur donnée appelée souvent un « masque ». Prenons par exemple pour masque la valeur m= 4, m=00000100 en binaire. Si i vaut 5, i=00000101 en binaire ainsi l'expression i & 4 donne 1, en effet :

i0 & m0 donne 1 & 1 soit 1
i1 & m1 donne 0 & 0 soit 0
i2 & m2 donne 1 & 0 soit 0
i3 & m3 donne 0 & 0 soit 0
etc.

Selon ce procédé on peut connaître l'état d'un clic avec l'opérateur & de la façon suivante :

```
if (mouse_b & 1)//info clic gauche dans le premier bit
   printf("bouton gauche presse\n");
if (mouse_b & 2)//info clic droit dans le deuxième bit
   printf("bouton droit presse\n");
```

extern volatile int mouse_pos;
mouse_pos donne la position horizontale à l'écran dans la partie haute de la variable et la position verticale y dans la partie basse de la variable. Ce peut être pratique par exemple pour savoir avec un seul test si la position de la souris a été modifiée ou non

Test 1.4 : contrôle de la souris

```
#include <allegro.h>

#define ERREUR(msg){\
   set_gfx_mode(GFX_TEXT,0,0,0,0);\
   allegro_message("err %s\nligne %d\nfile %s\n",msg,__LINE__,__FILE__);\
   allegro_exit();\
```

```
   return 1;\
}

int main(int argc, char *argv[])
{
int c;
int x,y,fin;

   allegro_init();

   // pour disposer de la souris
   install_mouse();

   if (set_gfx_mode(GFX_AUTODETECT_WINDOWED,800,600,0,0)!=0)
      ERREUR(allegro_error);

   // contrôle souris
   x=y=fin=0;

   // pour voir la souris
   show_mouse(screen);

   while (!fin){

      // si mouvement souris
      if (mouse_x!=x || mouse_y!=y){
         // effacer txt ancienne position
         textprintf_ex(screen,font,60,300,makecol(0,0,0),-1,
         "%d %d",x,y);
         // prendre nouvelle position
         x=mouse_x;
         y=mouse_y;
         // afficher nouvelle position
         textprintf_ex(screen,font,60,300,makecol(0,255,0),-1,
         "%d %d",x,y);
         // test d'arrêt : x à gauche
         if (x==0)
            fin=1;
      }

      // les clics
      if (mouse_b & 1) // gauche
         rectfill(screen,x, y, x+20, y+20, makecol(255,0,0));
      if (mouse_b & 2) // droit
         rectfill(screen,x, y, x+20, y+20, makecol(0,0,255));
      // milieu ou les deux à la fois fin du programme
      if (mouse_b &4)
         fin=1;
   }
   return 0;
}
END_OF_MAIN();
```

3.2 Contrôle du clavier

Nous présentons ici les caractéristiques principales de l'utilisation du clavier, soit deux approches : par scancode ou par buffer comme en mode console.

int install_keyboard()
Installe le "Allegro keyboard interrupt handler", boucle parallèle pour une mise à jour permanente de l'état du clavier. L'appel de cette fonction est obligatoire pour rendre possible l'utilisation d'entrées clavier dans le programme. La fonction retourne un nombre négatif en cas d'échec et 0 pour réussite.

Allegro propose deux entrées : d'une part un classique buffer clavier et d'autre part un ensemble de booléens qui donne l'état de chaque touche (scancode)

3.2.1 Approche du clavier par scancode

La librairie allegro stocke dans un tableau accessible en globale l'état de chaque touche du clavier 0 elle n'est pas pressée, 1 elle est pressée. C'est le tableau :

extern volatile char key[KEY_MAX];
A chaque indice du tableau correspond une touche du clavier définit par une macro constante dans l'ordre suivant :
#define KEY_A 1
#define KEY_B 2
#define KEY_C 3
#define KEY_D 4
#define KEY_E 5
etc ...KEY_Z

KEY_0 ... KEY_9,

KEY_0_PAD ... KEY_9_PAD,
KEY_F1 ... KEY_F12,

KEY_ESC, KEY_TILDE, KEY_MINUS, KEY_EQUALS, KEY_BACKSPACE, KEY_TAB,
KEY_OPENBRACE, KEY_CLOSEBRACE, KEY_ENTER, KEY_COLON, KEY_QUOTE,
KEY_BACKSLASH, KEY_BACKSLASH2, KEY_COMMA, KEY_STOP, KEY_SLASH,
KEY_SPACE,

KEY_INSERT, KEY_DEL, KEY_HOME, KEY_END, KEY_PGUP,
KEY_PGDN, KEY_LEFT, KEY_RIGHT, KEY_UP, KEY_DOWN,
KEY_SLASH_PAD, KEY_ASTERISK, KEY_MINUS_PAD,
KEY_PLUS_PAD, KEY_DEL_PAD, KEY_ENTER_PAD,
KEY_PRTSCR, KEY_PAUSE,

KEY_ABNT_C1, KEY_YEN, KEY_KANA, KEY_CONVERT, KEY_NOCONVERT,
KEY_AT, KEY_CIRCUMFLEX, KEY_COLON2, KEY_KANJI,
KEY_LSHIFT, KEY_RSHIFT,
KEY_LCONTROL, KEY_RCONTROL,
KEY_ALT, KEY_ALTGR,
KEY_LWIN, KEY_RWIN, KEY_MENU,
KEY_SCRLOCK, KEY_NUMLOCK, KEY_CAPSLOCK

Ces constantes sont les « scancodes » des caractères et plus largement de l'ensemble des touches du clavier. Pour savoir si une touche est pressée il suffit de faire un test sur la valeur correspondante dans le tableau key[] :

```
if (key[KEY_SPACE])
   printf("la touche espace est pressee\n");
```

Contrôler une Boucle d'événements

Typiquement sous allegro le scancode code de la touche escape est utilisé pour contrôler une boucle d'événements. La boucle d'événement est la boucle principale du programme qui permet de capturer les actions de l'utilisateur ou de réitérer des traitements en attendant un arrêt du programme par l'utilisateur. Dans l'exemple ci-dessous il n'y a pas d'événement et la boucle est vide. Le programme

installe le clavier, passe en mode graphique puis attend que l'utilisateur tape échapp pour s'arrêter.

```
int main()
{
   init_allegro() ;
   install_keyboard() ;
   if (set_gfx_mode(GFX_AUTODETECT,640,480,0,0)!=0)
      ERREUR(allegro_error);

   // la boucle d'évènements
   while ( !key[KEY_ESC]) {

      // ici placer les actions à faire

   }
   exit(EXIT_SUCCESS);
}
END_OF_MAIN() ;
```

La vitesse du mode scancode

Mais attention, le contrôle du clavier via scancodes est extrêmement rapide ! Pendant le temps où le doigt est appuyé sur la touche on va probablement passer de nombreuses fois dans le test et non pas une seule.

Dans le programme ci-dessous les flèches pilotent l'affichage de carrés de couleurs. Pendant le temps même très court où le doigt est appuyé on voit la couleur clignoter, et lorsque le doigt est relevé la dernière couleur est visible. La touches F1 affiche un petit compteur du nombre de passage dans le if pendant que le doigt est appuyé. La touche F2 remet ce compteur à 0.

Test 1.5 : contrôle clavier par scancodes

```
#include <allegro.h>

#define ERREUR(msg){\
   set_gfx_mode(GFX_TEXT,0,0,0,0);\
   allegro_message("err %s\nligne %d\nfile %s\n",msg,__LINE__,__FILE__);\
   allegro_exit();\
   return 1;\
}

int main(int argc, char *argv[])
{
int c,x,y,cmpt=0;

   allegro_init();
   //pour disposer du clavier
   install_keyboard();

   if (set_gfx_mode(GFX_AUTODETECT_WINDOWED,800,600,0,0)!=0)
      ERREUR(allegro_erreur);

   while (!key[KEY_ESC]){      // la boucle d'évènements

      // clavier via SCANCODES
      if (key[KEY_UP])
         rectfill(screen,350,270,450,370,makecol(rand()%256,0,255));
      if (key[KEY_DOWN])
         rectfill(screen,350,270,450,370,makecol(0,255,rand()%256));
```

```
      if (key[KEY_LEFT])
         rectfill(screen,350,270,450,370,makecol(255,rand()%256,0));
      if (key[KEY_RIGHT])
         rectfill(screen,350,270,450,370,makecol(0,0,0));

      // test pour vitesse SCANCODE
      if (key[KEY_F1]){
         textprintf_ex(screen,font,460,320,makecol(0,0,0),-1,
         "%d",cmpt);
         cmpt++;
         textprintf_ex(screen,font,460,320,makecol(0,255,0),-1,
         "%d",cmpt);
      }
      if (key[KEY_F2]){
         textprintf_ex(screen,font,460,320,makecol(0,0,0),-1,
         "%d",cmpt);
         cmpt=0;
         textprintf_ex(screen,font,460,320,makecol(255,0,0),-1,
         "%d",cmpt);
      }
   }
   return 0;
}
END_OF_MAIN();
```

3.2.2 Approche du clavier par buffer, fonctions associées

int keypressed();
Retourne TRUE s'il y a des touches pressées dans la pile du buffer clavier et FALSE sinon.
Elle est équivalente à la fonction kbhit() (de conio.h)

int readkey()
Retourne le caractère courant du buffer de caractère (comme la fonction getch() de la librairie conio.h)
Si le buffer est vide elle attend une entrée de caractère (une touche clavier pressée).

Le premier octet (low byte) de la valeur retournée contient la valeur ASCII de la touche, le second (hight byte) la valeur en scancode. Il y a possibilité d'identifier la touche pressée selon ces deux codages de la façon suivante :

```
   if ((readkey() & 0xff) == 'd')    // le codage ASCII
      printf("vous appuyez 'd'\n");
   // le codage scancode est récupéré avec un décalage de 8 bits à droite
   if ((readkey() >> 8) == KEY_SPACE)
      printf("vous appuyez espace\n");
```

Pour obtenir ce décalage l'opérateur de décalage bit à bit >> est utilisé qui translate de n positions sur la droite les bits d'une variable en remplissant de 0 sur la gauche. L'expression « readkey()>>8 » signifie que tous les bits du résultat renvoyé par readkey() se décalent de 8 bits sur la droite, ceux qui manquent à gauche sont remplacés par des 0.

readkey() ne peut pas retourner de valeurs de caractère plus grande que 255. Pour d'autres entrées de type "Unicode", codées sur 16 bits, il faut utiliser la fonction ureadkey()

void set_keyboard_rate(int delay, int repeat);
Sert à paramétrer la vitesse de répétition des touches du clavier pendant le temps où la touche est appuyée. Le temps est donné en millisecondes, passer 0 temps annule toute répétition et pour répéter la touche il faudra d'abord en avoir relevé le doigt.
Attention, cette fonction ne change rien au comportement de l'approche par SCANCODES vue précédemment.

Test 1.6 : contrôle clavier via buffer

```
#include <allegro.h>

#define ERREUR(msg){\
   set_gfx_mode(GFX_TEXT,0,0,0,0);\
   allegro_message("err %s\nligne %d\nfile %s\n",msg,__LINE__,__FILE__);\
   allegro_exit();\
   return 1;\
}

int main(int argc, char *argv[])
{
int c,x,y,fin=0,cmpt=0,choix;

   allegro_init();
   install_keyboard();

   if (set_gfx_mode(GFX_AUTODETECT_WINDOWED,800,600,0,0)!=0)
      ERREUR(allegro_erreur);

   // Pour clavier via BUFFER, la fonction set_keyboard_rate()
   // permet le contrôle de la répétition des touches du clavier :
   // après p1 temps en milliseconde chaque répétition a lieu tout
   // les p2 millisecondes. Passer 0,0 interdit la répétition de
   // touche pour le buffer clavier
   set_keyboard_rate(1000,100);
   while (!fin){

      // clavier via BUFFER (syle mode console avec conio.c)

      // keypressed() équivalent kbhit()
      if (keypressed()){
         // récupérer la touche avec readkey() équivalent getch()
         choix=readkey();

         // aiguillage selon la touche en considérant la valeur
         // scancode qui est stockée sur le 2e octet. Le premier
         // pour la valeur ascii s'obtient avec le masque 0xFF :
         // (choix & 0xFF).
         switch(choix>>8){

            case KEY_F3 :
               rectfill(screen,350,100,450,200,makecol(rand()%255,0,255));
               break;

            case KEY_F4 :
               rectfill(screen,350,100,450,200,makecol(0,255,rand()%255));
               break;

            case KEY_F5 :
               rectfill(screen,350,100,450,200,makecol(255,rand()%255,0));
               break;

            case KEY_F6 :
               rectfill(screen,350 ,100 , 450, 200, makecol(0,0,0));
               break;

            // contrôle de la boucle d'événements
            case KEY_SPACE : fin=1;
               break;
```

```
            default : break;
         }
      }
   }
   return 0;
}
END_OF_MAIN();
```

3.3 Bouger une forme à l'écran

Le programme suivant permet de bouger une forme à l'écran en utilisant les flèches.

Test 1.7 : bouger une forme à l'écran

```
#include<allegro.h>

#define ERREUR(msg){\
   set_gfx_mode(GFX_TEXT,0,0,0,0);\
   allegro_message("err %s\nligne %d\nfile %s\n",msg,__LINE__,__FILE__);\
   allegro_exit();\
   return 1;\
}

int main()
{
int done=0;        // pour fin boucle
int x,y;           // coordonnées forme
int tx,ty;         // taille forme
int px,py;         // pas d'avancement en x et y forme
int color;         // couleur forme
int choix;         // pour choix entrée utilisateur

   allegro_init();
   install_keyboard();
   srand(time(NULL));
   if (set_gfx_mode(GFX_AUTODETECT,ECRAN_X,ECRAN_Y,0,0)!=0)
      ERREUR(allegro_error);

   // initialisation position et taille de la forme
   tx=rand()%20+20;
   x=rand()%(ECRAN_X-tx*2);
   x+=tx;
   px=(rand()%5)+2;
   ty=rand()%20+20;
   y=rand()%(ECRAN_Y-ty*2);
   y+=ty;
   py=(rand()%5)+2;
   // affichage départ (un rectangle)
   color=makecol(255,0,0);
   rectfill(screen,x,y,x+tx,y+ty,color);

   // boucle événements
   while (!done){
      if (keypressed()){
         //effacement aux anciennes coordonnées
         rectfill(screen,x,y,x+tx,y+ty,makecol(0,0,0));
         // recup choix
         choix=readkey();
```

```
        switch (choix>>8){  // recup valeur sous forme scancode
           case KEY_UP :        y-=py;     break;
            case KEY_DOWN :     y+=py;     break;
            case KEY_LEFT :     x-=px;     break;
            case KEY_RIGHT :    x+=px;     break;
            case KEY_ESC :      done=1;   break;// pour test d'arrêt
            default :                      break;
        }
        // contrôle des bords, utilisation opérateur conditionnel
        x=(x+tx<0)    ? ECRAN_X-1 : x;
        x=(x>ECRAN_X) ? -tx+1     : x;
        y=(y+ty<0)    ? ECRAN_Y-1 : y;
        y=(y>ECRAN_Y) ? -ty+1     : y;

        // affichage à l'écran
        rectfill(screen,x,y,x+tx,y+ty,color);
      }
   }
   exit(EXIT_SUCCESS);
}
END_OF_MAIN();
```

Résumé C1

PROJET
console
Windows : remplacer le winmain par un main() allegro

LINKAGE
Menu Projet / build options / Linker setting / colonne link libraries
Pour une version dynamique (DLL) entrer :
alleg

pour une version static de la librairie entrer chacune des librairies :
alleg_s, gdi32, winmm, ole32, dxguid, dinput, ddraw, dsound
(remarque en cas de perte de cette liste ouvrir le fichier makefile.mgw qui est à la racine du répertoire de la librairie allegro et chercher la liste après LIBRARIES =)

Ensuite dans le menu Projet / build options / #define
entrer ALLEGRO_STATICLINK
ou taper :

```
#define ALLEGRO_STATICLINK // juste avant allego.h dans le code du
#include <allegro.h>       // programme
```

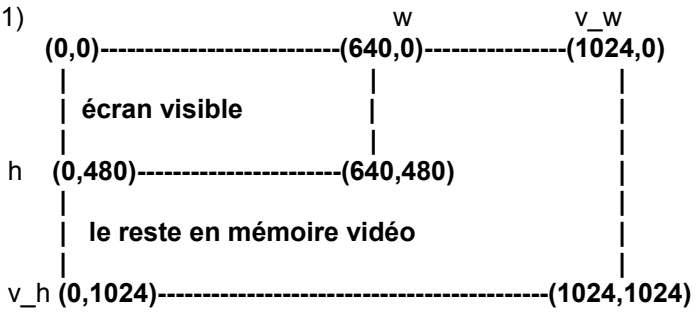
PREMIER PRG

Si linkage static ne pas oublier avant include librairie :
```
#define ALLEGRO_STATICLINK
```

ensuite :
```
#include <allegro.h>
int main()
{
   allegro_init();
   allegro_message("bonjour");
   return 0;

} END_OF_MAIN() ;
```

INITIALISATION MODE GRAPHIQUE

Utiliser la fonction :
int set_gfx_mode(int card, int w, int h, int v_w, int v_h);



2) Cinq gobales :
- **screen** pour affichage à l'écran
- **SCREEN_W** et **SCREEN_H** (résolution)
- VIRTUAL_W et VIRTUAL_H (taille écran virtuel)

3) Un ensemble de #define correspondant à des drivers graphiques
GFX_AUTODETECT // le plus commode
GFX_AUTODETECT_FULLSCREEN
GFX_AUTODETECT_WINDOWED
GFX_SAVE
GFX_TEXTE
GFX_GDI // windows uniquement

AVOIR UNE COULEUR RGB

La fonction :
int makeacol(int r, int g, int b); donne une couleur selon le mode couleur courant, 8, 15, 16, 24 ou 32 bits (par défaut 8 bits). Elle prend des valeurs de couleur entre 0 et 255

AFFICHER DU TEXTE

Le paramètre int bg décide de la couleur de fond du texte : si négatif, fond transparent (pas de couleur du fond) si 0 fond noir, si supérieur à 0 le nombre va correspondre à une couleur.

- Texte formaté :
  void textprintf_ex(BITMAP *bmp, const FONT *f, int x, y, color, int bg, const char *fmt, ...);
  Plus les fonctions dérivées (centré, justifié, drapeau droit)

- texte non formaté :
  void textout_ex(BITMAP *bmp, const FONT *f, const char *s, int x, y, int color,int bg);
  Plus les fonctions dérivées (centré, justifié, drapeau droit)

A propos de la couleur de fond pour chaque chaîne de caractères affichée :
Si bg > 0 le fond du texte prend la couleur indiquée par bg
SI bg < 0 le fond est transparent
Par défaut mode = 0 ce qui donne un fond noir pour le texte.

FONTE PAR DEFAUT

Extern FONT *font ; // le pointeur peut être réaffecté à une autre fonte

Caractéristiques : 13h bios type 8 x 8 taille des caractères latin étendu

CODAGE LETTRES-FORMATS UNICODE

void set_uformat(int type);
Le type correspond à une des valeurs :

U_ASCII	- fixed size, 8 bit ASCII characters	// idéal pour français
U_ASCII_CP	- alternative 8 bit codepage (see set_ucodepage())	
U_UNICODE	- fixed size, 16 bit Unicode characters	
U_UTF8	- variable size, UTF-8 format Unicode characters	

ATTENTION : APPEL AVANT allegro_init() ; afin de déterminer le codage des caractères qui sera utilisé pendant le déroulement du programme (risque problème sinon)

SOURIS

int install_mouse();
à appeler après allegro_init() ;

3 variables globales essentielles :
extern volatile int mouse_x;
extern volatile int mouse_y;
extern volatile int mouse_b;

```
Récupérer un clic :
if (mouse_b & 1)
   // clic gauche
if (mouse_b & 2)
   // clic droit
```

CLAVIER

int install_keyboard();
à appeler après allegro_init() ;

1) Approche SCANCODES

un tableau de valeurs booléennes
extern volatile char key[KEY_MAX];

Chaque indice correspond à une touche définie par une macro :
#define KEY_A ....

exemple d'utilisation :

```
while ( ! key[KEY_ESC]) { // tant que la touche échapp n'est pas pressée
   // instructions...     // faire les instructions
}
```

2) Approche BUFFER

int readkey(); // équivalent à getch dans conio.h

```
if ((readkey() & 0xff)  == 'd' )  // le codage ASCII
   printf("vous pressez 'd'\n");

if ((readkey() >> 8) == KEY_SPACE )// le codage scancode peut aussi
   printf("vous pressez espace\n");// être récupérer
```

void set_keyboard_rate(int delay, int repeat);
Sert à paramètre la répétition de touches clavier pour remplissage du buffer clavier :
le temps est donné en millisecondes, passer 0 temps annule toute répétition et pour répéter la touche il faudra d'abord en avoir relevé le doigt.
Cette fonction ne change rien au comportement de l'approche par SCANCODES

A noter aussi la fonction :
int keypressed(); // équivalent kbhit() ; renvoie vrai ou faux selon au moins une touche appuyée
// ou pas

PRIMITIVES DE DESSIN
Rappel des fonctions disponibles :

```
void vline(BITMAP *bmp, int x, int y1, int y2, int color);
void hline(BITMAP *bmp, int x1, int y, int x2, int color);
void line(BITMAP *bmp, int x1, int y1, int x2, int y2, int color);
void rect(BITMAP *bmp, int x1, int y1, int x2, int y2, int color);
void rectfill(BITMAP *bmp, int x1, int y1, int x2, int y2, int color);

void triangle(BITMAP *bmp, int x1, y1, x2, y2, x3, y3, int color);
void polygon(BITMAP *bmp, int vertices, const int *points, int color);

void circle(BITMAP *bmp, int x, int y, int radius, int color);
void circlefill(BITMAP *bmp, int x, int y, int radius, int color);
void ellipse(BITMAP *bmp, int x, int y, int rx, int ry, int color);
void ellipsefill(BITMAP *bmp, int x, int y, int rx, int ry, int color);
void arc(BITMAP *bmp, int x, y, fixed ang1, ang2, int r, int color);

void calc_spline(const int points[8], int npts, int *x, int *y);
void spline(BITMAP *bmp, const int points[8], int color);
void floodfill(BITMAP *bmp, int x, int y, int color);
```

Autres fonctions prenant des pointeurs de fonctions en argument :

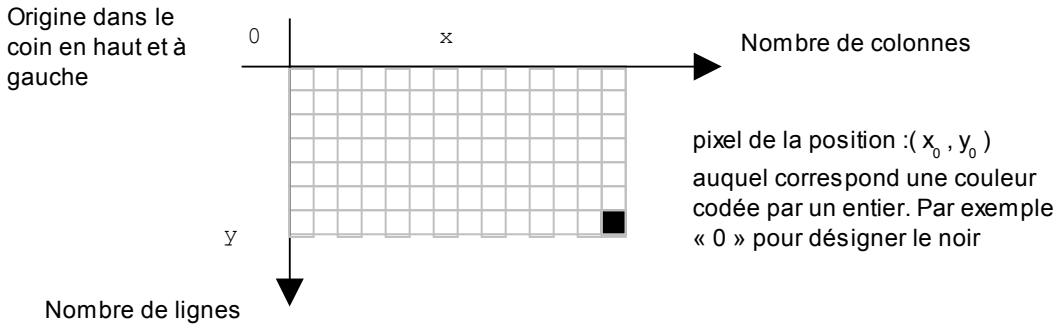
```
void do_line(BITMAP *bmp, int x1, y1, x2, y2, int d, void (*proc)(BITMAP *bmp, int x, int y, int d));
void do_circle(BITMAP *bmp, int x, int y, int radius, int d, void (*proc)(BITMAP *bmp, int x, int y, int d));
void do_ellipse(BITMAP *bmp, int x, int y, int rx, ry, int d, void (*proc)(BITMAP *bmp, int x, int y, int d));
void do_arc(BITMAP *bmp, int x, int y, fixed a1, fixed a2, int r, int d, void (*proc)(BITMAP *bmp, int x,
int y, int d));
```

1 Modes couleurs

1.1 Pixel : une position (x,y) et une couleur

Les pixels apparaissent à l'écran sous forme de petits carrés dont la taille dépend de la résolution de l'écran. Les résolutions habituelles sont 640 pixels par 480 pixels, 800 par 600, 1024 par 768 etc. Indépendamment de l'écran, pour chaque image numérique, les pixels quadrillent un plan rectangulaire de y lignes sur x colonnes. Ce plan rectangulaire est celui de l'image et chaque pixel y est une position fixe assortie d'une couleur.

La couleur du pixel est homogène mais elle peut être changée, elle est variable. L'image numérique est restituée comme une mosaïque, par les couleurs associées à chacun des pixels qui la constitue. Chaque couleur aura un numéro et le nombre total des couleurs possibles va dépendre du nombre de bits utilisés pour le codage de la couleur (8, 15, 16, 24, 32 bits).



Ainsi trois variables caractérisent le pixel dans un environnement de programmation, deux pour la position (x, y) fixe par rapport à l'image, une autre pour la couleur. Toutes trois prennent des valeurs entières.

1.2 Codage de la couleur

1.2.1 Modes "true colors" (15,16,24,32 bits) et palette (8 bits)

En RVB la couleur apparaît selon un mélange de Rouge, Vert et Bleu (en anglais RGB pour red, green, blue).
Le codage de ce mélange est relatif au nombre de bits donné comme « profondeur » de couleur (color depth).
15,16, 24, 32 bits correspondent aux modes dits « true colors »
8 bits correspond au mode « palette »

En modes « true color » on a :
- en 15 bits : 5 pour le rouge, 5 pour le vert, 5 pour le bleu
- en 16 bits : 5 pour le rouge, 6 pour le vert, 5 pour le bleu.

Dans ces deux cas La couleur est codée sur un unsigned short (2 octets) ensuite
- en 24 et 32 bits la couleur est codée sur un unsigned long (4 octets) et il y a un octet par composante de couleur.

En 32 bits l'octet restant est utilisé pour coder des informations particulières relatives à des effets de transparence ou de mixage d'images codées en différents formats par exemple.

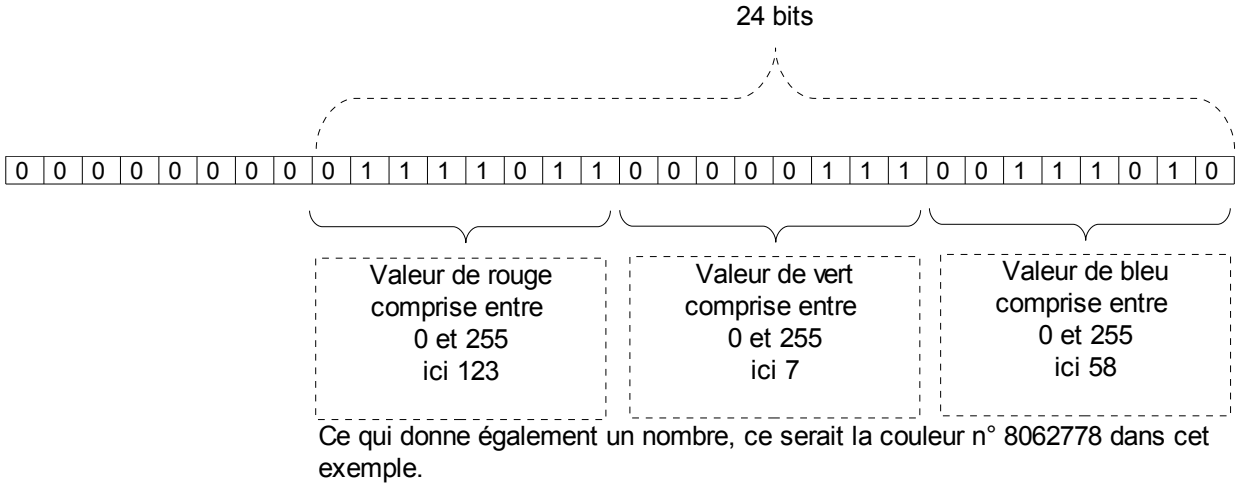
En mode « palette » :
Une couleur est codée à partir d'une structure qui est :

```
typedef struct RGB{ int r,g,b }RGB ;
```

Elle représente un mélange 18 bits où chaque composante de couleur est codée sur 6 bits.
Une « palette » de couleurs est un tableau de 256 structures RGB, chaque indice correspondant à une couleur. Dans allegro le type PALETTE est défini de la façon suivante :

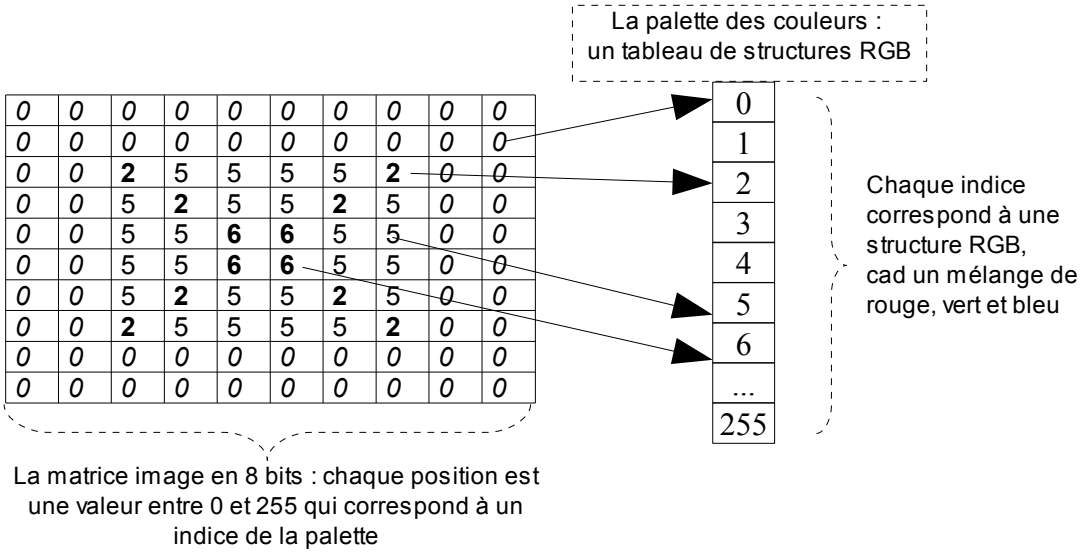
```
typedef RGB PALETTE[256] ;
```

Aperçu du mode "vraies couleurs" :



Ce qui donne également un nombre, ce serait la couleur n° 8062778 dans cet exemple.

Aperçu du mode "palette" 256 couleurs, 8 bits
Chaque indice du tableau va correspondre à une couleur, c'est-à-dire une structure RGB avec des valeurs de rouge, vert et bleu. Chaque pixel de l'image correspond à un indice de ce tableau, à savoir une valeur entr 0 et 255, ce qui donne pour l'image l'allure suivante :



Le mode 8 bits n'est pas à dédaigner. En effet la qualité de couleur est d'une précision 18 bits (r,g,b : 3 fois 6 bits) seul le nombre de couleurs simultanées est réduit à 256. Du point de vue esthétique il coïncide bien avec un univers graphique et nécessite la création d'ensembles de couleurs qui soient intéressants. Du point de vue technique, son économie de mémoire et de puissance fait qu'il réapparaît régulièrement dans des créneaux en développement, par exemple le web bas débit et en ce moment les pockets pc et téléphones mobiles.

1.2.2 Sélectionner un mode de couleur

Par défaut allegro est en 8 bits mode palette, mais il est possible de spécifier un autre mode avec la fonction set_color_depth().

void set_color_depth(int depth)
Cette fonction DOIT ÊTRE APPELEE AVANT set_gfx_mode sous peine de ralentir considérablement tout ce qui concerne l'affichage. Elle suffit de spécifier le format de couleur 15, 16, 24 et 32 bits souhaité. Le 16 bits est le plus avantageux sous le rapport rapidité/nombre de couleurs. Bien entendu le mode 8 bits par défaut n'a pas besoin d'être spécifié.

Test 2.1 : sélection du mode couleur

```
#include <allegro.h>

#define ERREUR(msg){\
   set_gfx_mode(GFX_TEXT,0,0,0,0);\
   allegro_message("err %s\nligne %d\nfile %s\n",msg,__LINE__,__FILE__);\
   allegro_exit();\
   return 1;\
}

int main()
{
int i,c;

   allegro_init();
   install_keyboard();
   srand(time(NULL));

   // sélection de la profondeur de couleur, si rien n'est spécifié
   // le mode couleur est en 8 bits
   set_color_depth(32);

   // mode graphique
   if(set_gfx_mode(GFX_AUTODETECT_WINDOWED,800,600,0,0)!=0)
      ERREUR(allegro_error);

   // boucle d'événements, fin si touche escape appuyée
   while(!key[KEY_ESC]){

      // si touche F1 dessiner une cible avec un dégradé de couleur
      if (key[KEY_F1]){
         c=rand()%256;
         for(i=256; i>0; i--)
            circlefill(screen,SCREEN_W/2,SCREEN_H/2,i,
                              makecol(i,c,256-i));
      }
   }
}
END_OF_MAIN();
```

Dans ce Test 1, chaque appuie sur la touche F1 affiche une cible (des cercles imbriqués) au centre de l'écran, en partant du plus grand vers le plus petit. Chaque cercle est d'une couleur légèrement différente de celle du cercle précédent de façon à produire un dégradé de couleurs à partir d'une tonalité fixe donnée aléatoirement par la valeur du vert. Pour voir les différence de mode couleur il suffit de recompiler le programme en spécifiant à chaque fois un nouveau mode.

1.3 Le mode "palette" (8 bits) par défaut sous allegro

1.3 .1 Créer une palette de couleurs

Palette de nuances de gris

Pour avoir du gris il suffit de mélanger la même quantité de rouge vert et bleu. Sachant que la palette à 256 couleurs et qu'il n'y a que 63 graduations d'intensité pour chaque couleur il faut répéter 4 fois le même mélange si l'on souhaite occuper toute la palette.

```
void nuance_de_gris(PALETTE pal)
{
int i ;
   for (i=0, v=0 ; i<256 ; i++){
   pal[i].r = pal[i].g = pal[i].b = i/4; // augmente de 1 toutes les
                                         // quatre itérations
   // la fonction set_palette rend la palette pal active
   set_palette(pal) ;
}
```

Remarque :
On peut obtenir le même résultat avec l'instruction suivante :
`pal[i].r = pal[i].g = pal[i].b = i >> 2;`

L'augmentation de « i » ira de 4 en 4 : la valeur 1 viendra quand i=4, la valeur 2 quand i = 8, 3 quand i = 16 etc.

void set_palette(const PALETTE p);
L'appel de cette fonction rend active la palette spécifiée au paramètre p. Ce sont donc ses couleurs qui sont visibles dans le programme.

Degradé de couleur constant

Autre exemple de palette, un dégradé qui sera toujours le même à chaque appel de la fonction. Le principe est pour chaque couleur de partir de zéro de monter à 63 et de redescendre à 0. Les couleurs ne partent pas ensembles, au départ c'est le rouge, lorsque le rouge est à 63 et amorce la redescente, le vert commence, lorsque le vert arrive à 63 et redescend le bleu démarre puis redescend après 63.

```
void degrade_constant ( PALETTE pal )
{
int c;
   for (c=0; c<64; c++) {
      pal[c].r = c;            // rouge de 0 à 63
      pal[c].g = 0;
      pal[c].b = 0;
   }
   for (c=64; c<128; c++) {
      pal[c].r = 127-c;        // rouge de 63 à 0
      pal[c].g = c-64;         // vert  de 0 à 63
      pal[c].b = 0;
   }
   for (c=128; c<192; c++) {
      pal[c].r = 0;
      pal[c].g = 191-c;        // vert de 63 à 0
      pal[c].b = c-128;        // bleu de  0 à 63
   }
   for (c=192; c<256; c++) {
      pal[c].r = 0;
      pal[c].g = 0;
      pal[c].b = 255-c;        // bleu de 63 à 0
```

```
   }
   set_palette(pal) ;          // pour activer la palette créée
}
```

Dégradé changeant

Autre cas, celui d'une palette qui sera différente à chaque appel de la fonction d'initialisation. La palette sera différente mais toujours construite sur le même modèle et elle conservera comme un air de famille quoique sous des couleurs différentes.

Test 2.2 : création d'une palette de couleurs

```
#include <allegro.h>
#include <time.h>

#define ERREUR(msg){\
   set_gfx_mode(GFX_TEXT,0,0,0,0);\
   allegro_message("err %s\nligne %d\nfile %s\n",msg,__LINE__,__FILE__);\
   allegro_exit();\
   return 1;\
}
/*****************************************
création d'une palette en dégradé
*****************************************/
void new_palette(PALETTE pal)
{
int i,rouge,pr,vert,pv,bleu,pb;

   // avoir du noir à l'indice 0
   pal[0].r=pal[0].g=pal[0].b=0;

   // valeurs de départ aléatoires pour le rouge, le vert, le bleu
   // les valeurs de couleurs sont comprises entre 0 et 63 compris
   rouge = rand()%54+5;
   vert = rand()%54+5;
   bleu = rand()%54+5;

   // une valeur aléatoire comprise entre -5 et 5 pour faire
   // progresser chaque valeur de couleur
   pr = rand()%11-5;
   pv = rand()%11-5;
   pb = rand()%11-5;

   // initiaiser chaque struct RGB de la palette à partir de
   ///l'indice 1 (0 utilisé pour le noir)
   for(i=1;i<256;i++){
      // pour chaque indice de couleur de la palette
      // 1 les valeurs de rouge, vert, bleu courantes sont
      // incrémentées du pas correspondant aléatoire
      rouge += pr;
      vert += pv;
      bleu +=pb;

      // 2 les nouvelles valeurs de rouge,vert,bleu sont affectée à
      // la couleur i de la palette
      pal[i].r = rouge;
      pal[i].g = vert;
      pal[i].b = bleu;

      // 3 Si une valeur de rouge, vert, bleu sort de la fourchette
```

```
      // permise (0 à 63) le pas correspondant est inversé
      if( rouge <=ABS(pr) || rouge >= 63-ABS(pr))
         pr*=-1;
      if( vert <=ABS(pv) || vert >= 63-ABS(pv))
         pv*=-1;
      if( bleu <=ABS(pb) || bleu >= 63-ABS(pb))
         pb*=-1;
   }

   // une fois la palette constituée elle est rendue opérationnelle
   // avec un appel à set_palette() :
   set_palette(pal);
}
/*****************************************
Visualisation de l'efficacité de la fonction
*****************************************/
int main()
{
int c;
PALETTE p;  // Déclaration d'une variable palette (1 tableau de 256
            // struct RGB)

   allegro_init();
   install_keyboard();
   srand(time(NULL));

   // Pas d'appel à set_color_depth() :
   // la profondeur de couleur par défaut est 8 bits

   if(set_gfx_mode(GFX_AUTODETECT_WINDOWED,800,600,0,0)!=0)
      ERREUR(allegro_error);

   // dessin d'une cible au centre constituée de cercles de rayon i
   // et de couleur i-1 (255 à 0)
   for(c=255; c>0; c--)
      circlefill(screen,SCREEN_W/2,SCREEN_H/2,c,c);

   // boucle d'événements, fin si touche escape appuyée
   while(!key[KEY_ESC]){

      // si touche F1 nouvelle palette de couleurs
      if (key[KEY_F1])
         new_palette(p);
   }
}
END_OF_MAIN();
```

1.3.2 Rotation de palette

On obtient une animation intéressante en faisant circuler les couleurs de la palette. Pour ce faire, sauver la couleur de la dernière position i (indice 255) et recopier la position i-1 dans i en faisant décroître i de 255 à 1. Pour finir recopier la dernière position précédemment sauvée à l'indice 1 (l'indice 0 reste en dehors, réservé pour la couleur du fond).

Test 2.3 : rotation d'une palette de couleurs

```
#include <allegro.h>
#include <time.h>
```

```
#define ERREUR(msg){\
   set_gfx_mode(GFX_TEXT,0,0,0,0);\
   allegro_message("err %s\nligne %d\nfile %s\n",msg,__LINE__,__FILE__);\
   allegro_exit();\
   return 1;\
}
/*****************************************
fonction de rotation d'une palette
*****************************************/
void rotation_palette(PALETTE pal)
{
RGB tmp;
int i;

   // 1 la dernière est sauvée dans une struct RGB temporaire
   tmp=pal[255];

   // 2 ensuite chaque position prend la valeur de la position
   // précédente jusqu'à l'indice 2 qui prend la valeur à l'indice 1
   // (on suppose l'indice 0, couleur noire, préservé)
   for (i=255; i>1; i--)
      pal[i]=pal[i-1];

   // 3 pour finir la couleur de l'indice 1 récupère à la valeur de
   // l'indice 255 du départ sauvé en tmp
   pal[1]=tmp;

   // 4 pour que la nouvelle palette soit visible, set_palette()
   set_palette(pal);
}
/*****************************************
visualisation
*****************************************/
int main()
{
int c,start;
PALETTE p;  // palette, tableau de 256 struct RGB

   allegro_init();
   install_keyboard();
   srand(time(NULL));

   // 8 bits par défaut
   if(set_gfx_mode(GFX_AUTODETECT_WINDOWED,800,600,0,0)!=0)
      ERREUR(allegro_error);

   // dessin d'une cible au centre constituée de cercles de rayon i
   // et de couleur i-1 (255 à 0)
   for(c=255; c>0; c--)
      circlefill(screen,SCREEN_W/2,SCREEN_H/2,c,c);

   // pour mesurer le temps, prise du moment de début
   start=clock();

   // boucle d'événements, fin si touche escape appuyée
   while(!key[KEY_ESC]){

      // si touche F1 nouvelle palette de couleurs
      if (key[KEY_F1])
         new_palette(p);
```

```
      // un appel de rotation toutes les 20 millisecondes
      if (clock()>start+20){
         rotation_palette(p);
         start=clock();
      }
   }
}
END_OF_MAIN();
```

Dans le Test 3 une palette nouvelle est créée avec la fonction new_palette() losque l'utilisateur appuie sur la touche F1. Cette fonction a été définie précédemment avec le test 2.

2. Bitmap en mémoire

2.1 Disposer d'une Bitmap dans le programme

2.1.1 Structure de données de la Bitmap

Bitmap est une contraction de « bit-map » qui signifie littéralement champ de bits à savoir une matrice d'octets. La librairie allegro définit le type BITMAP sous la forme d'une structure. Voici l'essentielle des données accessibles et manipulables par le programmeur :

```
     typedef struct BITMAP{
       int w, h;              // longueur et hauteur de l'image
(1)    int clip;              // clip actif 1 ou non 0
(2)    int cl, ct, cr, cb;    // zone clip comprise entre (cl,ct) et
                              // (cr,cb)
(3)    void*dat;                // ces deux champs pour les datas de
(4)    unsigned char *line[ ];// l'image (matrice)

     }BITMAP ;
```

(1) Le clip permet de définir à l'intérieur de l'image une zone rectangulaire accessible à des opérations de dessin. Aucun dessin n'est plus possible en dehors de cette zone si le clip est mis en oeuvre. Le clip sert surtout de sécurité afin de ne pas pouvoir écrire en dehors de l'image, à des adresses mémoire non réservées à l'image. Le clip est actif si la variable clip est différente de 0 et inactif si elle vaut 0. Par défaut le clip est actif sur la taille de l'image et garantit qu'il n'est pas possible d'écrire en dehors de la matrice des datas de l'image.

(2) La définition du clip. Les variables cl, ct, cr, cb donnent les coins haut-gauche et bas-droite de la zone du clip dans l'image. Par défaut le clip est initialisé avec les valeurs : left 0, top 0, right w et bottom h.

Au besoin, le clip ne se manipule pas directement mais via des fonctions fournies par la librairie :

void set_clip_rect(BITMAP*bitmap, int x1, int y1, int x2, int y2);
Cette fonction initialise une taille pour le clip, passer les coordonnées top-left et right-bottom ainsi que l'adresse de la bitmap concernée.
L'appel set_clip_rect(bitmap, 16, 16, 32, 32); permet d'écrire ensuite dans la bitmap de (16, 16) à (32, 32) compris mais pas en dehors au dessus de (15, 15) et après (33, 33).
L'appel set_clip_rect(bitmap, 0, 0, -1, -1); rend impossible l'écriture dans la bitmap concernée.
Si le clip passé dépasse la taille de l'image il est ramené à la taille de l'image. Il est également rectifié de façon cohérente si le coin haut gauche et bas droite sont inversés.

void get_clip_rect(BITMAP*bitmap, int *x1, int *y1, int *x2, int *y2);
Cette fonction permet de récupérer les valeurs du clip (passage par référence).

void add_clip_rect(BITMAP*bitmap, int x1, int y1, int x2, int y2);
Active le clip pour l'intersection entre le clip courant et celui spécifié dans les paramètres.

void set_clip_stateBITMAP*bitmap , int state)
Actionne le clip pour la bitmap en passant 1 et 0 pour désactiver le clip. Attention si le clip est désactivé de ne pas écrire en dehors de l'image sous peine de planter le programme.

int get_clip_state(BITMAP*bitmap)
Retourne 1 si le clip est actif et 0 sinon.

int is_inside_bitmap BITMAP*bitmap, int x, int y, int clip);
Si le paramètre clip est à 1, retourne 1 si le point de position (x,y) est à l'intérieur de la zone clip de l'image, 0 sinon. Si le paramètre clip est à 0 retourne si le point est dans l'image ou non.

(3)(4) Ces champs permettent d'accéder aux données de l'image, c'est à dire à l'espace mémoire qui les contient. En fait ce sont ici deux écritures pour le même espace mémoire.

Remarques
le « void * » est un pointeur générique c'est à dire un pointeur qui permet de transmettre l'adresse d'un objet dont on ignore le type. A l'origine le void* était le type unsigned char* parce que le langage C est faiblement typé.

Dans la structure BITMAP, void *dat est destiné à contenir l'adresse du bloc mémoire réservé pour l'image et void* est utilisé pour pouvoir produire indifféremment des images codée en 8 bits (unsigned char), 15-16 bits (unsigned short) ou 24-32 bits (unsigned long).

L'objectif est de pouvoir coder des images de n'importe quelle taille dans n'importe quel mode couleur (8,16,24,32 bits). Avec pour unité le « unsigned char* » le tableau « line » est un tableau de pointeurs sur octets. C'est à dire un tableau de tableaux d'octets. Chaque élément de « line » est l'adresse du premier élément d'un tableau d'octets dont on ignore le nombre. Avec cette formule on a une matrice dont on ne connaît ni le nombre de lignes, ni le nombre de colonnes.

De plus unsigned char *line[] est une expression incomplète en principe réservée aux paramètres de fonction et qui vaut 0 octet dans la structure. Elle ne peut donc y figurer qu'à la fin à cause de l'alignement des autres champs et elle devra faire l'objet d'une allocation particulière (voir détail de la fonction create_bitmap_ex plus bas).

2.1.2 Créer, détruire une Bitmap en mémoire

BITMAP *create_bitmap(int width, int height);
Cette fonction alloue l'espace mémoire requis pour une BITMAP de longueur width et de hauteur height puis retourne son adresse via un pointeur ou NULL si problème. Le clip est activé par défaut sur la taille de l'image ce qui rend impossible des débordements lors des opérations de dessin. Cette fonction crée toujours une bitmap dans le mode couleur courant qui est 8 bits par défaut.
Pour créer une bitmap dans un autre mode couleur en particulier il faut utiliser la fonction :

BITMAP *create_bitmap_ex(int color_depth, int width, int height);
Crée une bitmap dans un format de pixel spécifié par « color_depth » (8, 15, 16, 24 or 32 bits par pixel).

Compte tenu de l'initialisation spécifique que requiert une structure BITMAP, la fonction standard free() ne peut pas être utilisée pour libérer la mémoire allouée par create_bitmap(), il faut utiliser la fonction :

void destroy_bitmap(BITMAP*bmp)
Libère la mémoire précédement allouée pour une bitmap (tous les types utilisés par allegro : memory bitmap, sub-bitmap, video memory bitmap, or system bitmap , voir doc « bitmap objects »).

Exemple de création d'une BITMAP dans un programme

Le programme suivant initialise allegro, clavier et mode graphique puis crée une bitmap de la taille de l'écran avec un contrôle d'erreur, ensuite il attend que la touche échapp soit pressée. A la sortie de la boucle while la bitmap est désallouée, le programme est fini.

```
int main()
{
BITMAP *bmp ;

   allegro_init() ;
   install_keyboard() ;

   if (set_gfx_mode(GFX_AUTODETECT,640,480,0,0) !=0)
      ERREUR("mode graphique") ;

   // création d'une bitmap de 640 par 480
   bmp=create_bitmap(640,480);
   if ( !bmp)  // test si erreur lors de l'exécution
      ERREUR("création bitmap");

   while ( ! key[KEY_ESC] ){   }

   destroy_bitmap(bmp);
   return 0;

}
```

Précisions sur la création de bitmap

A titre indicatif et par curiosité scientifique voici la fonction d'initialisation des bitmaps extraite du code source d'Allegro suivie de quelques explications. Il s'agit de la fonction creat_bitmap_ex(). En fait la fonction create_bitmap est un appel à create_bitmap_ex avec le mode couleur courant spécifié comme profondeur de couleur « color depth ». Les lignes numérotées et en caractères gras correspondent aux initialisations des champs de la structure BITMAP qui ont été présentés :

```
BITMAP *create_bitmap_ex(int color_depth, int width, int height)
{
GFX_VTABLE *vtable;
BITMAP *bitmap;
int i;

    if (system_driver->create_bitmap)
      return system_driver->create_bitmap(color_depth,width,height);

    vtable = _get_vtable(color_depth);
    if (!vtable)
     return NULL;

(1) bitmap = malloc(sizeof(BITMAP) + (sizeof(char *) * height));
    if (!bitmap)
     return NULL;

(2) bitmap->dat = malloc(width * height * BYTES_PER_PIXEL(color_depth));
    if (!bitmap->dat) {
      free(bitmap);
      return NULL;
    }

(3) bitmap->w = bitmap->cr = width;
    bitmap->h = bitmap->cb = height;
```

```
    bitmap->clip = TRUE;
    bitmap->cl = bitmap->ct = 0;
    bitmap->vtable = vtable;
    bitmap->write_bank = bitmap->read_bank = _stub_bank_switch;
    bitmap->id = 0;
    bitmap->extra = NULL;
    bitmap->x_ofs = 0;
    bitmap->y_ofs = 0;
    bitmap->seg = _default_ds();

(4) bitmap->line[0] = bitmap->dat;
    for (i=1; i<height; i++)
      bitmap->line[i]=bitmap->line[i-1]+width*BYTES_PER_PIXEL(color_depth);

    if (system_driver->created_bitmap)
      system_driver->created_bitmap(bitmap);

(5) return bitmap;
   }
```

(1) Adresse et allocation de la taille d'une structure BITMAP et de ses champs c'est à dire « sizeof (BITMAP) ».

Mais il faut lui ajouter la taille voulue du tableau unsigned char*line[] c'est-à-dire le nombre de lignes demandées avec le paramètre height. Chaque ligne est un unsigned char*, d'où l'expression :
(sizeof((char*) * height) pour la taille à ajouter à la structure (char* et unsigned char* ont même taille).

(2) Adresse et allocation des datas de l'image en fonction de sa taille et du codage de la couleur 8,15,16,24,32 bits. C'est l'expression : weight * height * le nombre d'octets par pixel (1, 2,3,4)

Selon le paramètre color_depth (8,15,16,24,32) passé en argument à la fonction, la macro BYTES_PER_PIXEL donne le nombre d'octets de la façon suivante :

```
#define BYTES_PER_PIXEL(bpp)\
             (((bpp) <= 8) ?  1\
             : (((bpp) <= 16) ?  sizeof (unsigned short)\
             : (((bpp) <= 24) ?  3 : sizeof (unsigned long))))
```

Il s'agit de trois expressions conditionnelles imbriquées qui renvoient 1, 2, 3 ou 4 selon que le paramètre bpp vaut 8, 15 ou 16, 24, 32.

Remarque : expression conditionnelle
C'est une expression qui se décompose en un test plus deux possibilités en fonction des résultats du test : expr1 ? expr2 : expr3
On peut traduire par : « Est-ce que expr1 est vrai ? Si oui la valeur de l'expression est expr2 si non la valeur de l'expression est expr3 ».

Par exemple l'expression « (x >= 0) ? 1 : -1 » renvoie 1 ou -1 selon le signe positif ou négatif d'une variable x. De même (x >= 0) ? x : -x retourne la valeur absolue d'une variable x.

Il est possible d'imbriquer des expressions conditionnelles. C'est le cas avec BYTES_PER_PIXEL. Le nombre de bits qui code la couleur (color_depth) est passé à la macro BYTES_PER_PIXEL via le paramètre bpp. Le premier test détermine si cette valeur est inférieure ou égale à 8. Si oui le nombre d'octets par pixel est 1 et l'expression donne 1. Si non, nouveau test pour savoir si cette valeur est inférieure ou égale à 16. Si oui le nombre d'octets par pixel est 2 et l'expression vaut 2. Si non, nouveau test pour savoir si cette valeur est inférieure ou égale à 24. Si oui l'expression vaut 3 et sinon elle vaut la taille d'un long c'est à dire 4.

Ainsi la formule « width * height * BYTES_PER_PIXEL(color_depth) » donne la taille totale des datas de l'image en mémoire compte-tenu de sa longueur (width), de sa hauteur (height) et du nombre

d'octets utilisés pour coder l'image (BYTES_PER_PIXEL(color_depth)).

(3) Initialisation du clip sur la taille de l'image, par défaut le clip est actif (TRUE)

(4) Le bloc « dat » (weigth*heigth*nb_octet_couleur) des données de l'image est découpé en un nombre heigth de lignes :
1) l'adresse « dat » (bitmap->dat) est affectée à la première ligne du tableau line (bitmap->line[0]).
2) la deuxième ligne prend l'adresse de la première plus la taille de la ligne
3) et ainsi de suite jusqu'à la dernière ligne (heigth-1)

(5) Une fois l'initialisation terminée la fonction retourne le pointeur « bitmap » qui va être récupéré dans la fonction appelante avec un appel comme celui de l'exemple donné plus haut :
```
bmp=create_bitmap(640,480);
```

les champs de la structures BITMAP sont accessibles via le pointeur ce qui donne des expressions comme :
```
   bmp->w   // pour la longueur
   bmp->h   // pour la hauteur
```
etc.

On remarquera en particulier la possibilité d'accéder à chaque pixel de l'image soit par le pointeur void* dat qui se traite alors comme un tableau à une dimension de taille (bmp->w * bmp->h) soit par le tableau unsigned char* line[], avec pour une position (x,y) dans l'image l'expression : bmp->line[y][x] (voir la partie "accéder aux datas d'une bitmap").

2.2 Utiliser une Bitmap

2.2.1 Dessiner, placer du texte, effacer une Bitmap

Toutes les primitives de dessin et d'affichage de texte sont utilisables avec les bitmaps pour dessiner et afficher du texte via des pointeurs BITMAP* sur les bitmap de destination. Toutes ces fonctions
- utilisent le la profondeur de couleur courante (color depth, 8,15,16,24,32 bits)
- respectent le rectangle de clip de la bitmap de destination (elles n'écrivent pas en dehors)

Pour effacer une bitmap ou la colorer d'une couleur spécifique deux fonctions sont disponibles :

void clear_bitmap(BITMAP*bmp);
Met toutes les données de l'image à 0
void clear_to_color(BITMAP*bmp, int color) ;
Met toutes les données à la couleur spécifiée par color.

Exemple de dessin dans une bitmap mémoire

```
int main()
{
BITMAP *bmp ;
PALETTE pal;
int x,y,r;

   allegro_init() ;
   install_keyboard() ;
   if (set_gfx_mode(GFX_AUTODETECT,640,480,0,0) !=0)
      ERREUR("mode graphique") ;

   bmp=create_bitmap(61,100);
   if ( !bmp)
      ERREUR("creation bitmap");

   // une palette
   degrade_variable(pal);
```

```
   // DESSIN dans la bitmap
   clear_bitmap(bmp); // effacer

   // oeil gauche
   x=60/4;
   y=100/4;
   r=5;
   circle(bmp,x-r*2,y,r,100);
   circle(bmp,x-r*2,y,r-3,200);

   // oeil droit
   x=(60/4)*3;
   y=100/4;
   r=5;
   circle(bmp,x+r*2,y,r,100);
   circle(bmp,x+r*2,y,r-3,200);

   // nez
   triangle(bmp, 0, 99, 59, 99, 30, 0, 50);

   // dents
   for (x=0, y=89,r=0; x<60; x+=4,r^=1){
      if (r)
         rectfill(bmp, x,y,x+4,y+10,0);
      else
         rectfill(bmp, x,y,x+4,y+10,255);
   }

   // boucle vide et attention : il n'y a pas d'affichage dans ce
   // programme….
   while ( ! key[KEY_ESC] ){   }

   destroy_bitmap(bmp);
   exit(EXIT_SUCCESS);
}
END_OF_MAIN() ;
```

Dans l'exemple ci-dessus il n'y a pas encore d'affichage, la bitmap est créée, une palette est initialisée, le dessin est effectué mais il n'y a aucun affichage à l'écran. Pour ce faire nous allons utiliser la fonction blit() et la variable globale screen initialisée par la fonction set_gfx_mode() qui correspond à la bitmap de mémoire vidéo de l'écran.

2.2.2 Afficher (copier) une Bitmap : foncion blit()

Des fonctions d'affichage puissantes permettent de recopier des images ou des parties d'images bitmap dans une autre image bitmap toujours désignées avec des pointeurs BITMAP*, e particulier la fonction blit() :

```
void blit(  BITMAP *source,                 // l'image source
            BITMAP *dest,                   // l'image destination
            int source_x, int source_y, //coordonnée du point haut-gauche dans l'image source
            int dest_x,    int dest_y,     // coordonnée du point haut-gauche dans l'image destination
            int width,     int height );   // taille pour la copie à partir du point haut-gauche dans
                                           // l'image source
```

(Dans la documentation Allegro à « Blitting and sprites »)
blit vient de « bit-block_transfer » une méthode qui permet de copier des blocs d'infos d'un endroit de mémoire vers un autre et de façon presque instantanée. La fonction blit copie une zone rectangulaire d'une bitmap source vers bitmap destination. source_x et source_y correspondent au coin haut gauche choisi dans l'image source, width et height la taille à partir de ce point toujours dans l'image source : la zone rectangulaire dans l'image source est ainsi délimitée par le coin source_x source_y (gauche-haut) et le coin source_x+width et source_y+height (droite-bas).

La copie est effectuée dans l'image de destination à partir de la position spécifiée en dest_x, dest_y. Cette routine respecte les clips de l'image source et destination et garantit que l'on reste toujours dans les données des images (pas de débordement).

Remarque :
acquire_screen() et **release_screen()**
(voir doc "Bitmap objects")

Si la bitmap de destination est une bitmap de mémoire vidéo comme screen et non une bitmap créée en ram avec create_bitmap(), sous Windows il est alors nécessaire de verrouiller la bitmap de mémoire vidéo avant d'afficher dedans. Toutes les fonctions d'affichage d'Allegro le font automatiquement. Cependant c'est une opération assez lente, imperceptible pour un petit nombre d'affichages mais éventuellement un ralentissement se produit lors d'appels répétés très nombreux par exemple :

```
for (i= 0 < 640*480 ; i++)
   putpixel(screen, x,y,color);
```

Dans ce cas il est possible de verrouiller la mémoire vidéo une seule fois au début des opérations d'affichage avec un seul déverrouillage à la fin. Deux fonctions de verrouillage – déverrouillage sont prévues pour cette situation : acquire_screen() et release_screen()

Ce sont en fait deux macros pour les fonctions :
void acquire_bitmap(BITMAP *bmp);
et
void release_bitmap(BITMAP *bmp);
lorsqu'elles sont utilisées avec la bitmap écran screen :
acquire_screen(); est équivalent à acquire_bitmap(screen); et
release_screen(); est équivalent à release_bitmap(screen);

Lorsqu'elles sont utilisées, chaque appel de acquire_screen() doit être suivi, après les appels des fonctions d'affichages, d'un appel à release_screen(). Entre les deux il ne faut pas utiliser timer, keyboard ou autres routines non-graphiques qui permettent des entrées sous peine de plantage. Ce qui donne pour l'exemple précédent :

```
acquire_screen() ;
for (i= 0 < 640*480 ; i++)
   putpixel(screen, x,y,color);
release_screen();
```

Cette modification permet alors d'accélérer sensiblement l'affichage sous windows.

Affichage à l'écran d'une bitmap mémoire

le programme précédent qui dessine un bonhomme peut maintenant être complété par un affichage à l'écran. Dans les lignes qui suivent nous avons ajouté une animation basique de la palette qui change toute les 280 millisecondes :

```
// position du sprite dessiné au centre de l'écran
x=(ECRAN_X-bmp->w)/2;
y=(ECRAN_Y-bmp->h)/2;

// affichage à l'écran
blit(bmp,screen,0,0,x,y,SCREEN_W,SCREEN_H);

// BOUCLE EVENT
while (!key[KEY_ESC]){

   // un temps d'attente  en millisecondes
   rest(280);
```

```
      // une nouvelle palette tous les 280
      degrade_pas_variable(pal);
   }
```

Le test n°4 récapitule ce que nous venons de voir concernant l'allocation dynamique de bitmap, le dessin dedans et l'affichage écran. La fonction dessine( ) bombarde une bitmap de rectangles de couleurs. Dans le main, après les initialisations requises, dans la boucle d'évènements, appuyer sur la touche F1 crée une nouvelle palette de couleurs. Appuyer sur la touche F2 efface la bitmap soit en couleur soit en noir, crée un nouveau dessin dedans et l'affiche,à l'écran.

Test 2.4 : allocation bitmap, dessin et affichage écran

```
#include <allegro.h>
#include <time.h>

#define ERREUR(msg){\
   set_gfx_mode(GFX_TEXT,0,0,0,0);\
   allegro_message("err %s\nligne %d\nfile %s\n",msg,__LINE__,__FILE__);\
   allegro_exit();\
   return 1;\
}
/*****************************************************************
La fonction dessine :
un bombardement de rectangles dans la bitmap passée en argument
*****************************************************************/
void dessine (BITMAP*bmp)
{
int i,x,y,tx,ty,color;
   // 1 faire le dessin
   for (i=0; i<20; i++){
      // taille max de 100 pixels
      tx=5+rand()%96;
      ty=5+rand()%96;
      // position à l'intérieur de la bitmap
      x=rand()%(bmp->w-tx);
      y=rand()%(bmp->h-ty);
      // une couleur de la palette sauf noir
      color=1+rand()%255;
      // dessin d'un rect plein
      rectfill(bmp, x,y,x+tx,y+ty,color);
   }
}
/*****************************************************************
*****************************************************************/
int main()
{
PALETTE p;  // Déclaration 1 tableau de 256 struct RGB palete
BITMAP*bmp; // Déclaration d'un pointeur sur une structure BITMAP
int x,y;    // pour position de la Bitmap à l'écran
   allegro_init();
   install_keyboard();
   srand(time(NULL));

   // Pas d'appel à set_color_depth() :
   // la profondeur de couleur par défaut est 8 bits

   if(set_gfx_mode(GFX_AUTODETECT_WINDOWED,800,600,0,0)!=0)
      ERREUR(allegro_error);
```

```
   // 1 allocation dynamique d'une structure BITMAP
   bmp=create_bitmap(SCREEN_W/2,SCREEN_H/2);
   if (!bmp)
      ERREUR("create_bitmap");

   // 2 effacer contenu éventuel qui traine en mémoire (toutes les
   // positions sont forcées à 0)
   clear_bitmap(bmp);

   // 3 dessiner dans la Bitmap
   dessine(bmp);

   // 4 affichage au centre de l'écran de la Bitmap
   x=(SCREEN_W-bmp->w)/2;
   y=(SCREEN_H-bmp->h)/2;
   blit(bmp, screen, 0,0, x, y, bmp->w, bmp->h);

   // boucle d'événements, fin si touche escape appuyée
   while(!key[KEY_ESC]){

       // si touche F2 effacer,dessiner dans la bitmap et afficher
       // la bitmap
       if (key[KEY_F2]){
          // 1 effacer avec couleur de fond si valeur 1
          if (rand()%2)
             clear_to_color(bmp,rand()%256);
          // sinon effacer en noir
          else
             clear_bitmap(bmp);
          // 2 dessin dans la bitmap
          dessine(bmp);
          // 3 affichage écran
          blit(bmp, screen, 0,0, x, y, bmp->w, bmp->h);
       }
       // si touche F1 nouvelle palette de couleurs
      if (key[KEY_F1])
        new_palette(p);
   }

   // libérer la mémoire allouée à la sortie
   destroy_bitmap(bmp);
   return 0;
}
END_OF_MAIN();
```

2.2.3 Bouger une bitmap

Pour bouger une bitmap et avoir une animation il faut dans une boucle, effacer l'écran, modifier les coordonnées de la bitmap, contrôler si elle sort par les bords, la réafficher à l'écran aux nouvelles cordonnées, ce qui donne la petite séquence algorithmique :

1 effacer la bitmap à l'écran
2 modifier coordonnées de la BITMAP (contrôle des bords éventuels)
3 afficher la bitmap à l'écran aux nouvelles coordonnées
retour en 1

Le programme n°5 reprend le programme n°4 et il ajoute un mouvement horizontal permanent à la bitmap qui va de gauche à droite et de droite à gauche. Elle change de sens à chaque fois qu'elle touche un bord.

```
int main()
{
PALETTE p;  // tableau de 256 struct RGB
BITMAP*bmp; // Déclaration d'un pointeur sur une structure BITMAP
int x,y;    // pour position de la Bitmap à l'écran
int px;     // pour déplacement de la bitmap horizontalement

   allegro_init();
   install_keyboard();
   srand(time(NULL));
   if(set_gfx_mode(GFX_AUTODETECT_WINDOWED,800,600,0,0)!=0)
      ERREU(allegro_error);

   // 1 allocation dynamique d'une structure BITMAP
   bmp=create_bitmap(SCREEN_W/2,SCREEN_H/2);
   if (!bmp)
      ERREUR("create_bitmap");

   // 2 effacer contenu éventuel qui traine en mémoire (toutes les
   // positions sont forcées à 0)
   clear_bitmap(bmp);
   // 3 dessiner dans la Bitmap
   dessine(bmp);

   // 4 affichage au centre de l'écran de la Bitmap
   x=(SCREEN_W-bmp->w)/2;
   y=(SCREEN_H-bmp->h)/2;
   blit(bmp, screen, 0,0, x, y, bmp->w, bmp->h);
  // pour le déplacement, une valeur de 1 ou -1
  px=(rand()%2)*2-1;
   // boucle d'événements, fin si touche escape appuyée
   while(!key[KEY_ESC]){
      // bouger la bitmap :
      // 1 modifier ses coordonnées (ici mouvement uniquement
      // horizontal)
      x+=px;
      if (x<0 || x+bmp->w>SCREEN_W )
         px*=-1;
      // 2 effacer l'écran
      // si l'écran n'est pas effacé avant affichage, dans certain
      // cas on verra des bavures du dessin. Une solution consiste
      // à appeler :
      // clear_bitmap(screen);
      // mais c'est assez moche comme résultat, nous verrons
      // comment améliorer l'affichage avec un doudle buffer dans
      // le chapitre suivant.

      // 3 afficher
      blit(bmp, screen, 0,0, x, y, bmp->w, bmp->h);
      // juste ralentir un peu
      rest(5);

      // si touche F2 effacer,dessiner dans la bitmap et afficher
      // la bitmap
```

```
      if (key[KEY_F2]){
          // effacer
          clear_bitmap(bmp);
          // dessin dans la bitmap
          dessine(bmp);
       }
       // si touche F1 nouvelle palette de couleurs
      if (key[KEY_F1])
         new_palette(p);
   }
   // libérer la mémoire allouée à la sortie
   destroy_bitmap(bmp);
   return 0;
}
END_OF_MAIN();
```

2.3 Sauver, récupérer une Bitmap

2.3.1 Sauvegarde d'une Bitmap

Soit une bitmap créée dans un programme avec la fonction create_bitmap(). Pour sauver cette image il suffit d'appeler l'une des quatre fonctions suivantes :

int save_bitmap(const char *filename, BITMAP *bmp, const RGB *pal);
Ecrit la bitmap « bmp » dans un fichier. Il y a trois formats d'images possibles BMP, PCX, TGA et le nom du fichier « filename » doit contenir l'extension pour le format souhaité (« .bmp », « .pcx » ou « .tga »). Si la bitmap est en 8 bits il faut spécifier sa palette de couleur « pal » et passer NULL sinon.

int save_bmp(const char *filename, BITMAP *bmp, const RGB *pal);
Fonction alternative qui écrit une bitmap au format BMP en 256 couleur ou en 24 bits truecolor.

int save_pcx(const char *filename, BITMAP *bmp, const RGB *pal);
Ecrit une bitmap au format PCX en 256 couleur ou en 24 bits truecolor.

int save_tga (const char *filename, BITMAP *bmp, const RGB *pal);
Ecrit une bitmap au format TGA en 256 couleur, 15 bit, 24 ou 32 bits truecolor.

Le programme de test n°6 ci-dessous reprend le programme précédent auquel il ajoute une fonction de sauvegarde. Elle est obtenue en pressant sur la touche S.

Test 2.6 : sauvegarder une bitmap

```
int main()
{
PALETTE p;      // un tableau de 256 struct RGB)
BITMAP*bmp;     // Déclaration d'un pointeur sur une structure BITMAP
int x,y;        // pour position de la Bitmap à l'écran
int px;         // pour déplacement de la bitmap horizontalement
char nom[100]; // pour le nom du fichier à sauver
int cmpt=0;     // pour numéroter les sauvegardes

   allegro_init();
   install_keyboard();
   srand(time(NULL));
   if(set_gfx_mode(GFX_AUTODETECT_WINDOWED,800,600,0,0)!=0)
      ERREUR(allegro_error);
```

```
   // Bitmap initiale,affichage en x,y
   bmp=create_bitmap(SCREEN_W/2,SCREEN_H/2);
   if (!bmp)
      ERREUR("create_bitmap");

   clear_bitmap(bmp);
   dessine(bmp);
   x=(SCREEN_W-bmp->w)/2;
   y=(SCREEN_H-bmp->h)/2;
   blit(bmp, screen, 0,0, x, y, bmp->w, bmp->h);
   px=(rand()%2)*2-1;
   // une palette initiale
   new_palette(p);

   // interdire les répétition de touche
   set_keyboard_rate(0,0);

   // boucle event
   while(!key[KEY_ESC]){

       // bouger la bitmap
       x+=px;
       if (x<0 || x+bmp->w>SCREEN_W )
          px*=-1;
       blit(bmp, screen, 0,0, x, y, bmp->w, bmp->h);
       rest(5);

       if (keypressed()){
          switch(readkey()>>8){

             // si S faire une sauvegarde
             case KEY_S :
                sprintf(nom,"save%d.bmp",cmpt++);
                save_bitmap(nom, bmp, p);
                break;

             // dessiner
             case KEY_F2 :
                clear_bitmap(bmp);
                dessine(bmp);
                break;

             // nouvelle palette
             case KEY_F1 :
                new_palette(p);
                break;
          }
       }
   }
   // libérer la mémoire allouée à la sortie
   destroy_bitmap(bmp);
   return 0;
}
END_OF_MAIN();
```

Remarque :
Pour faire un nom de fichier automatique qui change à chaque nouvelle sauvegarde nous utilisons ici la fonction standard sprintf(). Le premier paramètre est un tableau de char destiné à recevoir la chaîne de caractères fabriquée avec les autres paramètres. Le deuxième paramètre est une chaine de

caractères formatée comme dans la fonction printf(). Elle est suivi d'une liste des variables nécessaires en fonction des format demandés dans la chaîne. L'appel de sprintf(nom,"save %d.bmp",cmpt++); permet de fabriquer des mons "save0.bmp", "save1.bmp" etc. en fonction de la valeur de la variable cmpt qui est incrémentée de un à chaque nouvelle sauvegarde.

2.3.2 Faire une photo d'écran

Pour faire une photo d'écran ou d'une partie d'écran le plus simple est de créer une bitmap mémoire à la taille voulue (tout l'écran ou juste une partie) puis copier dedans la partie de l'écran souhaitée avec la fonction blit(), ensuite appeler save_bitmap().

Le programme n°7 ajoute au précédent la possibilité de photos d'écran obtenues en pressant la touche P.

Test 2.7 : Faire une photo d'écran

```
int main()
{
PALETTE p;     //  tableau de 256 struct RGB
BITMAP*bmp;    // Déclaration d'un pointeur sur une structure BITMAP
BITMAP*ecr  ;  // idem pour photo écran
int x,y;       // pour position de la Bitmap à l'écran
int px;        // pour déplacement de la bitmap horizontalement
char nom[100]; // pour le nom du fichier à sauver
int cmpt=0;    // pour numéroter les sauvegardes

   allegro_init();
   install_keyboard();
   srand(time(NULL));

   if(set_gfx_mode(GFX_AUTODETECT_WINDOWED,800,600,0,0)!=0)
      ERREUR(allegro_error);

   // bitmap pour photo écran de la taille de l'écran
   ecr=create_bitmap(SCREEN_W,SCREEN_H);

   // Bitmap initiale pour dessiner,affichage en x,y
   bmp=create_bitmap(SCREEN_W/2,SCREEN_H/2);
   if (!bmp || !ecr)
      ERREUR("create_bitmap");

   clear_bitmap(bmp);
   dessine(bmp);
   x=(SCREEN_W-bmp->w)/2;
   y=(SCREEN_H-bmp->h)/2;
   blit(bmp, screen, 0,0, x, y, bmp->w, bmp->h);
   px=(rand()%2)*2-1;

   // une palette initiale
   new_palette(p);

   // interdire les répétition de touche
   set_keyboard_rate(0,0);

   // boucle event
   while(!key[KEY_ESC]){

       // bouger la bitmap
       x+=px;
```

```
        if (x<0 || x+bmp->w>SCREEN_W )
           px*=-1;
        blit(bmp, screen, 0,0, x, y, bmp->w, bmp->h);
        rest(5);

        if (keypressed()){
           switch(readkey()>>8){
              // faire une photo écran
              case KEY_P :
                 // copier l'écran dans la bitmap "ecr"
                 blit(screen,ecr,0,0,0,0,SCREEN_W,SCREEN_H);
                 sprintf(nom,"save_ecran%d.bmp",cmpt++);
                 save_bitmap(nom, ecr, p);
                 break;

              // faire une sauvegarde de la bitmap bmp
              case KEY_S :
                 sprintf(nom,"save%d.bmp",cmpt++);
                 save_bitmap(nom, bmp, p);
                 break;

              // dessiner
              case KEY_F2 :
                 clear_bitmap(bmp);
                 dessine(bmp);
                 break;

              // nouvelle palette
              case KEY_F1 :
                 new_palette(p);
                 break;
           }
        }
     }
     // libérer la mémoire allouée à la sortie
     destroy_bitmap(bmp);
     return 0;
}
END_OF_MAIN();
```

2.3.3 Récupérer un fichier Bitmap dans le programme

BITMAP* load_bitmap(const char *filename, RGB *pal);
Cette fonction charge un fichier bitmap (BMP, LBM, PCX, et TGA) en mémoire à partir du nom et chemin spécifié par la chaîne « filename » et retourne un pointeur sur une structure BITMAP. Si l'image est en 8 bits ses couleurs sont stockées dans une pallette « pal », pour les autres formats de couleur l'argument « pal » peut être NULL. Si le format de l'image ne correspond pas au format courrant du programme l'image est convertie dans le format courant. Retourne NULL si erreur.

Le test n°8 suivant est configuré en 8 bits. Il charge une image qui doit se trouver dans le même répertoire que le programme et s'appeler "image.bmp". Si ce n'est pas le cas, lors de l'appel de la fonction load_bitmap() mettez pour le premier paramètre le nom de l'image que vous souhaitez loader tout en indiquant le répertoire où elle se trouve.

Test 2.8 : récupérer une bitmap dans le programme (load)

```
#include <allegro.h>
#include <time.h>
```

```
#define ERREUR(msg){\
   set_gfx_mode(GFX_TEXT,0,0,0,0);\
   allegro_message("err %s\nligne %d\nfile %s\n",msg,__LINE__,__FILE__);\
   allegro_exit();\
   return 1;\
}

int main()
{
PALETTE p,po; // 2 palettes pour, en 8 bits, test sur les palettes
BITMAP*bmp;
int x,y,px;

   allegro_init();
   install_keyboard();
   srand(time(NULL));

   // Couleur :
   // si autre que 8 bits il n'y a pas de palette à gérer
   //set_color_depth(16);
   if(set_gfx_mode(GFX_AUTODETECT_WINDOWED,800,600,0,0)!=0)
      ERREUR(allegro_error);

   // récupérer une image dans le programme. Si 8 bits passer une palette
   // au deuxième paramètre pour récupérer la palette de l'image.
   // Si 15,1,24,32 bits,  le paramètre PALETTE prend la valeur NULL
   bmp=load_bitmap("image.bmp",po);     // si 8 bits
   //bmp=load_bitmap("image.bmp",NULL); // si autre
   if (!bmp)
      ERREUR("load_bitmap");

   // si mode palette 8 bits activer la palette des couleurs de l'image
   set_palette(po);

   x=(SCREEN_W-bmp->w)/2;
   y=(SCREEN_H-bmp->h)/2;
   blit(bmp, screen, 0,0, x, y, bmp->w, bmp->h);

   // boucle event
   while(!key[KEY_ESC]){

       // si 8 bits, jeu sur les palettes :

       // nouvelle palette
       if (key[KEY_F1])
          new_palette(p);
       // restore palette originale
       if (key[KEY_F2])
          set_palette(po);
       // modification de la couleur à l'indice 0
       if (key[KEY_F3]){
          po[0].r=po[0].g=po[0].b=0;
          set_palette(po);
       }
   }
   // libérer la mémoire allouée à la sortie
   destroy_bitmap(bmp);
   return 0;
}
END_OF_MAIN();
```

2.4 Accéder aux datas d'une Bitmap

Il est possible d'accéder à la valeur de chaque pixel d'une image soit pour prendre sa valeur soit pour la modifier. Le premier moyen consiste à utiliser les fonctions suivantes :

void putpixel(BITMAP *bmp, int x, int y, int color);
Ecrit la couleur color d'un pixel à la position x,y de la bitmap bmp.

int getpixel(BITMAP *bmp, int x, int y);
Récupère la couleur d'un pixel à une position x,y de la bitmap bmp.
Retourne -1 si le point est en dehpors de l'image.

Ces deux fonctions font partie des primitives de dessin et à ce titre :
- utilisent le la profondeur de couleur courante (color depth, 8,15,16,24,32 bits)
- respectent le rectangle de clip de la bitmap de destination (elles n'écrivent pas en dehors)

Un second moyen consiste à accéder directement à la matrice de l'image. Soit une bitmap une position x,y et une couleur color , en 8 bits cela donne :

```
   bmp->line[y][x]=color ;     // affecte une couleur à une position
   color=bmp->line[y][x];      // récupère la couleur à une position
```

Mais pour les modes 16, 32 bits, puisque line est un tableau de char* qui va permettre de définir dynamiquement une matrice de char, un cast est nécessaire ce qui donne :

```
// en 16 bits
((short*)bmp->line[y])[x]=color; // affecte une couleur à une position
color=((short*)bmp->line[y])[x]  // récupère la couleur à une position

// en 32 bits
((long*)bmp->line[y])[x]=color; // affecte une couleur à une position
color=((long*)bmp->line[y])[x]  // récupère la couleur à une position
```

Pour illustrer cet accès aux données de l'image bitmap, le test n°9 propose de binariser une image précédemment loadée. A partir d'un seuil, les valeurs inférieures au seuil passe à la valeur minimum 0 et celles supérieures ou égales passent à la valeur maximum 255. Dans ce test La binarisation est effectuée à partir d'une image en 16 bits et pour chacune des valeurs de rouge, de vert et de bleu constitutives de la couleur de chaque pixel. Initialement le seuil est fixé à 127, mais il peut être modifié à tout moment en pressant la touche F3. La fonction de binarisation prend en paramètre la bitmap source à partir de laquelle le traitement est effectué, une bitmap destination de la même taille dans laquelle le résultat du traitement est sauvegardé et la valeur de seuil à partir de laquelle le traitement est effectué. La fonction de binarisation est appelée avec la touche F2. Pour chaque couleur de pixel les valeurs de rouge, vert et bleu sont obtenues à l'aide des fonctions :

int getr(int c);
int getg(int c);
int getb(int c);
Chacune de ces fonctions extrait, à partir de la couleur passée en paramètre, la composante, rouge ou vert ou bleu qui lui correspond. Les valeurs des composantes sont dans la fourchette de 0 à 255 compris. Ces fonctions opèrent dans le mode de couleur courant du programme, indifféremment 8, 15, 16, 24 ou 32 bits. Il y a dans allegro d'autres fonctions pour récupérer les composantes de couleur en spécifiant le codage de l'image (getr8(), getr15() ..., getb32(), voir la documentation allegro à "Truecolor pixel formats")

```
int r, g, b, color_value;
   color_value = getpixel(screen, 100, 100);
   r = getr(color_value);
   g = getg(color_value);
   b = getb(color_value);
```

Test 2.9 : accéder aux datas d'une image bitmap

```
#include <allegro.h>
#include <time.h>

#define ERREUR(msg){\
   set_gfx_mode(GFX_TEXT,0,0,0,0);\
   allegro_message("err %s\nligne %d\nfile %s\n",msg,__LINE__,__FILE__);\
   allegro_exit();\
   return 1;\
}
/************************************************
************************************************/
void binarisation(BITMAP*scr, BITMAP*dest,int seuil)
{
int color,r,g,b,x,y;

   // parcourir l'image
   for (y=0; y<scr->h; y++)
      for (x=0; x<scr->w; x++){
          // récupérer la couleur de chaque pixel
          color=getpixel(scr,x,y);
          // récupérer chaque composante de couleur
          r=getr(color);
          g=getg(color);
          b=getb(color);
          // rappel opérateur conditionel :
          // resultat = (test vrai) ? val si oui : val si non ;
          // test de binarisation pour chaque composante de couleur
          r= (r<=seuil) ? 0 : 255;
          g= (g<=seuil) ? 0 : 255;
          b= (b<=seuil) ? 0 : 255;
          putpixel(dest,x,y,makecol(r,g,b));
      }
}/************************************************
************************************************/
int main()
{
BITMAP*bmp,*image;
int x,y,seuil;

   allegro_init();
   install_keyboard();
   srand(time(NULL));

   // Couleur en 16 bits
   set_color_depth(16);
   if(set_gfx_mode(GFX_AUTODETECT_WINDOWED,800,600,0,0)!=0)
      ERREUR(allegro_error);

   // récupérer une image dans le programme
   image=load_bitmap("image.bmp",NULL);
   if (!image)
      ERREUR("load_bitmap");

   // créer une bitmap supplémentaire de la même taille
   bmp=create_bitmap(image->w,image->h);
   if (!bmp)
      ERREUR("create_bitmap");
```

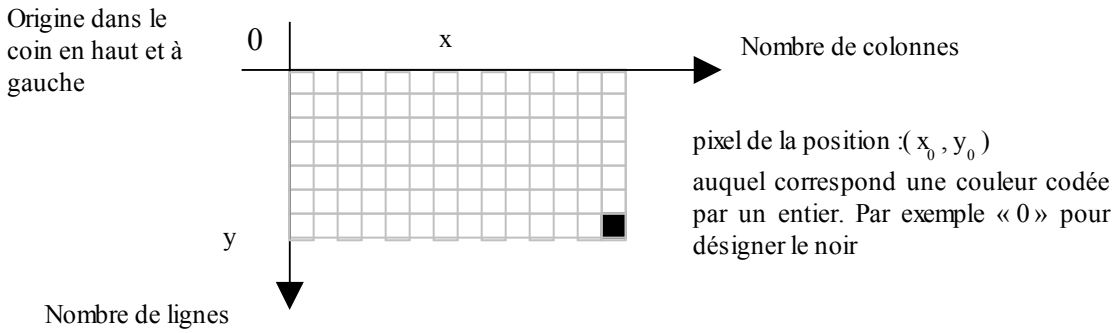
```
   //position pour affichage de l'image et affichage à l'écran
   x=(SCREEN_W-bmp->w)/2;
   y=(SCREEN_H-bmp->h)/2;
   blit(image, screen, 0,0, x, y, bmp->w, bmp->h);

   // initialisation d'un seuil à 127
   seuil=127;

   // contrôle répétition clavier
   set_keyboard_rate(0,0);

   // boucle event
   while(!key[KEY_ESC]){

      if (keypressed()){
         switch(readkey()>>8){

            // affiche image original
            case KEY_F1 :
               blit(image, screen, 0,0, x, y, bmp->w, bmp->h);
               break;

            // traitement binarisation
            case KEY_F2 :
               binarisation(image, bmp,seuil);
               blit(bmp, screen, 0,0, x, y, bmp->w, bmp->h);
               break;

            //modification du seuil
            case KEY_F3 :
            // effacer ancienne valeur
               textprintf_ex(screen,font,x,y+bmp->h+30,
               makecol(0,0,0),-1,"valeur de seuil= %d",seuil);
               // nouvelle valeur
               seuil=rand()%255;
               // affichage
               textprintf_ex(screen,font,x,y+bmp->h+30,
               makecol(255,255,255),-1,"valeur de seuil= %d",seuil);
               break;
         }
      }
   }
   // libérer la mémoire allouée à la sortie
   destroy_bitmap(image);
   destroy_bitmap(bmp);
   return 0;
}
END_OF_MAIN();
```

Résumé C2

MODES COULEUR

Pixel

L'image peut se représenter par une matrice à deux dimensions de nombres. Le pixel est à une position (x,y) un nombre dans cette matrice et correspond au codage d'une couleur.



Codage 15, 16, 24, 32 bits (mode « true colors ») et 8 bits (mode « palette »)

En RVB la couleur apparaît selon un mélange de rouge, vert et bleu (en anglais RGB pour red, green, blue).
Le codage de ce mélange est relatif au nombre de bits donné comme « profondeur » de couleur (color depth).
15,16, 24, 32 bits correspondent aux modes dits « true colors »
8 bits correspond au mode « palette »

En mode true color la couleur est codée sur un short (15 et 16 bits) ou un long (24 et 32 bits)
En mode « palette »
Une couleur est codée à partir d'une structure qui est :
typedef struct RGB{ int r,g,b }RGB ;

Une « palette » de couleurs est un tableau de 256 structures RGB, chaque indice correspondant à une couleur.
typedef RGB PALETTE[256] ;

Sélectionner une profondeur de couleur 8, 15, 16, 24 ou 32 bits PRG 2.1

void set_color_depth(int depth)
Cette fonction appelée avant set_gfx_mode permet de spécifier un format de couleur 15, 16, 24 et 32 bits sont tous reconnus. Sinon par défaut c'est le mode 8bits.

Sous Allegro par défaut c'est le mode 8 bits « palette » : PRG 2.2

Créer une palette de couleurs, il y a autant d'algorithmes de création de palettes que d'imagination pour les créer. Voici un exemple basique d'une palette de niveaux de gris :

```
void gris(PALETTE pal)
{
int i ;
   for (i=0 ;i<256 ;i++)
      pal[i].r=pal[i].g=pal[i].b= i>>2;
   set_palette(pal);
}
```

Rendre opérationnelle une palette créée ou modifiée :

void set_palette(PALETTE pal) ;
cette fonction rend active la palette passée en paramètre.

Animation par rotation de palette

PRG 2.3

On obtient une animation intéressante en faisant circuler les couleurs de la palette.

BITMAP EN MEMOIRE

Structure de données

Les informations d'une image bitmap (littéralement champ ou plan de bits) sont codées avec une structure :

```
typedef struct BITMAP{
   int w, h;              // longueur et hauteur
   int clip;              // clip actif 1 ou non 0
   int cl, ct, cr, cb;    // zone clip comprise entre (cl,ct) et  (cr,cb)
   unsigned char *line[]; // les mêmes datas sous forme matrice
}BITMAP ;
```

Créer, détruire une BITMAP

PRG 2.4

BITMAP *create_bitmap(int width, int height);
Alloue la mémoire pour une bitmap de width sur height. Si problème la fonction retourne NULL. Par défaut le clip est activé et sa zone est la taille de l'image. La profondeur de couleur (colordepth) est 8,15,16 24 ou 32 bits, celle du mode courant.

void destroy_bitmap(BITMAP*bmp)
Libère la mémoire précédement allouée pour une bitmap (tous les types utilisés par allegro : memory bitmap, sub-bitmap, video memory bitmap, or system bitmap , voir doc « bitmap objects »).

Dessiner, placer du texte, effacer une BITMAP

Toutes les primitives de dessin et d'affichage de texte sont utilisables avec les bitmaps pour dessiner et afficher du texte via des pointeurs BITMAP* sur les bitmap de destination.

void clear_bitmap(BITMAP*bmp);
Met toutes les données de l'image à 0

void clear_to_color(BITMAP*bmp, int color) ;
Met toutes les données à la couleur spécifiée par color.

ATTENTION : Toutes les primitives de dessin :
- utilisent le la profondeur de couleur courante (color depth, 8,15,16,24,32 bits)
- respectent le rectangle de clip de la bitmap de destination (elles n'écrivent pas en dehors)

Afficher (copier) une BITMAP : fonction blit()

Des fonctions d'affichage puissantes permettent de recopier des images ou des parties d'images bitmap dans une autre image bitmap toujours désignées avec des pointeurs BITMAP*, e particulier la fonction blit() :

```
void blit(  BITMAP *source, // l'image source
            BITMAP *dest,   // l'image destination
            int source_x,   //coord du point haut-gauche dans l'image
            int source_y,   // source
```

```
int dest_x,     // coordonnée du point haut-gauche dans
int dest_y,     // l'image destination
int width,      // taille pour la copie à partir du point
int height);    // haut-gauche dans l'image source
```

Bouger une BITMAP

PRG 2.5

Principe de base, une boucle :
BOUCLE :
1 effacer la bitmap à l'écran
2 modifier coordonnées de la BITMAP (contrôle des bords éventuels)
3 afficher la bitmap à l'écran aux nouvelles coordonnées
retour en 1

Sauvegarde d'une BITMAP

PRG 2.6

int save_bitmap(const char *filename, BITMAP *bmp, const RGB *pal);
Ecrit la bitmap « bmp » dans un fichier.
Il y a trois formats d'images possibles BMP, PCX, TGA et le nom du fichier « filename » doit contenir l'extension pour le format souhaité (« .bmp », « .pcx » ou « .tga »). Si la bitmap est en 8 bits il faut spécifier sa palette de couleur « pal ».

Faire une photo écran

PRG 2.7

Pour faire une photo d'écran, le plus simple est de copier avec un blit() la zone écran souhaitée dans une bitmap mémoire de la taille voulue et créée avec create_bitmap() puis d'appeler save_bitmap() :

Récupérer une BITMAP (load)

PRG 2.8

BITMAP* load_bitmap(const char *filename, RGB *pal);
Cette fonction charge un fichier bitmap (BMP, LBM, PCX, et TGA) en mémoire à partir du nom et chemin spécifié par la chaîne « filename » et retourne un pointeur sur une structure BITMAP. Si l'image est en 8 bits ses couleurs sont stockées dans une pallette « pal », pour les autres formats de couleur l'argument « pal » peut être NULL. Si le format de l'image ne correspond pas au format courrant du programme l'image est convertie dans le format courant. Retourne NULL si erreur.

Accéder aux datas d'une BITMAP

PRG 2.9

void putpixel(BITMAP *bmp, int x, int y, int color);
Ecrit la couleur color d'un pixel à la position x,y de la bitmap bmp.

int getpixel(BITMAP *bmp, int x, int y);
Récupère la couleur d'un pixel à une position x,y de la bitmap bmp.
Retourne -1 si le point est en dehpors de l'image.

Ces deux fonctions font partie des primitives de dessin et à ce titre :
- utilisent le la profondeur de couleur courante (color depth, 8,15,16,24,32 bits)
- respectent le rectangle de clip de la bitmap de destination (elles n'écrivent pas en dehors)

Accès direct en mémoire :

Soit une bitmap une position x,y et une couleur color :
en 8 bits :

```
bmp->line[y][x]=color ; // affecte une couleur à une position
color=bmp->line[y][x];  // récupère la couleur à une position
```

Pour les modes 16, 32 bits un cast est nécessaire :

```
((short*)bmp->line[y])[x]=color ;      ((long*)bmp->line[y])[x]=color ;
color=((short*)bmp->line[y])[x];       color=((long*)bmp->line[y])[x];
```

1. Avoir des acteurs (formes ou personnages) les animer et les afficher

Quelles sont les variables dont j'ai besoin pour définir un player ? Un ennemi ? Sur quelles « structures de données » mon programme va t-il reposer ?

1.1 Définir des acteurs à l'aide de structures

1.1.1 Exemple des "nénuphs"

Idée du nénuph, concept
Un nénuph est un dérivé de nénuphar qui a acquis de l'autonomie suite à des transformations génétiques de la plante. Le nénuph est une métamorphose de la plante en une sorte de confetti aquatique, cappable de mouvements et d'une vie collective primitive. Voilà pour l'idée de nénuph, maintenant formons un concept de nénuph.

Le nénuph peut se déplacer (1). Il a une position qui peut changer et quand il se déplace il va plus ou moins vite dans une direction ou une autre (2). Sa forme est ronde. Il est plus ou moins gros (3) et il a une couleur (4). Il peut communiquer avec d'autres nénuphs et aussi avec son environnement (5). Éventuellement, il doit trouver de la nourriture. Du fait de cette possibilité de communication, le nénuph n'est pas toujours de bonne humeur (6), en fait ça dépend de son caractère (7). Quoi qu'il en soit l'humeur du nénuph a des répercussions sur ses échanges.

Définition des primitives conceptuelles, implémentation
Chacun des sept traits qui caractérisent ce concept de nénuph peut être assimilé à une simple variable et les relations des nénuphs entre eux, ou des nénuphs avec leur environnement, vont devenir des calculs sur ces variables. La totalité des variables éventuellement de types différents qui définissent le nénuph sont regroupées dans une structure, ce qui donne :

```
     struct nenuph{
(1)     int x,y;
(2)     int dx,dy;
(3)     int rayon;
(4)     int couleur;
(5)     float perception;
(6)     float bonneHumeur;
(7)     int caractere;
     }
```

(1) Variables pour la position du nénuph

(2) variables pour le déplacement du nénuph. Plus les valeurs seront élevées et plus le nénuph se déplacera vite.

(3) Le nénuph est affiché sous la forme d'un cercle. C'est le rayon du cercle qui définit la taille du nénuph à partir de sa position courante. Grâce à ce rayon il sera également possible, par le calcul, de savoir si des nénuphs se superposent ou non.

(4) La couleur du nénuph. Elle peut changer en fonction de l'humeur.

(5) Variable qui correspond à un allongement du rayon du cercle. Cet allongement permet d'avoir une zone autour du nénuph et que le nénuph pourra contrôler. Par exemple il pourra percevoir d'autres nénuphs sans entrer en collision avec eux dés lors qu'ils seront dans sont rayon de perception. De

même sa perception lui permettra de se nourrir et d'appréhender toute chose se trouvant dans sa capacité perceptive.

(6) Définir une caractéristique psychologique du nénuph. Cette caractéristique prendra de l'importance dans les relations avec d'autres nénuphs. Elle peut varier avec différentes causes et être graduée, sensible. Par exemple la bonne humeur peut être proportionnelle à la nourriture trouvée et abordée ou sensible à la qualité du miam. La bonne humeur peut naître ou se tarir en fonction de rencontres imprévues de nénuph ...

(7) La bonne humeur peut être plus ou moins assujettie à un caractère plus ou moins misnénuphobe (haine du nénuph).

Comment faire pour avoir par exemple 50 nénuphs dans le programme ? Le plus simple est d'avoir un tableau de struct nenuph, également il sera plus pratique de définir le type nénuph :

```
#define NB_NENUPH  50
typedef struct nenuph{

  (... ) // contenu de la struct nenuph vue plus haut

}t_nenuph;
t_nenuph  ALL[NB_NENUPH];
```

Eventuellement une liste chainée pourrait être préféré au tableau si le nombre des nénuphs doit beaucoup varier. Dans ce cas il faut ajouter un champs pointeur de type struct nenuph et avoir une ancre initialisée à NULL dés sa déclaration :

```
typedef struct nenuph{

  (... ) // contenu de la struct nenuph vue plus haut
  struct nenuph* suiv;

}t_nenuph;
t_nenuph*prem=NULL;
```

Fonction d'initialisation

L'initialisation consiste à mettre des valeurs de départ pour un nénuph. Admettons que l'on s'appuie sur le hasard, notre fonction d'initialisation aura une allure de ce type :

```
     struct nenuph  init_nenuph()
     {
     struct nenuph n;
(1)     n.x             =rand()%SCREEN_W;
        n.y             =rand()%SCREEN_H;
(2)     n.dx            =(rand()%10)-5;
        n.dy            =(rand()%10)-5;
(3)     n.rayon         =rand()%20+20;
(4)     n.perception    =50-n.rayon;
(5)     n.bonneHumeur   =rand()%100;
(6)     n.caractere     =rand()%10;
(7)     n.couleur       =(n.perception*n.bonneHumeur)%256;
(8)     return n;
     }
```

(1) Initialisation de la position de départ au hasard dans l'écran.

(2) Valeurs de mouvements comprises entre -5 et 5 pour la verticale et l'horizontale.

(3) Taille du nénuph qui sera un cercle de rayon compris entre 20 et 39 pixels

(4) La capacité perceptive des nénuph est inversement proportionnelle à la taille du nénuph. Plus le

nénuph est petit, plus sa perception est grande. Inversement, plus le nénuph est grand, plus sa perception est petite. Nous avons choisi arbitrairement la constante 50 comme repère dans cet effet de proportion.

(5) Pour la bonne humeur nous avons pris une fourchette de 100 valeurs (de 0 à 99) afin d'établir plus tard dans le programme une echelle de la progression de l'humeur. Cette variable permettra d'établir des comparaisons entre les entités, et, sur la base de ces comparaisons, influencer telle ou telle décision dans un sens ou dans un autre. Par exemple, admettons que l'humeur joue un rôle sur l'aptitude à se déplacer du némuph, plus son humeur est sombre (c'est à dire proche de 0) moins il se déplace. D'autre part l'humeur pourrait être communicative et permettre des influences réciproques entre nénuphs ; des influences qui pourraient être pondérées par les traits de caractère.

(6) Si le caractère est bon, il y a une une bonne influence, en revanche, elle peut s'avérer catastrophique... Il serait intéressant de définir des caractères qui serviront à colorer des processus relationnels entre les entités. Admettons qu'il y ait en tout et pour tout 10 possibilités de caractères. Évidement il est envisagable de faire évoluer les caractères des entités. Le caractère d'une entité peut se dégrader ou s'améliorer par exemple en fonction de compositions de couleurs réalisées par l'ensemble de la communauté des nénuphs...

(7) La couleur du nénuph résulte de la bonne humeur mise en relation avec la perception. Comme l'humeur sera amenée à varier, la couleur aussi et donc les caractères également, etc.

(8) La fonction retourne comme valeur une struct nenuph initialisée (les structures sont des variables scalaires ou "left value" c'est à dire qu'elles peuvent faire l'objet d'affectations).

On arrive assez vite à des interactions assez complexes. C'est là qu'intervient la recherche de modèles plus construits des relations entre nos entités. Sinon, on finit par ne plus rien comprendre à son ensemble de variables interconnectées. Il faut alors beaucoup de chance pour arriver à un système qui fasse état d'une cohérence intéressante, et surtout, on ne peut plus agir sur un tel système parce que l'on ne sait plus comment il marche. Il n'est alors plus possible d'établir des comportements et des caractéristiques aux nénuphs.

1.2.2 Structure de données d'un acteur « formes » (rect, cercles, etc.)

Le programme de test que nous allons faire permet de d'animer une cinquantaine de formes géométriques simples à l'écran. Ces formes sont des cercles, des rectangles ou des triangles. Elles n'ont aucune interaction entre elles, elles ne font que se déplacer à des vitesses différentes.

Chaque entité "forme" est caractérisée par un type de forme (cercle, rectangle ou triangle), une ou deux variables de taille pour construire la forme, une position à l'écran, un pas de déplacement horizontal, un pas de déplacement vertical, une couleur. La structure de données pour une forme est la suivante :

```
typedef struct entite{
   int type;         // type de l'entité : 0=rect, 1=cercle, 2=triangle
   int tx,ty;        // variables pour construire la forme de l'entité
   int x,y;          // position horizontal et verticale
   int px,py;        // pour déplacement de l'entité
   int color;        // couleur de l'entité
}t_entite;
```

Pour avoir plusieurs formes dans le programme, il faut un tableau de structures t_entite ou un tableau de pointeurs t_entite

```
#define NBMAX      50
t_entite *tab[NBMAX] ;
```

Remarque langage C :
Le type des entités pourrait être défini avec trois #define :

```
#define RECT       0
#define CERCLE     1
#define TRIANG     2
```

Mais en C nous avons aussi la possibilité de définir une suite de constantes par des mots avec un **enum** de la façon suivante :

```
enum{ RECT, CERCLE, TRIANG, FIN};
```

Par defaut les mots de l'enum sont incrémentés de 1 en 1 à partir de 0. Ici RECT vaut 0, CERCLE 1, TRIANG 2, FIN 3. L'incrémentation est toujours de 1 en 1 mais il est possible de définir un ou plusieurs départ avec des affectations, par exemple :

```
enum{ RECT=3, CERCLE, TRIANG=7, FIN};
```

ici RECT vaut 3, CERCLE vaut 4, TRIANG vaut 7 et FIN vaut 8.

C'est cette syntaxe que nous utiliserons dans le programme qui suit pour les constantes de type de forme.

1.2 Double buffer, mouvements de formes, affichages

1.2.1 Principe du double buffer

Le principe du double buffer est simplement d'agencer l'ensemble des formes que l'on veut mouvoir dans une image bitmap intermédiaire (le double buffer) puis d'afficher d'un seul bloc cette image à l'écran en écrasant ce qui se trouvait précédemment affiché à l'écran. Ce qui donne, dans la boucle d'évènements, les étapes suivantes :

1) Effacer l'image intermédiaire (le buffer).
2) Pour n objets, modifier les positions des objets en fonction de leurs vitesses et trajectoires respectives etc.
3) Afficher les formes (cercles, rectangles triangles, dessins, petites images etc.) à leurs nouvelles positions dans la bitmap intermédiaire.
4) copier cette image à l'écran.

L'image intermédiaire fonctionne un peu comme une table de montage sur laquelle sont assemblés tous les éléments constitutifs du résultat souhaité. La majeure partie de l'affichage se fait en mémoire centrale (RAM) sans passer par l'écran et la mémoire vidéo beaucoup plus lente et l'avantage est un gain de rapidité important. De plus il n'y a pas lieu d'effacer chaque forme. En effet tous les pixels de la bitmap intermédiaire peuvent être mis à zéro en un seul appel d'une fonction d'effacement fournie par l'environnement., clear_bitmap() sous Allegro.

Du point de vue esthétique cette technique donne une animation fluide. Elle fonctionne par plans d'ensemble successifs, sans visualisation de la construction de chaque élément séparé. C'est comme dans un dessin animé traditionnel mais avec la différence importante que le développement de l'animation est calculé en temps réel.

1.2.2 Bouger une forme

Exemple, affichage double buffer d'une forme déplacée au clavier

```
int main()
{
t_entite f;   // une forme
BITMAP*page;  //le double buffer
int vmax=5;   //pour contrôler la vitesse du déplacement

   allegro_init();
```

```
   install_keyboard();
   srand(time(NULL));

   if (set_gfx_mode(GFX_AUTODETECT,ECRAN_X,ECRAN_Y,0,0)!=0)
      ERREUR("gfx_mode");

  // initialisation de la forme (un rectangle)
  // pour l'horizontal
  f.tx=rand()%200+20;          // taille
  f.x=rand()%(ECRAN_X-f.tx*2); // position initiale
  f.px=(rand()%vmax)+1;        // vitesse de déplacement
  // pour la verticale
  f.ty=rand()%200+20;          // taille
  f.y=rand()%(ECRAN_Y-f.ty*2); // position initiale
  f.py=(rand()%vmax)+1;        // vitesse de déplacement
  // couleur
  f.color=makecol(255,0,0);
 ----------------------------------------------------------------------
 // (1) creation d'un buffer, la bitmap intermédiaire
 page=create_bitmap(SCREEN_W,SCREEN_H);
 if (!page)
    ERREUR("creation bitmap");
 clear_bitmap(page);
 ----------------------------------------------------------------------

   while (!key[KEY_ESC]){  // boucle events

      //(2) move
      x+=px;
      y+=py;

      // contrôle orientation : attention vitesse !
      if (key[KEY_UP])
         f.py=(--f.py)%vmax;
      if (key[KEY_DOWN])
         f.py=(++f.py)%vmax;
      if (key[KEY_LEFT])
         f.px=(--f.px)%vmax;
      if (key[KEY_RIGHT])
         f.px=(++f.px)%vmax;

      // contrôle des bords
      if (f.x<0)
         f.px=ABS(f.px);
      if (f.x+f.tx>SCREEN_W)
         f.px=-ABS(f.px);
      if (f.y<0)
         f.py=ABS(f.py);
      if (f.y+f.ty>SCREEN_H)
         f.py=-ABS(f.py);
     ----------------------------------------------------------------
     //(3) affichage
     clear_bitmap(page);
     rectfill(page,f.x,f.y,f.x+f.tx,f.y+f.ty,f.color);
     blit(page,screen,0,0,0,0,page->w,page->h);
   } ----------------------------------------------------------------
   return 0;
}
END_OF_MAIN();
```

(1) Il s'agit de créer le buffer, l'image intermédiaire, ici à la taille de l'écran.

(2) Dans la boucle d'événements, modification des coordonnées de la forme. Si les flèches sont pressées le pas d'avancement est modifié.
(3) Gestion de l'animation :
- la page est effacée
- la forme est affichée dans la bitmap intermédiaire aux nouvelles coordonnées
- une fois le nouveau dessin terminé, la bitmap intermédiaire est affichée à l'écran

1.2.3 Bouger des formes en parallèle

Ce processus d'animation du double buffer permet de mouvoir autant de formes que l'on veut en simultané du point de vue de l'animation. Au lieu de modifier les coordonnées d'une seule forme puis de l'afficher en mémoire puis à l'écran, il suffit d'avoir de nombreuses formes, de modifier leurs coordonnées, de les afficher chacune dans l'image intermédiaire aux nouvelles coordonnées et une fois l'image intermédiaire complète la placarder à l'écran.

Pour ce faire le test n°1 est un programme complet qui anime NB_MAX entités. Nous avons choisi de les regrouper toutes dans un tableau de pointeurs

Test 3.1 : Formes mobiles

```
#include <allegro.h>
#include <time.h>

#define ERREUR(msg){\
   set_gfx_mode(GFX_TEXT,0,0,0,0);\
   allegro_message("err %s\nligne %d\nfile %s\n",msg,__LINE__,__FILE__);\
   allegro_exit();\
   exit(EXIT_FAILURE);\
}

#define NB_MAX        50  // le nombre maximum d'entités dans le programme

// définition d'un acteur dans notre prg :  une entité du type "t_entite"
typedef struct entite{
   int type;         // type de l'entité : 0=rect, 1=cercle, 2=triangle
   int x,y;          // position horizontal et verticale
   int tx,ty;        // variables utilisées pour construire la forme de
l'entité
   int px,py;        // pour déplacement de l'entité
   int color;        // couleur de l'entité
}t_entite;

// pour définir une suite de constantes par un texte
enum{ RECT, CERCLE, TRIANG, FIN};// RECT vaut 0, CERCLE 1, TRIANG 2, FIN 3

// déclaration des trois fonctions
void    init_entite      (t_entite *f);
void    bouge_entite     (t_entite *f);
void    affiche_entite   (BITMAP *bmp, t_entite *f);
/*********************************************************************
*********************************************************************/
int main()
{
t_entite*E[NB_MAX];   // un tableau de pointeurs sur t_entite de NB_MAX
éléments
BITMAP *page;         // le buffer image
int i;

   allegro_init();
   install_keyboard();
```

```
   srand(time(NULL));

   // couleur en 16 bits
   set_color_depth(16);
   // mode graphique fenêtré en 800x600
   if (set_gfx_mode(GFX_AUTODETECT_WINDOWED,800,600,0,0)!=0)
      ERREUR(allegro_error);

   // création buffer
   page=create_bitmap(SCREEN_W,SCREEN_H);
   if (!page)
      ERREUR("create_bitmap");

   // initialisation des entites
   for (i=0; i<NB_MAX; i++){
      E[i]=(t_entite*)malloc(sizeof(t_entite));  // pointeurs donc allouer
      init_entite(E[i]);
   }
   // boucle events et autonomie du programme
   while (!key[KEY_ESC]){
       // animation permanente des entités :
       // 1) effacer buffer
       clear_bitmap(page);
       // pour chaque entite :
       // 2 et 3) bouger et contrôler bords, 4)afficher dans buffer
       for (i=0; i<NB_MAX; i++){
          bouge_entite(E[i]);   // bouger  entite avec contrôle des bords
          affiche_entite(page,E[i]);// afficher entite dans buffer
       }
       // 5 plaquage final à l'écran
       blit(page,screen,0,0,0,0,SCREEN_W,SCREEN_H);

       // si touche enter réinitialisation des entités
       if (key[KEY_ENTER])
          for (i=0; i<NB_MAX; i++)
             init_entite(E[i]);

       rest(5);
   }
   // fin : tout désallouer
   for (i=0; i<NB_MAX; i++)
      free(E[i]);
   destroy_bitmap(page);
   exit(EXIT_SUCCESS);
}
END_OF_MAIN();
/*********************************************************************
initialiser l'entité
*********************************************************************/
#define TXMAX     90      // constantes pour fixer une taille limite
#define TYMAX     90
void init_entite(t_entite *e)
{
int r,g,b;

   // le type
   e->type=rand()%FIN;          // RECT=0, CERCLE=1, TRIANG=2 FIN=3

   // taille
   e->tx = 10+rand()%TXMAX;
```

```
   if (e->type==CERCLE)         // si cercle ty ou tx comme rayon
      e->ty = e->tx;
   else
      e->ty = 10+rand()%TYMAX;  // sinon une surface type rect

   //position
   e->x=rand()%(SCREEN_W-TXMAX);
   e->y=rand()%(SCREEN_H-TYMAX);

   // déplacement
   e->px= (rand()%11)-5; // val entre -5 et 5 pour déplacement horizontal
   e->py= (rand()%11)-5; // idem pour déplacement vertical

   // création couleur avec trois valeurs aléatoires:
   r=rand()%256;
   g=rand()%256;
   b=rand()%256;
   e->color=makecol(r,g,b);
}
/*****************************************************************************
Bouger l'entité
*****************************************************************************/
void bouge_entite(t_entite *e)
{
   // base du mouvement ajouter le pas d'avancement à la position courante
   e->x+=e->px;
   // ensuite contrôle des bords
   if (e->x<0)                 // sortie gauche ? si oui
      e->px= rand()%5;         // avancement forcé positif (changer de sens)
   if (e->x+e->tx > SCREEN_W )// sortie droite ? si oui
      e->px= -1*(rand()%5);    // forcé négatif (changé de sens)

   // idem verticale
   e->y+=e->py;
   if (e->y<0)                 // sortie en haut ? si oui
      e->py= rand()%5;         // avancement forcé positif (changer de sens)
   if (e->y+e->ty > SCREEN_H )// sortie en bas ? si oui
      e->py= -1*(rand()%5);    // forcé négatif (changé de sens)
}
/*****************************************************************************
affichage de chaque entité dans le double buffer en fonction de son type
*****************************************************************************/
void affiche_entite(BITMAP *bmp, t_entite *e)
{
int cx,cy;

   switch(e->type){
      case RECT :
         rectfill( bmp,e->x, e->y,e->x+e->tx, e->y+e->ty, e->color);
         break;

      case TRIANG :
         // trois points : (x,y+ty)  (x+tx, y+ty) (x+tx/2, y)
         triangle( bmp,e->x, e->y+e->ty,
                       e->x+e->tx, e->y+e->ty,
                       e->x+e->tx/2 ,e->y,
                       e->color);
         break;

      case CERCLE :
         cx=e->x+e->tx/2; //tx/2 comme rayon
```

```
         cy=e->y+e->tx/2;
         circlefill( bmp, cx, cy, e->tx/2, e->color);
         break;
   }
}
```

1.2.4 Ajouter une image en fond

Nous allons maintenant ajouter au programme 3.1 précédent une image en fond de façon à ce que les formes se déplacent sur une image. Deux choses changent : 1) il faut récupérer une image pour le fond dans le programme et 2) plutôt que d'effacer le buffer, si l'image à la taille de l'écran, il suffit de copier cette image dans le buffer avant d'y placer les formes.

Test 3.2 : formes mobiles sur une image

```
// attention se reporter au test 3.1 pour avoir le programme complet.

int main()
{
t_entite*E[NB_MAX];   // tableau de pointeurs sur t_entite de NB_MAX
éléments
BITMAP *page;         // le buffer image
BITMAP*fond;          // pour récupérer l'image du fond
int i;

   allegro_init();      // obligatoire au début du programme
   install_keyboard();  // pour avoir le clavier
   srand(time(NULL));   // pour suite de nombrea aléatoires

   // couleur en 16 bits
   set_color_depth(16);
   // mode graphique fenêtre n 800x600
   if (set_gfx_mode(GFX_AUTODETECT_WINDOWED,800,600,0,0)!=0)
      ERREUR(allegro_error);

   // création buffer
   page=create_bitmap(SCREEN_W,SCREEN_H);
   // load image fond
   fond=load_bitmap(".//images//eau.bmp",NULL);
   if (!page ||!fond)
      ERREUR("create_bitmap");
   // initialisation des entites
   for (i=0; i<NB_MAX; i++){
      E[i]=(t_entite*)malloc(sizeof(t_entite));
      init_entite(E[i]);
   }
   // boucle events et autonomie du programme
   while (!key[KEY_ESC]){
       // une animation permanente des entités
      // 1 effacer buffer avec image fond
      blit(fond,page,0,0,0,0,page->w,page->h);
       // pour chaque entite :
       // 2 et 3) bouger et contrôler bords, 4)afficher dans buffer
       for (i=0; i<NB_MAX; i++){
          bouge_entite(E[i]);       // bouger  entite, contrôle des bords
```

```
            affiche_entite(page,E[i]);// afficher entite dans buffer
         }
         // 5 plaquage final à l'écran
         blit(page,screen,0,0,0,0,SCREEN_W,SCREEN_H);

         // si touche enter réinitialisation des entités
         if (key[KEY_ENTER])
            for (i=0; i<NB_MAX; i++)
               init_entite(E[i]);

         // ralentir pour voir
         rest(5);
      }
      // fin : tout désallouer
      for (i=0; i<NB_MAX; i++)
         free(E[i]);
      destroy_bitmap(page);
      destroy_bitmap(fond);
      exit(EXIT_SUCCESS);
}
END_OF_MAIN();
```

L'image n'aura pas nécessairement la taille du fond, dans ce cas il y a la possibilité de l'étirer ou de la compresser. Allegro fournit une fonction dérivée de la fonction blit() qui va nous permettre de modifier la taille d'une image .

1.3 Quelques manipulations d'affichage

1.3.1 Étirer compresser l'image du fond (homothétie)

void stretch_blit(BITMAP *source, BITMAP *dest,
int source_x, source_y, source_width,source_height,
int dest_x, dest_y, dest_width, dest_height);

La fonction stretch_blit ()permet de donner à une image les dimensions que l'on souhaite en réalisant sur elle une homotétie. Les paramètres « source » et « dest » sont respectivement les images source et destination, les variables source_... correspondent à la zone de l'image source que l'on veut copier mais attention il ne faut pas dépasser la taille de l'image source.
Les variables dest_... la taille de la zone de destination qui peut elle dépasser la taille de l'image destination.
Une homotétie est réalisée de la taille source à la taille destination. Source et destination doivent être de la même color depth, la source doit être une bitmap mémoire.

Nous allons ajouter au programme 3.1 la possibilité d'élargir ou de rétrécir l'image du fond avec l'utilisation des flèches et en même que se déroule l'animation des formes.

Il y a plusieurs petits changements. La taille de l'image de fond et sa position dépend maintenant de quatre variables et les valeurs de ces variables augmentent la taille de l'image lorsque l'on appuie sur les flèches, elles la diminuent si on appuie simultanément sur contrôle et flèche. Contrairement au programme 3.2 précédent il faut maintenant d'effacer le buffer parce que l'image du fond peut être plus petite que l'écran si l'utilisateur la compresse.

Test 3.3 : étirer-compresser l'image de fond

```
int main()
{
t_entite*E[NB_MAX];   // un tableau de pointeurs sur t_entite de NB_MAX
éléments
```

```
BITMAP *page;         // le buffer image
BITMAP*fond;          // pour récupérer l'image du fond
int x,y,tx,ty;        // variables pour contrôler l'affichage du fond
int i;

   allegro_init();      // obligatoire au début du programme
   install_keyboard();  // pour avoir le clavier
   srand(time(NULL));   // pour suite de nombrea aléatoires

   // couleur en 16 bits
   set_color_depth(16);
   // mode graphique fenêtre n 800x600
   if (set_gfx_mode(GFX_AUTODETECT_WINDOWED,800,600,0,0)!=0)
      ERREUR(allegro_error);

   // création buffer
   page=create_bitmap(SCREEN_W,SCREEN_H);
   // load image fond
   fond=load_bitmap(".//images//eau.bmp",NULL);
   if (!page)
      ERREUR("create_bitmap");

   // initialisation des entites
   for (i=0; i<NB_MAX; i++){
      E[i]=(t_entite*)malloc(sizeof(t_entite));
      init_entite(E[i]);
   }
  // initialisation taille fond départ pour homothétie
   x=0;
   y=0;
   tx=page->w;
   ty=page->h;
   // boucle events et autonomie du programme
   while (!key[KEY_ESC]){
       // une animation permanente des entités
      // 1 effacer buffer : l'image de fond est peut-être plus petite
      // que l'écran
      clear_bitmap(page);
      // et afficher image de fond (cause si le fond devient plus petit)
      stretch_blit(fond,page,0,0,fond->w,fond->h,x,y,tx,ty);
      // pour chaque entite :
      // 2) bouger et 3)contrôler bords, 4)afficher dans buffer
      for (i=0; i<NB_MAX; i++){
         bouge_entite(E[i]); // bouger  entite avec contrôle des bords
         affiche_entite(page,E[i]);// afficher entite dans buffer
      }
      // 5 plaquage final à l'écran
      blit(page,screen,0,0,0,0,SCREEN_W,SCREEN_H);

      // si touche enter réinitialisation des entités
      if (key[KEY_ENTER])
         for (i=0; i<NB_MAX; i++)
            init_entite(E[i]);

     // jouer à déformer l'image du fond :
```

```
      // contrôle à partir du bord gauche
      if (key[KEY_LEFT]) {
         if (key_shifts & KB_SHIFT_FLAG){// avec shift compresser
            x++;
            tx--;
         }
         else{                          // sans, étirer vers gauche
            x--;
            tx++;
         }
      }
      // contrôle à partir du bord haut
      if (key[KEY_UP]) {
         if (key_shifts & KB_SHIFT_FLAG){ // avec shift compresser
            y++;
            ty--;
         }
         else{
            y--;                        // sans étirer vers haut
            ty++;
         }
      }
      // contrôle à partir du bord droit
      if (key[KEY_RIGHT]) {
         if (key_shifts & KB_SHIFT_FLAG)// avec shift compresser
            tx--;
         else                           // sans étirer vers droite
            tx++;
      }
      // contrôle à partir du bord bas
      if (key[KEY_DOWN]) {
         if (key_shifts & KB_SHIFT_FLAG)// compresser
            ty--;
         else                           // sans étirer ves bas
            ty++;
      }
      // retrouver les proportions du départ
      if (key[KEY_SPACE]){
         x=0;
         y=0;
         tx=page->w;
         ty=page->h;
      }
       // ralentir pour voir
       rest(5);
   }
   // fin : tout désallouer
   for (i=0; i<NB_MAX; i++)
      free(E[i]);
   destroy_bitmap(page);
   destroy_bitmap(fond);
   exit(EXIT_SUCCESS);
}
END_OF_MAIN();
```

1.3.2 Voir à travers une forme : modifier le mode d'affichage

Les formes du programme précédent sont des formes de couleur animées et projetées à l'écran. Mais il y a moyen de modifier le mode d'affichage afin par exemple de faire courir la forme sur une image et

de voir l'image à travers elle comme à travers une serrure. A titre indicatif l'environnement propose six modes d'affichage, chacun désigné par une macro constante :

DRAW_MODE_SOLID,
DRAW_MODE_COPY_PATTERN,
DRAW_MODE_SOLID_PATTERN,
DRAW_MODE_MASKED_PATTERN,
DRAW_MODE_TRANS,
DRAW_MODE_XOR

Le mode graphique est fixé dans le programme par un appel à la fonction :

void drawing_mode(int mode, BITMAP *pattern, int x_anchor, int y_anchor);
Chacun des modes d'affichage détermine un mode de remplacement des pixels lorsqu'ils sont modifiés par une fonction de tracé de formes géométriques (pixels, lignes, rectangles, triangles, cercles, polygones et même une fonction de remplissage « floodfill »). Nous n'allons pas les présenter tous mais seulement les deux dont nous avons besoin pour un tracé normal et un déplacement de formes géométriques sur des images.

DRAW_MODE_SOLID est le mode par défaut de coloration normale. La couleur des pixels concernés par un tracé est remplacée par la couleur passée en argument à la fonction de tracé.

DRAW_MODE_COPY_PATTERN est le mode dans lequel les pixels sont copiés à partir d'une image de référence, comme un plan de référence. Si par exemple un rectangle est tracé à l'écran, les pixels qui le constituent prennent la couleur des pixels aux positions correspondantes de l'image de référence. Dans ce cas la couleur passée en argument à la fonction de tracé est ignorée. Ce mode présente l'avantage d'être le plus rapide.

A titre indicatif, toujours dans le cadre d'une utilisation d'image d'arrière plan (pattern) il y a le DRAW_MODE_SOLID_PATTERN et le DRAW_MODE_MASKED_PATTERN.

Le premier consiste à comparer la couleur des pixels de l'image de référence avec la couleur utilisée comme masque (ce qui est de la couleur « masque » n'est pas affiché). Cette dernière est 0 en 256 couleurs et rose en vraie couleur (maximum de rouge et de bleu et zéro vert). Si la couleur trouvée n'est pas celle de masque alors le pixel prend la couleur passée en argument à la fonction de tracé. Dans le cas contraire, il prend la couleur 0, noire. L'image de référence agit comme un filtre. Les formes ne sont visibles que sur les parties de couleur différentes de la couleur masque.
Le second, DRAW_MODE_MASKED_PATTERN est équivalent au précédent sauf que les pixels trouvés de la couleurs du masque dans l'image « pattern » seront laissés tel quel et non passés à zéro.

Ainsi dans le programme 3.1 des formes mobiles, pour que l'on voit une image à travers les formes, il y a juste à ajouter le fait de récupérer une image et d'appeler la fonction drawing_mode(), c'est à dire les lignes suivantes :

```
BITMAP* pat;
   //  mon_image.bmp doit être dans le répertoire du programme
   pat  = load_bitmap(".\\mon_image.bmp",NULL);
   if(! image )
      ERREUR("creation image");

   drawing_mode ( DRAW_MODE_COPY_PATTERN, image, 0, 0);
```

A partir de cet appel de la fonction « drawing_mode » le mode d'affichage graphique change dans le programme . il prend comme image de référence (pattern) l'image « pat » loadée juste avant. Le mode d'affichage peut être changé très rapidement et très souvent dans le programme.

Exemple, une image est vue à travers un rectangle qui glisse sur elle.

```
int main()
{
BITMAP *pat,*page;
int x,y,px,py;
```

```
   allegro_init();
   install_keyboard();
   srand(time(NULL));

   // mode couleur 16 bits (mais tous les autres modes sont valables)
   set_color_depth(16);
   if (set_gfx_mode(GFX_AUTODETECT, ECRAN_X, ECRAN_Y, 0, 0) != 0)
      ERREUR(allegro_error);

   // le buffer
   page=create_bitmap(SCREEN_W,SCREEN_H);
   // load du "pattern", l'image d'arrière plan qui va servir au
   // remplissage des  fonctions de dessin.

   pat=load_bitmap("pat1.bmp",NULL);
   if (!pat||!page)
      ERREUR("load image");

   // position du rectangle baladeur
   x=rand()%(SCREEN_W-60);
   y=rand()%(SCREEN_H-100);
   // pas d'avancement du rectangle baladeur
   px=rand()%11-5;
   py=rand()%11-5;

   // choix du mode de dessin
   drawing_mode(DRAW_MODE_COPY_PATTERN, pat, 0, 0);

   // boucle d'événements
   while (!key[KEY_ESC]){

      // effacement page
      clear_bitmap(page);

      // affichage du rectangle : la couleur est inutile
      rectfill(page,x,y,x+60,y+100,0);

      // modification des pas d'avancement
      if (key[KEY_LEFT])   px--;
      if (key[KEY_RIGHT])  px++;
      if (key[KEY_UP])     py--;
      if (key[KEY_DOWN])   py++;
      // modification des coordonnées
      x+=px;
      y+=py;

      // contrôles des bords
      px=(x<0)? ABS(px):(x>SCREEN_W-60)? -ABS(px) : px;
      py=(y<0)? ABS(py):(y>SCREEN_H-100)? -ABS(py) : py;

      // affichage à l'écran
      blit(page,screen,0,0,0,0,page->w,page->h);
      rest(30);
   }
   destroy_bitmap(pat);
   destroy_bitmap(page);
   return 0;
}
END_OF_MAIN();
```

1.3.3 Effet de transparence

A titre indicatif, voici un exemple pour l'utilisation du mode transparence sous allegro. C'est un mode qui permet de voir à travers une image selon un filtre à imaginer et à calculer dans le programme. Ce mode gourmand en calcul ne fonctionne qu'en 32 bits. La documentation d'allegro n'est pas très riche en explication sur ce sujet et il n'est pas facile de comprendre complètement de quoi il s'agit (ce reporter à "Transparency and patterned drawing" dans la documentation). Dans ce genre de situation je préconise d'être pragmatique et de s'appuyer sur un exemple de code pas trop compliqué.
Essentiellement , il y a deux couples d'appels de fonctions à mettre en place.

1) en dehors de la boucle d'évènements, il faut appeler la fonction set_write_alpha_blender().

void set_write_alpha_blender();
Cette fonction active le mode d'édition du canal alpha et passe une image RGB 32 bits au format RGBA. Après l'appel de cette fonction il est possible de paramétrer le drawing mode à transparence avec un appel de la fonction drawing_mode() et la macro DRAW_MODE_TRANS. Dans l'exemple ci-dessous nous aurons :

```
set_write_alpha_blender();
drawing_mode(DRAW_MODE_TRANS, NULL, 0, 0);
```

2) Ensuite dans la boucle d'évènements, nous utilisons les fonctions :

void set_alpha_blender();
Si le mode d'édition canal alpha est activé cette fonction permet d'activer le mode de dessin utilisé par les images en RGBA. Ensuite il est possible d'utiliser la fonction draw_trans_sprite() d'affichage en transparence

void draw_trans_sprite(BITMAP*bmp, BITMAP*sprite, int x, int y);
Permet d'afficher une image en transparence si le mode est correctement initialisé.

Ce qui nous donne dans l'exemple dans l'exemple ci-dessous :

```
set_alpha_blender();
draw_trans_sprite(page, imTrans, x, y);
```

Exemple , effet de transparence

Le programme affiche une image en fond et à partir d'une image de référence crée une autre image transparente selon un certain degré. Ensuite dans la boucle d'évènements l'image transparente est promenée avec les coordonnées de la souris.

```
#include <allegro.h>

#define ERREUR(msg){\
   set_gfx_mode(GFX_TEXT,0,0,0,0);\
   allegro_message("err %s\nligne %d\nfile %s\n",msg,__LINE__,__FILE__);\
   allegro_exit();\
   exit(EXIT_FAILURE);\
}

int main()
{
int x, y, c, a;
BITMAP *fond,*imRef,*imTrans,*page;
    //initialiser
    allegro_init();
    install_keyboard();
    install_mouse();

    set_color_depth(32);
```

```
    if(set_gfx_mode(GFX_AUTODETECT_WINDOWED, 800, 600, 0, 0)!=0)
       ERREUR(allegro_error);

    //load de l'image de fond
    fond = load_bitmap(".//images//eau.bmp", NULL);
    if (!fond)
       ERREUR("load fond");
    //load de l'image qui va devenir transparente
    imRef = load_bitmap(".//images//imRef.bmp", NULL);
    if (!imRef)
       ERREUR("load imRef");
    // image finale transparente
    imTrans = create_bitmap(imRef->w, imRef->h);
    if (!imTrans)
       ERREUR("create_bitmap");

    //initialiser le drawing mode(alpha channel blend)
    drawing_mode(DRAW_MODE_TRANS, NULL, 0, 0);
    set_write_alpha_blender();

    //créer un filtre
    for (y=0; y<imRef->h; y++) {
        for (x=0; x<imRef->w; x++) {
            //récup de la couleur de chaque pixel dans l'iamge référence
            c = getpixel(imRef, x, y);
            a = getr(c) + getg(c) + getb(c);
            //calculer une transparence moyenne à 50%
            a = a/3;
            //copier la couleur résultante dans l'image transparente
            putpixel(imTrans, x, y, a);
        }
    }

    //initialiser un double buffer
    page = create_bitmap(SCREEN_W, SCREEN_H);

    //afficher l'image de fond
    blit(fond, page, 0, 0, 0, 0, SCREEN_W, SCREEN_H);

    while (!key[KEY_ESC])
    {
        //recup des coordonnées souris
        x = mouse_x - imTrans->w/2;
        y = mouse_y - imTrans->h/2;
        //dessiner l'image transparente
        set_alpha_blender();
        draw_trans_sprite(page, imTrans, x, y);

        //afficher le buffer à l'écran
        blit(page, screen, 0, 0, 0, 0, SCREEN_W, SCREEN_H);

        //raffraichir le fond à la position occupée par l'image
        blit(fond, page, x, y, x, y, imTrans->w,imTrans->h);
    }
    destroy_bitmap(fond);
    destroy_bitmap(imTrans);
    destroy_bitmap(page);
    destroy_bitmap(imRef);
    return 0;
}
END_OF_MAIN();
```

1.4 Remplacer les formes (rect, cercles etc.) par des images

1.4.1 Affichage en mode masque

Dans une image tous les pixels de la couleur du masque peuvent être ignorés lors des affichages. Ces couleurs sont 0 en 8 bits (en général le noir) et rose (maximun rouge et bleu, 0 vert) en true color. En d'autres termes, si j'ai une image de bonhomme sur un fond noir je peux le déplacer sur un décor sans que le fond noir apparaisse. Tout ce qui dans mon image est de la couleur du masque devient transparent, on voit au travers, le bonhomme sera parfaitement découpé dans le décor où il évolue.
Les fonctions d'affichage en mode masque fondamentales sont :

void masked_blit(BITMAP *source, BITMAP *dest,
int source_x, int source_y,
int dest_x, int dest_y,
int width, int height);

Elle est identique à la fonction blit() si ce n'est qu'elle préserve tous les pixels transparents de la couleur du masque. Les deux bitmaps source et destination doivent être dans le même mode couleur (8,15,16,24 ou 32 bits).

void masked_stretch_blit(BITMAP *source, BITMAP *dest,
int source_x, source_y, source_w, source_h,
int dest_x, dest_y, dest_w, dest_h);

Identique à la fonction stretch_blit() mais en mode masque cette fois.

La librairie Allegro met à notre disposition une famille de fonctions plus pratiques pour l'affichage des images bitmap souvent appelées "sprites" dans l'univers du jeu vidéo. Il y a en particulier la fonction :

void draw_sprite(BITMAP *bmp, BITMAP *sprite, int x, int y);
C'est la principale fonction pour l'affichage de sprites. Elle dessine une copie de la bitmap « sprite » dans la bitmap de destination « bmp » à la position (x,y). C'est équivalent à :

```
masked_blit(sprite, bmp, 0, 0, x, y, sprite->w, sprite->h);
```

La fonction draw_sprite() est toujours en mode masque et tous les pixels de la couleur du masque sont ignorés, les pixels de la couleur 0 en mode 256 couleurs et rose (max rouge et bleu, 0 vert) dans les autres modes.
Cette fonction permet également d'avoir le sprite en mode 256 couleurs et la bitmap de destination en truecolor ce qui permet d'utiliser des effets de palettes avec le sprite dans le mode true color.

Dans le test 3.4 nous reprenons la base du programme 3.1 avec quelques modifications. Dans la structure entité, la couleur est remplacée par une bitmap. Les entités utilisent la même image, la même adresse d'image leur est affectée. Mais chaque entité a sa propre taille et l'image est pour chacune déformée. Lors du contrôle des bords si une entité sort elle est renvoyée dans l'autre sens avec une nouvelle vitesse de déplacement. Par ailleurs afin de mettre en évidence le mode masque, pour l'affichage des entités il y a possibilité en appuyant sur F1 de basculer entre une image dont le fond correspond à la couleur du masque et la même image mais dont le fond ne correspond pas à la couleur du masque . La souris permet de piloter une image qui remplace le curseur.

Test 3.4 : mode masque et le cavalier

```
#include <allegro.h>
#include <time.h>

#define ERREUR(msg){\
   set_gfx_mode(GFX_TEXT,0,0,0,0);\
   allegro_message("err %s\nligne %d\nfile %s\n",msg,__LINE__,__FILE__);\
   allegro_exit();\
   exit(EXIT_FAILURE);\
}
#define NB_MAX        50  // le nombre maximum d'entités dans le programme
```

```
// définissions d'un acteur dans notre prg :  une entité du type "t_entite"
typedef struct entite{
   BITMAP*im;          // image pour l'entité
   int x,y;           // position horizontal et verticale
   int tx,ty;         // variables utilisées pour la taille de l'entité
   int px,py;         // pour déplacement de l'entité

}t_entite;

// déclaration des fonctions
void    init_entite      (t_entite *e, BITMAP*im);
void    bouge_entite     (t_entite *e);

*************************************************************************
*************************************************************************/
int main()
{
t_entite*E[NB_MAX];   // tableau de pointeurs de NB_MAX t_entite*
BITMAP *page;         // le buffer image
BITMAP*fond;          // pour récupérer l'image du fond
BITMAP *masque, *non_masque, *souris; // pour test mode masque avec souris
int x,y,tx,ty;
int i,cmpt=0;;

   allegro_init();      // obligatoire au début du programme
   install_keyboard();  // pour avoir le clavier
   install_mouse();     // pour avoir la souris
   srand(time(NULL));   // pour suite de nombrea aléatoires

   // couleur en 16 bits
   set_color_depth(16);
   // mode graphique fenetre n 800x600
   if (set_gfx_mode(GFX_AUTODETECT_WINDOWED,800,600,0,0)!=0)
      ERREUR(allegro_error);

   // création buffer
   page=create_bitmap(SCREEN_W,SCREEN_H);
   // load image fond
   fond=load_bitmap(".//images//eau.bmp",NULL);
   // load les 2 sprite avec et sans masque
   masque=load_bitmap(".//images//masque_truemode.bmp",NULL);
   non_masque=load_bitmap(".//images//non_masque.bmp",NULL);
   // au départ
   souris=non_masque;
   // erreurs ?
   if (!page ||!fond||!masque||!non_masque)
      ERREUR("create ou load_bitmap");

   // initialisation des entites
   for (i=0; i<NB_MAX; i++){
      E[i]=(t_entite*)malloc(sizeof(t_entite));
      // ici toutes les entités ont la même image (mais possibilité que
      // chacune ait sa propre image)
      init_entite(E[i],souris);
   }
   // boucle events et autonomie du programme
   while (!key[KEY_ESC]){

       // effacer buffer avec fond
       stretch_blit(fond,page,0,0,fond->w,fond->h,0,0,page->w,page->h);
```

```
        // sélection image
        if(key[KEY_F1]){
           if((++cmpt)%2)
              souris=masque;
           else
              souris=non_masque;
           for (i=0; i<NB_MAX; i++)
              E[i]->im=souris;
           while(key[KEY_F1]){} // bidouille pour éviter répétitions
        }
        // mise à jour entités
        for (i=0; i<NB_MAX; i++){
           bouge_entite(E[i]);
           x=E[i]->x;
           y=E[i]->y;
           tx=E[i]->tx;
           ty=E[i]->ty;
           stretch_sprite(page, E[i]->im, x, y, tx, ty);
           //équivalent à :
           //masked_stretch_blit(E[i]->im,page,
                                 0,0,E[i]->im->w,E[i]->im->h,x,y,tx,ty);
        }
        // récup coordonnées souris
        x=mouse_x;
        y=mouse_y;
        // l'image comme curseur
        masked_blit(souris,page,
                    0,0,x-souris->w/2 ,y-souris->h/2 ,souris->w,souris->h);

        // plaquage final à l'écran
        blit(page,screen,0,0,0,0,SCREEN_W,SCREEN_H);

        // ralentir pour voir
        rest(5);
    }
    // fin : tout désallouer
    for (i=0; i<NB_MAX; i++)
       free(E[i]);
    destroy_bitmap(page);
    destroy_bitmap(fond);
    destroy_bitmap(masque);
    destroy_bitmap(non_masque);
    exit(EXIT_SUCCESS);
}
END_OF_MAIN();
/*********************************************************************
initialiser l'entité
*********************************************************************/
#define TXMAX     90
#define TYMAX     90

void init_entite(t_entite *e, BITMAP*im)
{
    // l'image
    e->im=im;
    // taille
    e->tx = 10+rand()%TXMAX;
    e->ty = 10+rand()%TYMAX;
    //position
    e->x=rand()%(SCREEN_W-TXMAX);
    e->y=rand()%(SCREEN_H-TYMAX);
```

```
   // déplacement
   e->px= (rand()%11)-5; // val entre -5 et 5 pour déplacement horizontal
   e->py= (rand()%11)-5; // idem pour déplacement vertical

}
/*****************************************************************************
Mouvements des entités
*****************************************************************************/
void bouge_entite(t_entite *e)
{
   // base du mouvement ajouter le pas d'avancement à la position courante
   e->x+=e->px;
   // contrôl du bord, sortie gauche
   if (e->x<0)
      e->px= rand()%5;       // forcé positif,change sens et vitesse
   // contrôle du bord droit
   if (e->x+e->tx > SCREEN_W )
      e->px= -1*(rand()%5);  // forcé négatif, change sens et vitesse
   e->y+=e->py;
   // contrôle du bord haut
   if (e->y<0)
      e->py= rand()%5;      // forcé positif, change sens et vitesse
   // contrôle du bord bas
   if (e->y+e->ty > SCREEN_H )
      e->py= -1*(rand()%5); // forcé négatif, change sens et vitesse
}
```

1.4.2 Étirement, renversement, rotations, pivot

Pour l'affichage des bitmaps la librairie Allegro fournit un ensemble d'autres fonctions qui permettent différents effets.

void stretch_sprite(BITMAP *bmp, BITMAP *sprite, int x, int y, int w, int h);
Identique à draw_sprite() mais en plus l'image est étirée ou compressée selon la longueur w et la largeur h. Les bitmaps source et destination doivent être dans le même mode couleur et l'image sprite doit être une bitmap mémoire RAM, et non vidéo ou autre.

void draw_sprite_v_flip(BITMAP *bmp, BITMAP *sprite, int x, int y);
void draw_sprite_h_flip(BITMAP *bmp, BITMAP *sprite, int x, int y);
void draw_sprite_vh_flip(BITMAP *bmp, BITMAP *sprite, int x, int y);
Ces fonctions sont identiques à draw_sprite() mais elles réalisent en plus un renversement de l'image, comme avec un miroir et en fonction de l'axe vertical, horizontal ou les deux ensemble. l'image sprite doit être une bitmap mémoire RAM.

void rotate_sprite(BITMAP *bmp, BITMAP *sprite, int x, int y, fixed angle);
Cette fonction copie la bitmap "sprite" dans la bitmap "bmp". Le coin haut-gauche de l'image sprite est à la position (x,y) et, en fonction de l'angle donné par « angle », l'image sprite subit une rotation autour de son centre : la position (x,y) reste invariable, c'est seulement l'affichage qui est changé.

L'angle est donné dans le type particulier à la librairie Allegro « fixed » qui code un nombre à virgule sur un entier avec 16 bits pour la partie entière et 16 bits pour la partie décimale 16.16 soit un nombre entre -32768 et 32767. Comme il s'agit d'un type non standard et les conversions avec les types standards font appel à des fonctions spécifiques fournies par la librairie (voir la documentation à « fixed point math routines »).

En particulier :
fixed itofix(int x); convertit un entier en fixed (identique à x<<16)
int fixtoi(fixed x); Converti un fixed en entier avec arrondi (0.51 donne 1)

Le cercle est divisé en 256 unités et tourne dans le sens des aiguilles d'une montre. Pour remonter de 90° il faut passer la valeur -64, pour remonter de 180° c'est le double, -128 etc. Dans l'autre sens ce sont des valeurs positives. On peut parcourir le cercle avec uniquement des puissances de 2, ce qui donne :
complet 1 avec un pas de 256
divisé en 2 avec un pas de 128
divisé en 4 avec un pas de 64
divisé en 8 avec un pas de 32
divisé en 16 avec un pas de 16
divisé en 32 avec un pas de 8
divisé en 64 avec un pas de 4
divisé en 128 avec un pas de 2
divisé en 256 avec un pas de 1

Les fonctions d'affichage avec rotation sont utilisables quelques soient le type des bitmaps source et destination (RAM, vidéo …) ou le mode couleur de ces bitmap (8…32 bits)

void rotate_sprite_v_flip(BITMAP *bmp, BITMAP *sprite, int x, int y, fixed angle);
Identique à rotate_sprite() mais bascule en miroir le sprite verticalement. Pour un effet de miroir uniquement horizontal utiliser draw_sprite() et ajouter 128 (avec l'expression itofix(128)) à l'angle.
Pour basculer dans les deux plans vertical et horizontal utiliser cette fonction et ajouter 128 (avec l'expression itofix(128)) à l'angle.

void rotate_scaled_sprite(BITMAP *bmp, BITMAP *sprite, int x, int y, fixed angle, fixed scale);
Identique à rotate_sprite mais étire ou compresse l'image en même temps selon le paramètre d'échelle « scale ».

void rotate_scaled_sprite_v_flip(BITMAP *bmp, BITMAP *sprite, int x, int y, fixed angle, fixed scale)
Dessine le sprite avec modification de l'echelle comme pour rotate_scaled_sprite() mais le sprite est basculé verticalement en miroir d'abord

void pivot_sprite(BITMAP *bmp, BITMAP *sprite, int x, int y, int cx, int cy, fixed angle);
Identique à rotate_sprite() mais permet une rotation autour d'un pivot (cx,cy) quelconque y compris en dehors de l'image sprite.

void pivot_sprite_v_flip(BITMAP *bmp, BITMAP *sprite, int x, int y, int cx, int cy, fixed angle);
Comme rotate_sprite_v_flip(), baculement en miroir vertical de l'image sprite mais avec en plus une rotation autour du pivot (cx,cy)

void pivot_scaled_sprite(BITMAP *bmp, BITMAP *sprite, int x, int y, int cx, int cy, fixed angle, fixed scale));
Comme rotate_scaled_sprite() mais en plus le sprite pivote autour de la position (cx,cy)

void pivot_scaled_sprite_v_flip(BITMAP *bmp, BITMAP *sprite, int x, int y, int cx, int cy, fixed angle, fixed scale)
Comme rotate_scaled_sprite_v_flip() mais en plus pivote autour de la position (cx,cy)

Exemple : une image de tank tourne autour d'un pivot mobile

Voici un exemple de code qui utilise la fonction pivot_sprite(). Une image de tank tourne autour d'un pivot et le point de pivot peut être déplacé en appuyant sur les flèches.

```
#include <allegro.h>
#include <time.h>

#define ERREUR(msg) {\
   set_gfx_mode(GFX_TEXT,0,0,0,0);\
   allegro_message("Error : \n%s\n", msg);\
   allegro_exit();\
```

```
   return 1;\
}
// deux couleurs
#define BLANC   makecol(255,255,255)
#define ROUGE   makecol(255,128,64)

int main()
{
BITMAP *tank;
BITMAP *page;
float angle=0;
int x,y,pivx,pivy;

   allegro_init();
   install_keyboard();
   set_color_depth(16);
   if (set_gfx_mode(GFX_AUTODETECT, 800, 600, 0, 0) != 0)
      ERREUR(allegro_error);

   // load image : attention mettre une image dans le répertoire du
   //programme et le bon nom
   tank=load_bitmap("tank_1.bmp",NULL);

   // double buffer
   page=create_bitmap(SCREEN_W,SCREEN_H);

  if (!tank||!page)
     ERREUR("load image");

   // départ : l'image au centre, le pivot au centre de l'image
   x=SCREEN_W/2;
   y=SCREEN_H/2;
   pivx=tank->w/2;
   pivy=tank->h/2;

   while (!key[KEY_ESC]){

      // le pivot est contrôlé pour rester dans l'image
      if (key[KEY_LEFT]){
         pivx-=2;
         pivx=(pivx<0)?0:pivx;
      }
      if (key[KEY_RIGHT]){
         pivx+=2;
         pivx=(pivx>tank->w-1)?tank->w-1:pivx;
      }
      if (key[KEY_UP]){
         pivy-=2;
         pivy=(pivy<0)?0:pivy;
      }
      if (key[KEY_DOWN]){
         pivy+=2;
         pivy=(pivy>tank->h-1)?tank->h-1:pivy;
      }
      // l'angle évolue de façon constante par pas de 0,5
      angle+=0.5;
      angle=(angle>256)?0:angle;

      // affichage sprite avec encadrement en rouge
      clear_bitmap(page);
      pivot_sprite(page,tank,x,y,pivx,pivy,ftofix(angle));
```

```
      hline(page,x-50,y,x+50,ROUGE);
      vline(page,x,y-50,y+50,ROUGE);

      //infos
      textprintf_ex(page,font,0,10,BLANC,-1,
      "Changer le pivot pivot : fleches gauche, droite, haut, bas");
      textprintf_ex(page,font,0,20,BLANC,-1,"Pivot = %d, %d",pivx,pivy);
      textprintf_ex(page,font,0,30,BLANC,-1,"x = %d, y = %d",x,y);

      // affichage buffer à l'écran
      blit(page,screen,0,0,0,0,SCREEN_W,SCREEN_H);

      // pour ralentir la vitesse d'exécution
      rest(10);
   }
   return 0;
}
END_OF_MAIN();
```

2. Sprites et animation

Un sprite est tout simplement une image bitmap dans un scénario quelconque de jeux ou de n'importe quel programme. Toutefois, à la différence des images en général, c'est une image souvent petite soit fixe et immobile, dans un décor par exemple, soit animée à l'écran comme une voiture ou un avion éventuellement conduits avec le clavier ou un joystick. Le sprite peut également apparaître comme une petite séquence d'animation composée d'un ensemble de petites images bitmaps, un bonhomme qui se déplace en marchant, un monstre etc..

2.1 Remplacer une image par une séquence d'images

Il s'agit de l'implantation dans un programme du procédé d'animation classique du type dessin animé, sous forme de petites séquences courtes nécessitant peu d'images. Ces séquences sont en général utilisées de nombreuses fois dans des circonstances éventuellement différentes du programme. Nous allons prendre pour base une séquence de bonhomme qui marche.

2.1.1 Base de l'animation : un bonhomme marche sur place

Le mouvement du bonhomme est décomposé en étapes et chaque étape du mouvement est une petite image bitmap (figure 1). Pour constituer l'animation il faut charger dans le programme la totalité des images puis les afficher les unes à la suite des autres. Le mieux est de stocker toutes les images dans un tableau, et dans l'ordre de l'animation, afin que les indices du tableau correspondent à l'ordre du déroulement de l'animation. Ensuite une variable compteur d'image pourra facilement prendre successivement les valeurs d'indices 0, 1, 2, etc. un modulo sur le nombre total d'images permettra de revenir à 0 et de mettre en boucle l'animation.

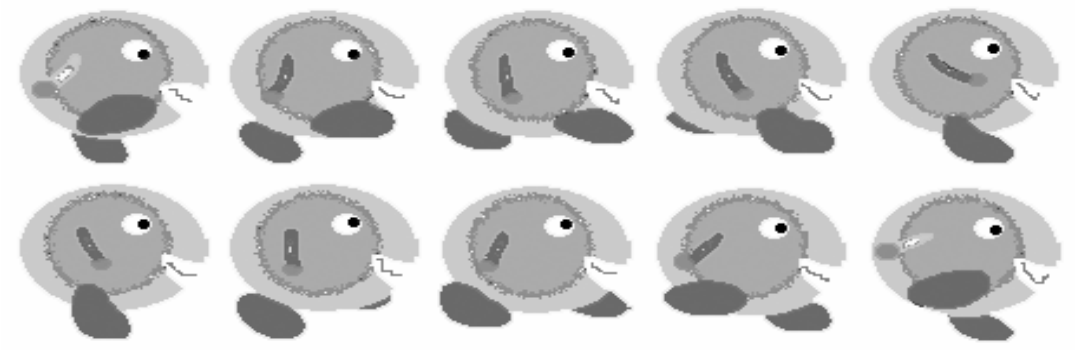


Fig 1 : les images d'une séquence d'animation

Algorithme de l'animation
Après chargement des images dans le tableau, pour voir le bonhomme simplement marcher sur place à une position fixe de l'écran l'algorithme sera :

1) Effacer le buffer .
2) Afficher l'image n_i courante dans le buffer.
3) Incrémenter n_i pour passer à l'image suivante. Si n_i égale l'indice de la dernière image, la valeur de l'indice de la première lui est affectée.
4) Afficher le buffer à l'écran.
5) Mettre le processus en boucle, retourner en 1.

Test 3.5 : un bonhomme qui marche sur place

```
#include <allegro.h>

#define ERREUR(msg){\
   set_gfx_mode(GFX_TEXT,0,0,0,0);\
   allegro_message("err %s\nligne %d\nfile %s\n",msg,__LINE__,__FILE__);\
   allegro_exit();\
   exit(EXIT_FAILURE);\
}
// nombre d'images dans l'animation
#define NB_IMAGES   10

int main (){

BITMAP* ANIM[NB_IMAGES];
BITMAP* page;
int i, posx, posy;
char buf[80];
int image_cmpt;

   allegro_init();
   install_keyboard();

   set_color_depth(16);
   if (set_gfx_mode(GFX_AUTODETECT_WINDOWED, 800, 600, 0, 0) != 0)
      ERREUR(allegro_error);

   // allocation buffer
   page = create_bitmap(SCREEN_W, SCREEN_H);
   if (!page)
      ERREUR("create page");
```

```
   // load des images de l'animation
   for(i=0; i<NB_IMAGES;i++){
      sprintf(buf,".//images//bonhomme//bonh%d.bmp",i);
      ANIM[i] = load_bitmap(buf, NULL );
      if( ! ANIM[i] )
         ERREUR("load images");
   }
```

```
   // le sprite est centré à l'écran
   posx = (SCREEN_W/2)- (ANIM[0]->w/2);
   posy = (SCREEN_H/2)- (ANIM[0]->h/2);
   // initialisation du compteur d'image sur la première à 0
   image_cmpt = 0;
```

```
   // animation en boucle
```

```
   while ( !key[KEY_ESC] ){
      // effacement du buffer
      clear_bitmap(page);

      // affichage de l'image courante à sa position
      draw_sprite(page, ANIM[image_cmpt], posx, posy);

      // incrémentation du compteur pour passer à l'image suivante; Si le
      // compteur arrive NB_IMAGE il est ramené à 0 par le modulo
      image_cmpt= ++image_cmpt%NB_IMAGES;

      // affichage à l'écran
      blit(page, screen, 0,0,0,0,SCREEN_W, SCREEN_H);
      rest(40);
   }
   // sortie
   destroy_bitmap(page);
   for( i = 0; i<NB_IMAGES; i++)
      destroy_bitmap(ANIM[i] );

   exit(EXIT_SUCCESS);
}
END_OF_MAIN();
```

2.1.2 Bonhomme déplacé au clavier

Pour qu'il y ait un déplacement du bonhomme dans le programme précédent il suffit de modifier la position de l'image, les variables posx et posy, avant affichage dans le buffer. C'est ce que nous allons faire avec un personnage type Zelda en appuyant sur les touches flèches. Nous avons huit images, deux par direction (figure 2) :

Fig 2 : quatre séquences de deux images chacune

Comme dans le programme précédent nous allons ranger les images dans un tableau et l'ordre est important : indices 0 et 1 le bonhomme monte, 2 et 3 il descend, 4 et 5 il va à gauche, 6 et 7 à droite. Les images sont appelées z0.bmp, z1.bmp, z2.bmp etc. dans le même ordre afin de rendre plus facile leur chargement dans le programme.

Lorsque l'on appuie sur les touches :
KEY_UP (flèche haut) ce sont les images des indices 0 et 1 qui sont utilisées
KEY_DOWN (bas) images des indices 2 et 3
KEY_LEFT (gauche) images des indices 4 et 5
KEY_RIGHT (droite) images des indices 6 et 7

Pour chaque touche on a besoin d'un compteur d'image qui permettra d'afficher une fois la première, une fois la seconde. De plus à chaque fois que l'on appuie sur une de ces touches la position de l'image est modifiée, ce qui donne :

1) vérifier si l'image reste dans la zone de jeu
2) si oui modifier les coordonnées en fonction de la touche appuyée (en x ou y)

3) afficher l'image courante désignée par le compteur d'image
4) incrémenter le compteur d'image (de 0 à 1 ou de 1 à 0)

Test 3.6 : un bonhomme piloté par le clavier

```
#include <allegro.h>
#include <stdio.h>

#define ERREUR(msg){\
   set_gfx_mode(GFX_TEXT,0,0,0,0);\
   allegro_message("err %s\nligne %d\nfile %s\n",msg,__LINE__,__FILE__);\
   allegro_exit();\
   exit(EXIT_FAILURE);\
}
/*********************************************************************
*********************************************************************/
int main()
{
char buf[80];
BITMAP *im[8],*page;
int i,fin,x,y,haut,bas,gauche,droite;

   allegro_init();
   install_keyboard();

   set_color_depth(16);
   if (set_gfx_mode(GFX_AUTODETECT_WINDOWED, 800, 600, 0, 0) != 0)
      ERREUR(allegro_error);

   // double buffer
   page=create_bitmap(SCREEN_W,SCREEN_H);
   if (!page)
      ERREUR("creation buffer");

   // charger les huits images du sprite qui sont dans le dossier image,
   // dans le dossier zelda à partir du répertoire du programme
   for (i=0;i<8;i++){
       sprintf(buf,".\\images\\zelda\\z%d.bmp",i);
       im[i]=load_bitmap(buf,NULL);
       if (!im[i])
          ERREUR("load image");
   }

   // position de départ au centre de l'écran, première image
   fin=haut=bas=gauche=droite=0;
   x=SCREEN_W/2;
   y=SCREEN_H/2;

   // affichage départ
   draw_sprite(page,im[7],x,y);
   textprintf_ex(page, font, 10,10, makecol(255,255,255),-1,
   "press fleches haut,bas,gauche, droite et esc pour quitter");

   // boucle events
   while (!fin){

      if (keypressed()){
         clear_bitmap(page);
         textprintf_ex(page, font, 10,10, makecol(255,255,255),-1,
         "press fleches haut,bas,gauche, droite et esc pour quitter");
```

```
         switch( readkey()>>8){ // aiguillage sur la valeur de scancode

            case KEY_UP : // images 0, 1 le bonhomme monte
               // contrôle du bord de l'écran et changement position
               if (y>0)
                  y--;

               // affichage de l'image suivante à la nouvelle position
               draw_sprite(page,im[0+haut],x,y);

               // Pour que la valeur de haut oscille en tre 0 et 1
               // on peut utiliser l'opérateur ^ : met à 0 ce qui est
               // identique et à 1 ce qui diffère, ça donne l'expression :
               haut^=1;
               // autre solution : haut=1-haut;
               break;

            case KEY_DOWN : // images 2, 3 il descend
               if (y+im[2+bas]->h<SCREEN_H)
                  y++;
               draw_sprite(page,im[2+bas],x,y);
               bas^=1;
               break;

            case KEY_LEFT : // 4, 5 il va à gauche
               if (x>0)
                  x--;
               draw_sprite(page,im[4+gauche],x,y);
               gauche^=1;
               break;

            case KEY_RIGHT : // 6,7 va à droite
               if (x+im[6+droite]->h<SCREEN_W)
                  x++;
               draw_sprite(page,im[6+droite],x,y);
               droite^=1;
               break;

            default :
               fin=1 ;
               break;
         }
      }
      blit(page,screen,0,0,0,0,SCREEN_W,SCREEN_H);
   }

   destroy_bitmap(page);
   for (i=0;i<8;i++)
       destroy_bitmap(im[i]);

   exit(EXIT_SUCCESS);
}
END_OF_MAIN();
```

2.1.3 Contrôler la vitesse de l'animation (fréquence images)

Quand on regarde le programme ci-dessus de déplacement d'un personnage avec le clavier on constate qu'il y a un décalage entre le débit de l'animation et le déplacement du personnage. L'animation est trop rapide par rapport au déplacement du personnage. Accélérer le déplacement du

personnage n'est pas réellement une solution car le personnage est alors condamné à toujours aller vite. L'idéal est de pouvoir agir sur la vitesse de l'animation indépendamment de la vitesse du déplacement du sprite. L'objectif est maintenant de découpler la vitesse de l'animation de celle de l'avancement.
Dans l'exemple suivant l'animation est composée de six images qui représentent chacune une étape de la course d'un chat (figure 3).

Fig 3 : animation du chat

Dans le programme précédent l'image de l'animation du bonhomme change à chaque fois qu'il avance. Pour découpler les deux il suffit de faire en sorte qu'il puisse avancer sans changer d'image. L'image courante reste la même plusieurs tours ; elle est en attente avant son changement. Le plus simple est de décider d'un nombre de tours à attendre puis de compter chaque tour et lorsque le nombre de tours est passé de changer l'image.

Pour ce faire, une variable « compte » compte les tours, une variable « nbTourMax» qui définit le nombre de tours maximum et une variable « courant » qui donne l'indice de l'image courante (toutes les images de l'animation étant stockées dans un tableau) le principe suivant fonctionne très bien :

```
if (compte++ > nbTourMax){
   tmps=0;
   courant=(courant+1) % 6; // courant va de 0 à 5 puis retourne à 0,
}                           // boucle sur les 6 images
```

« courant » est incrémenté de 1 uniquement lorsque « compte » atteint « nbTourMax », à chaque fois « compte » est réinitialisé à 0. Il y a ainsi possibilité d'agir sur la vitesse de l'animation en fonction d'un déplacement donné.

Test 3.7 : contrôle de la vitesse d'une animation : le chat qui traverse l'écran

```
#include <allegro.h>
#include <stdio.h>
#include <time.h>
#define ERREUR(msg){\
   set_gfx_mode(GFX_TEXT,0,0,0,0);\
   allegro_message("err %s\nligne %d\nfile %s\n",msg,__LINE__,__FILE__);\
   allegro_exit();\
   exit(EXIT_FAILURE);\
}
#define NB_IMAGES      6
/*********************************************************************
*********************************************************************/
int main()
{
char buf[80];
BITMAP *im[NB_IMAGES],*page;
int i;
int x,y;                          // position images
int imcourante,compte,nbTourMax;  // contrôle sélection images
```

```
  allegro_init();
  install_keyboard();
  srand(time(NULL));

  set_color_depth(16);
  if (set_gfx_mode(GFX_AUTODETECT_WINDOWED, 800, 600, 0, 0) != 0)
     ERREUR(allegro_error);
  // double buffer
  page=create_bitmap(SCREEN_W,SCREEN_H);
  if (!page)
     ERREUR("creation buffer");

  // charger les six images du sprite, à partir du répertoire du
  // programme, dans le dossier image dans le dossier cat dans l
  for (i=0;i<NB_IMAGES;i++){
      sprintf(buf,".\\images\\cat\\cat%d.bmp",i);
      im[i]=load_bitmap(buf,NULL);
      if (!im[i])
         ERREUR("load image");
  }
  // position de départ à gauche de l'écran
  x=0;
  y=(SCREEN_H/8)*3;
```

```
  // variables pour contrôler animation
  imcourante=0;
  compte=0;
  nbTourMax=5;
```

```
  // boucle events
  while (!key[KEY_ESC]){

     // effacer buffer
     clear_bitmap(page);
```

```
     // contrôle l'image courante
     if ( compte++ >nbTourMax){
        compte=0;
        imcourante=(imcourante+1)%NB_IMAGES;
     }
```

```
// éventuellement, pour changer le temps d'attente d'une image
     if(key[KEY_ENTER])
        nbTourMax=rand()%20;
     // avancer la position
     x+=5;
     if(x>SCREEN_W)
        x=-im[imcourante]->w;

     // afficher l'image courante dans le buffer à la bonne position
     draw_sprite(page,im[imcourante],x,y);

     // afficher le buffer à l'écran avec indication du temps
     textprintf_ex(page,font,10,30,makecol(0,255,0),-1,
     "image = %d, attente = %d, attente max  %d", imcourante, compte,
     nbTourMax);
     blit(page,screen,0,0,0,0,SCREEN_W,SCREEN_H);

     //ralentir processeur
     rest(20);
  }
```

```
   // nettoyage sortie
   destroy_bitmap(page);
   for (i=0;i<NB_IMAGES;i++)
       destroy_bitmap(im[i]);
   // sortie
   exit(EXIT_SUCCESS);
}
END_OF_MAIN();
```

2.2 Généralisation du contrôle d'un sprite

2.2.1 Définir une structure de données

Quelles sont finalement toutes les variables susceptibles de caractériser une animation sprite ?
L'exemple du chat permet de contrôler la vitesse du déroulement de l'animation indépendamment du déplacement des images dans l'espace. A la vitesse du déroulement peut être ajoutée une variable pour la direction de l'animation à l'envers (reverse) ou à l'endroit. Par ailleurs il ne faut pas oublier de stocker le nombre d'images de l'animation (6 dans l'exemple du chat).

Également le déplacement peut faire l'objet du même type de contrôle que celui de l'animation. La position peut n'être modifiée en x et en y qu'en fonction d'un paramètre d'attente, ce qui permet d'affiner considérablement la vitesse et le mouvement de l'objet. Mais afin de limiter le nombre des variables à gérer, une autre solution pour le déplacement est d'utiliser des flottants avec un pas en décimal. Comme les positions à l'écran ne peuvent être que des valeurs entières, il faut par exemple cinq fois 0,2 pour faire 1 et avancer effectivement de 1 pixel à l'écran. Si toutefois on veut avancer l'image par exemple par sauts de 5 pixels tous les n tours il faudra se rabattre sur la première solution. Dans l'exemple qui suit nous avons opté pour le contrôle de la position et du déplacement en flottants. La structure de données résultante est la suivante :

```
typedef struct SPRITE{
  // le deplacement
    float x,y;          // position
    float px,py;        // pas du deplacement
    int tx,ty;          // taille
  // l'image
    int imcourante;     // l'image courante
    int nb_image;       // le nombre max des images de l'animation
    int tmps;           // pour compter le temps d'affichage de l'image
    int maxtmps;        // temps maxi pour chaque image
    int dir;            // sens de l'animation (à reculons ou avancer)
}t_sprite ;
```

Dans le programme de test n°3.8 présenté plus bas, l'animation comporte 32 images et toutes les images sont regroupées sur un seul fichier. Les 32 sprites de taille identique y sont disposés à intervals réguliers sur 8 colonnes. Avant de présenter le programme complet, voici comment récupérer les images et les stocker dans un tableau.

2.2.2 Récupérer une série d'images stockées dans un seul fichier

Dans un même programme nous pouvons avoir plusieurs séquences d'animation et certaines peuvent être simultanées, en parallèle. Le nombre des fichiers images nécessaires au programme peut augmenter considérablement et il sera préférable de les regrouper toutes dans un seul fichier.

Le nombre d'images pour une séquence d'animation est très variable, en général c'est peu, entre 2 et une dizaine, parfois davantage comme dans l'exemple ci-dessous où il y a 32 images pour l'animation d'un météorite. Mais il peut y avoir plusieurs séquences d'animation dans le programme, dans ce cas très rapidement le répertoire du programme est encombré. Pour l'éviter il suffit de regrouper toutes les

images sur une seule image et d'opérer la découpe dans le programme. Comme dans les exemples du bonhomme, du personnage Zelda et du chat montrés plus haut, chaque séquence d'animation sera rangée en lignes et colonnes dans le fichier image de regroupement. Par exemple soit un fichier nommé « sequences.bmp » il aura par exemple une allure du type de la figure 4.

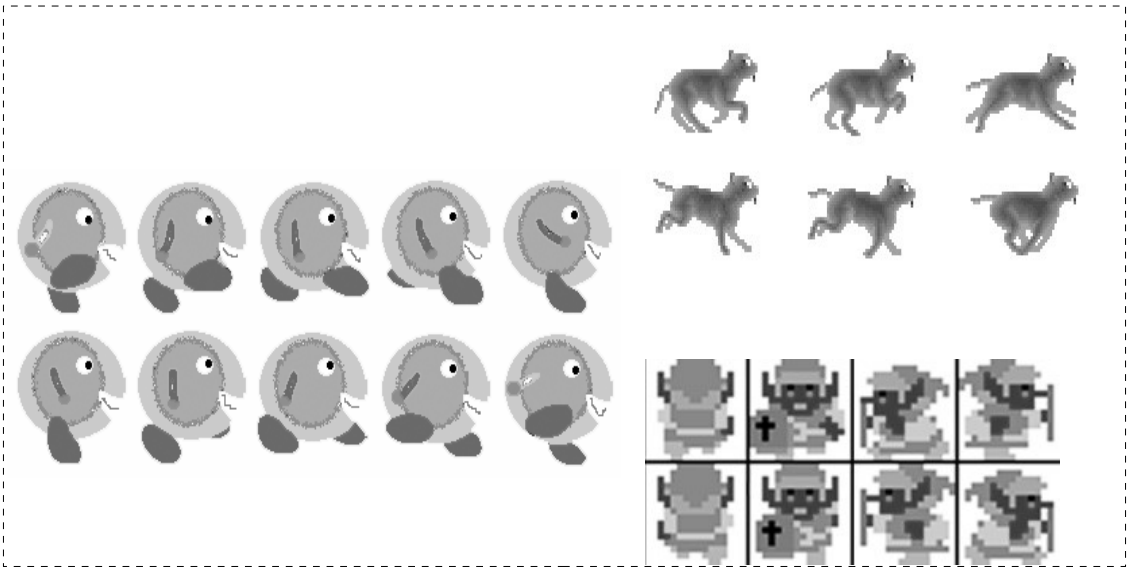


Fig 4 : plusieurs séquences d'animation réunies sur un même fichier

Première chose, il faut le charger normalement dans le programme avec palette s'il est en 8 bits et NULL sinon :

```
BITMAP*groupe_animes ;
            groupe_animes=load_bitmap(« sequences.bmp »,NULL) ;
```

Ensuite voici une fonction de découpe qui permet de récupérer chaque image des animations et de les ranger dans un ou plusieurs tableaux dans le programme :

```
(1)BITMAP* recup_sprites( BITMAP*scr, // bitmap origine
                          int w,      // taille de l'image élément d'une
                          int h,      // animation
                          int startx, // début de la série à partir
                          int starty, //    de la position x et y
                          int col,    // nombre de colonnes
                          int element)// numéro de l'image dans la série
    {
    BITMAP *bmp;
    int x,y;

 (2)    bmp=create_bitmap(w,h);
        if (bmp!=NULL){
 (3)       x= startx+(element%col)*w;
           y= starty+(element/col)*h;
 (4)       blit(scr,bmp,x,y,0,0,w,h);
        }
        return bmp;
    }
```

(1) La fonction « recup_sprites » prend en argument la bitmap d'origine avec les indications nécessaires pour retrouver chaque élément image du fichier :
- « w » et « h » donnent la taille d'une image d'une animation. En général toutes les images d'une animation ont la même taille pour faciliter ce découpage image par image.
- « startx » et « starty » donnent la position dans le fichier de la première image de l'animation étant

sous entendu que toutes les images de chaque séquence sont correctement alignées en lignes et colonnes.
- « col » est le nombre de colonnes selon lequel les images de la séquence sont rangées
- « element » est le numéro d'ordre de l'image que l'on souhaite récupérer. Par exemple, si l'on prend la séquence du bonhomme, l'image 8 de l'animation est celle de la troisième colonne de la seconde ligne.

(2) La fonction renvoie un pointeur BITMAP* qui désigne l'image dans une séquence désignée par « element ». Le premier point est d'allouer la mémoire requise par cette bitmap compte tenu de sa taille (w,h)

(3) En cas de succès de l'allocation pour la bitmap, les coordonnées (x,y) de l'image élément souhaitée sont données du fait du point de départ de la séquence (startx,starty), du numéro de l'élément, du nombre de colonnes et des taille en largeur et hauteur. Le numéro de l'élément modulo le nombre de colonnes donne la position de l'élément sur le plan horizontale et la division de ce même numéro par le nombre de colonne donne sa position sur le plan vertical. Il suffit de multiplier par la longueur pour x et la hauteur pour y puis d'ajouter la position de départ (startx,starty) afin d'obtenir la position en pixels de l'élément.

(4) Ensuite l'élément est recopié avec la fonction blit() à partir de la position (x,y) et selon la taille de l'élément dans l'image source sur la bitmap de destimation « bmp » allouée à cet effet. Ensuite l'adresse de l'image élément recopiée est renvoyée.

Test 3.8 : généralisation du contrôle d'une séquence sprite : une balle rebondit et tourne sur elle-même

```
#include <allegro.h>
#include <stdio.h>
#include <time.h>

#define ERREUR(msg){\
   set_gfx_mode(GFX_TEXT,0,0,0,0);\
   allegro_message("err %s\nligne %d\nfile %s\n",msg,__LINE__,__FILE__);\
   allegro_exit();\
   exit(EXIT_FAILURE);\
}
// le nombre total des images de l'animation
#define NB_IMAGE   32

// généralisation controle du sprite : structure de données
typedef struct SPRITE{
    // le deplacement
    float x,y;          // position et
    float px,py;        // pas du deplacement en flottants

    // paramètres pour l'image
    int tx,ty;          // taille

    // paramètres de l'animation
    int imcourante;     // l'image courante
    int nb_image;       // le nombre max des images de l'animation
    int tmps;           // pour compter le temps d'affichage de l'image
    int maxtmps;        // temps maxi pour chaque image
    int dir;            // sens de l'animation (à reculons ou avancer)
}t_sprite ;

void      cntl_anim       (t_sprite *s);
void      avance          (t_sprite *s);
BITMAP*   recup_sprites   (BITMAP*scr, int w, int h, int startx,
                           int starty, int col, int i);
```

```
/*********************************************************************
*********************************************************************/
int main()
{
BITMAP *page,*im[NB_IMAGE],*tmp;
int i;
t_sprite balle;     // la bitmap est manipulée avec un pointeur dans ce
t_sprite *s=&balle; // programme et ceci évite de faire un malloc

   allegro_init();
   install_keyboard();
   srand(time(NULL));

   set_color_depth(16);
   if (set_gfx_mode(GFX_AUTODETECT_WINDOWED, 800, 600, 0, 0) != 0)
      ERREUR(allegro_error);
  ------------------------------------------------------------
  // récup de l'image qui contient tout
  tmp=load_bitmap(".\\images\\all_sprites.bmp",NULL);

  if (!tmp)
     ERREUR("creation image temporaire image");

  // découpage dans tableau im[]
  for (i=0;i<NB_IMAGE;i++){
    im[i]=recup_sprites(tmp,64,64,0,0,8,i);
    if (!im[i])
       ERREUR("recup images");
  }
  destroy_bitmap(tmp);
  ------------------------------------------------------------
   page=create_bitmap(SCREEN_W,SCREEN_H);

   if (!page)
      ERREUR("creation buffer")
  ------------------------------------------------------
  // initialisation du sprite :
  //position
  s->x=rand()%(SCREEN_W-im[0]->w);
  s->y=rand()%(SCREEN_H-im[0]->h);
  // déplacement
  s->px=(((float)rand())/RAND_MAX)*3;
  s->py=(((float)rand())/RAND_MAX)*3;
  // taille (pas de variation
  s->tx=im[0]->w;
  s->ty=im[0]->h;
  // succession des images
  s->imcourante=0;
  s->nb_image=NB_IMAGE-1;
  s->tmps=0;
  s->maxtmps=rand()%3+1;
  // direction (avant-reculons)
  s->dir=1;
  ------------------------------------------------------
   while (!key[KEY_ESC]){  // boucle événements

      clear_bitmap(page);
    ------------------------------------------------------------------
    // mise à jour du sprite, animation image et position
    cntl_anim(s);  // anime image
    avance(s);     // position
    ------------------------------------------------------------------
```

```
      // infos
      textprintf_ex(page,font,10,10,makecol(0,255,0),-1,
      "x = %3d, y = %3d", (int)s->x,(int)s->y);
      textprintf_ex(page,font,10,20,makecol(0,255,0),-1,
      "pas de deplacement en x = %.2f, en y = %.2f",s->px,s->py);
      textprintf_ex(page,font,10,30,makecol(0,255,0),-1,
      "retard image = %2d,retard max =%2d, direction = %2d",
      s->tmps,s->maxtmps,s->dir);

      // image
      draw_sprite(page,im[s->imcourante],s->x,s->y);

      //affichage ecran
      blit(page,screen,0,0,0,0,SCREEN_W,SCREEN_H);

      // ralentir process
      rest(1);
   }
   // sortie propre
   destroy_bitmap(page);
   for (i=0;i<NB_IMAGE;i++)
      destroy_bitmap(im[i]);
   return 0;
}
END_OF_MAIN();
/*********************************************************************
récupérer sur un fichier la séquence d'animation
*********************************************************************/
BITMAP* recup_sprites(  BITMAP*scr,             // bitmap origine
                        int w, int h,           // taille element
                        int startx, int starty, // à partir de
                        int col,                // nombre de colones
                        int i)                  // élément ieme
{
BITMAP *bmp;
int x,y;

   bmp=create_bitmap(w,h);
   if (bmp!=NULL){
      x= startx+(i%col)*w;
      y= starty+(i/col)*h;
      blit(scr,bmp,x,y,0,0,w,h);
   }
   return bmp;
}
/*********************************************************************
contrôler l'animation des images (qui peut fonctionner à l'envers)
*********************************************************************/
void cntl_anim(t_sprite *s)
{
   // attention avec cette formule maxtmps ne doit jamais valoir 0
   s->tmps=(++s->tmps)%s->maxtmps;
   if (!s->tmps){
      s->imcourante+=s->dir; // passer à l'image suivante en fonction de la
                             // direction de l'animation

      // contrôler que l'on reste dans le tableau des images
      // si on va à l'envers
      s->imcourante=(s->imcourante<0) ? s->nb_image : s->imcourante;
      // si sens normal
      s->imcourante=(s->imcourante>s->nb_image) ? 0 : s->imcourante;
```

```
      // à titre indicatif, une autre écriture plus compacte qui marche
      // pour les deux sens :
      //s->imcourante=((s->imcourante+s->dir)+s->nb_image)%s->nb_image;
   }
}
/*************************************************************************
contrôle du déplacement dans l'écran
*************************************************************************/
void avance(t_sprite *s)
{
int res=0;

   // 1 AVANCER
   s->x+=s->px;
   s->y+=s->py;

   // 2 CONTROLE DES BORDS

   // ATTENTION : j'ai utilisé une affectation du résultat de chaque test
   // dans un variable res pour savoir à la fin s'il y a eu un bord touché
   // et si oui faire des modifications

   if (res=(s->x<0)){  // horizontal
      s->x=0;
      s->px=(((float)rand())/RAND_MAX)*5;
   }

   else if (res=(s->x+s->tx > SCREEN_W)){
      s->x=SCREEN_W-s->tx;
      s->px=(((float)rand())/RAND_MAX)*-5;
   }

   if (res=(s->y<0)){ // vertical
      s->y=0;
      s->py=(((float)rand())/RAND_MAX)*5;
   }
   else if (res=(s->y+s->ty>SCREEN_H)){
      s->y=SCREEN_H-s->ty;
      s->py=(((float)rand())/RAND_MAX)*-5;
   }

   if (res){  // si bord touché
      // change direction image (reverse)
      s->dir= 2*(rand()%2)-1;
      // le retard image
      s->tmps=0;
      s->maxtmps=rand()%3+1;
   }
}
```

2.3 Animer plusieurs séquences simultanément

L'objectif est de faire fonctionner plusieurs séquences d'animation en parallèle et nous allons mettre en scène un jardin au bord d'une rivière. Dans les airs passe un dragon, une abeille et un moustique, dans l'eau un poisson et sur terre un crabe et un serpent. La structure de données pour un personnage est identique à celle du programme précédent sauf que pour le déplacement, nous avons utilisé le même mécanisme de retard que celui de l'image. Comme chaque personnage doit disposer de sa propre séquence d'animation nous avons doté la structure d'un champ supplémentaire qui est

un tableau de pointeurs bitmap. Ce tableau est dynamique parce que les animations n'ont pas toutes le même nombre d'images.

```
typedef struct SPRITE{

    // le deplacement (sans float)
    int x,y;         // position
    int px,py;       // pas du déplacement
    int wx,wy;       // pour retarder avancement en x et y
    int xcmpt,ycmpt; // pour compter le retard

    // l'image
    int tx,ty;       // taille
    int imcourante;  // l'image courante
    int nb_image;    // le nombre max des images de l'animation
    int maxtmps;     // temps maxi pour chaque image
    int tmps;        // pour compter le temps d'affichage de l'image
    int dir;         // sens de l'animation (à reculons ou avancer)
    BITMAP**anim;    // tableau dynamique de pointeurs pour stocker l'anim

}t_sprite ;
```

Ensuite, nous avons besoin d'un tableau pour stocker tous les personnages. Comme il est nécessaire d'accéder à chaque personnage individuellement , nous allons identifier chaque indices du tableau par une constante grâce à un enum (comme dans le test 3.1) :

```
enum{DRAGON,POISSON,CRABE,ABEILLE,MOUSTIQUE,SERPENT,NB_SPRITE};
```

Il s'en suivra que le dragon sera à l'indice 0, le poisson à l'indice 1, le crabe à 2, l'abeille à 3, le moustique à 4, le serpent à 5 pour un total de 6 personnages. Pour le tableau nous optons pour un tableau de pointeurs :

```
t_sprite* tab[NB_SPRITE] ;
```

Dans le programme ci-dessous, après avoir récupérer l'image du décor, le premier point est d'initialiser le tableau des personnages. C'est le rôle de la fonction construct_sprite(). Cette fonction, pour chaque personnage, alloue la mémoire requise puis fait appel à deux sous fonctions. La fonction init_position() pour initialiser les champs de positions, de taille, de déplacement et d'attente de chaque personnage et la fonction init_image() pour initialiser les champs de l'animation, notamment allouer le tableau dynamique de bitmap* et récupérer les images de l'animation spécifique du personnage. Pour le reste il n'y a pas de nouveauté.

Test 3.9 : Animer plusieurs personnages simultanément, le jardin

```
#include <allegro.h>
#include <stdio.h>
#include <time.h>

#define ERREUR(msg){\
   set_gfx_mode(GFX_TEXT,0,0,0,0);\
   allegro_message("err %s\nligne %d\nfile %s\n",msg,__LINE__,__FILE__);\
   allegro_exit();\
   exit(EXIT_FAILURE);\
}
// structure de données
typedef struct SPRITE{

    // le deplacement (sans float mais avec le même mécanisme de retard que
    // celui utilisé pour pour l'image)
    int x,y;         // position
```

```
    int px,py;       // pas du deplacement
    int tx,ty;       // taille
    int wx,wy;       // pour retarder avancement en x et y
    int xcmpt,ycmpt; // pour compter le retard

    // l'image
    int imcourante;  // l'image courante
    int nb_image;    // le nombre max des images de l'animation
    int maxtmps;     // temps maxi pour chaque image
    int tmps;        // pour compter le temps d'affichage de l'image
    int dir;         // sens de l'animation (à reculons ou avancer)
    BITMAP**anim;    // un tableau de pointeurs pour stocker chaque petite
anim

}t_sprite ;

// la série des indices des sprites tous regroupés dans un tableau
// de t_sprite* déclaré en local dans le main():
enum{DRAGON,POISSON,CRABE,ABEILLE,MOUSTIQUE,SERPENT,NB_SPRITE};

void      construct_sprites   (t_sprite*tab[]);
void      init_position       (t_sprite*s,int posx, int posy, int tx,
                               int ty,int px, int py,int wx, int wy);
void      init_image          (t_sprite*s,int nbimage, int maxtmps,int dir,
                               char*name, int nb_col);
void      recup_images_anime  (t_sprite*s,char*fname,int nb_col);
void      destruct            (t_sprite*tab[]);
void      avance_sprite       (BITMAP*dest,t_sprite *s);
/*************************************************************************
*************************************************************************/
int main()
{
t_sprite* all[NB_SPRITE];
BITMAP *page,*decor;
int i,start;

   allegro_init();
   install_keyboard();
   srand(time(NULL));

   set_color_depth(16);
   if (set_gfx_mode(GFX_AUTODETECT_FULLSCREEN, 640, 480, 0, 0) != 0)
      ERREUR(allegro_error);

   // le buffer et l'image du fond.
   // Toutes les images sont dans le dossier dragon
   page=create_bitmap(SCREEN_W,SCREEN_H);
   decor=load_bitmap(".\\images\\dragon\\decor.bmp",NULL);
   if (!page||!decor)
      ERREUR("create page||load decor");
  ------------------------------------
  | // construire tous les spites    |
  | construct_sprites(all);          |
  ------------------------------------
   // boucle
   start=clock();
   while (!key[KEY_ESC]){

      // ralentir sans bloquer le programme. On entre dans le bloc toutes
      // les 10 millisecondes, mais si l'on entre pas les instructions qui
      // suivent le bloc peuvent être exécutées (en l'occurrence y en a
```

```
      // pas )
      if (clock()>start+10){
         start=clock();

         // efface page avec décor
         blit(decor,page,0,0,0,0,decor->w,decor->h);

         // animation des sprites
         for (i=0; i<NB_SPRITE;i++)
            avance_sprite(page,all[i]);

         // affichage écran
         blit(page,screen,0,0,0,0,SCREEN_W,SCREEN_H);
      }
   }
   destroy_bitmap(page);
   destruct(all);
   return 0;
}
END_OF_MAIN();


/*********************************************************************
Allouer chaque sprite et mettre tout à 0
*********************************************************************/
void construct_sprites(t_sprite*tab[])
{
int i;
    // allouer la mémoire pour chaque personnage
    for (i=0; i<NB_SPRITE; i++){
       tab[i]=(t_sprite*)malloc(sizeof(t_sprite));
       memset( tab[i],0,sizeof(t_sprite));
    }
    // 1 LE DRAGON
    // paramêtres position =le sprite, position,taille,deplacement, attente
    init_position(tab[DRAGON],500,0,128,64,-5,0,1,0);
    //paramêtres image = nb_image, maxtmps,dir, nomfichier, nb colonnes
    init_image(tab[DRAGON],6,5,1,".\\images\\dragon\\dragon.bmp",3);

    // 2 LE POISSON
    init_position(tab[POISSON],300,400,64,32,3,0,1,0);
    init_image(tab[POISSON],3,8,1,".\\images\\dragon\\poisson.bmp",3);

    // LE CRABE
    init_position(tab[CRABE],300,212,64,32,2,0,6,0);
    init_image(tab[CRABE],4,20,1,".\\images\\dragon\\crabe.bmp",4);

    // L'ABEILLE
    init_position(tab[ABEILLE],100,122,50,40,-3,0,1,0);
    init_image(tab[ABEILLE],6,8,1,".\\images\\dragon\\abeille.bmp",6);

    // LE MOUSTIQUE
    init_position(tab[MOUSTIQUE],500,70,50,40,4,0,1,0);
    init_image(tab[MOUSTIQUE],6,2,1,".\\images\\dragon\\moustique.bmp",6);

    // LE SERPENT
    init_position(tab[SERPENT],350,200,100,50,-2,0,1,0);
    init_image(tab[SERPENT],7,4,1,".\\images\\dragon\\serpent.bmp",4);
}
/*********************************************************************
*********************************************************************/
```

```
void init_position(t_sprite*s,int posx,int posy,  // position
                               int tx, int ty,     // taille
                               int px, int py,     // deplacement
                               int wx, int wy)     // attente deplacement
{
   s->x=posx;  // pour tous les champs, une simple affectation
   s->y=posy;  // des valeurs passées
   s->tx=tx;
   s->ty=ty;
   s->px=px;
   s->py=py;
   s->wx=wx;
   s->wy=wy;
}
/*****************************************************************************
*****************************************************************************/
void init_image( t_sprite*s,                          // le sprite
                 int nbimage, int maxtmps,int dir, // param anime
                 char*name, int nb_col)        // pour récup sur fichier
{
int i;
   s->maxtmps=maxtmps;
   s->dir=dir;
   s->nb_image=nbimage;

   // construire le tableau des images (tableau de pointeurs)
   s->anim=(BITMAP**)malloc(sizeof(BITMAP*)*nbimage);
   // allouer chaque pointeur BITMAP* du tableau
   for (i=0;i<nbimage;i++){
      s->anim[i]=create_bitmap(s->tx,s->ty);
      if(!s->anim[i])
         ERREUR("creation sprite");
   }
   // récupérer
   recup_images_anime(s,name,nb_col);
}
/*****************************************************************************
*****************************************************************************/
void recup_images_anime(t_sprite*s,char*fname,int nb_col)
{
BITMAP*tmp=NULL;
int x,y,i;
   tmp=load_bitmap(fname,NULL);
   if (!tmp)
      ERREUR("recup anime dragon");

   for (i=0;i<s->nb_image;i++){
      x= (i%nb_col) * s->tx;
      y= (i/nb_col) * s->ty;
      blit(tmp,s->anim[i],x,y,0,0,s->tx,s->ty);
   }
   destroy_bitmap(tmp);
}
/*****************************************************************************
*****************************************************************************/
void destruct(t_sprite*tab[])
{
int i,j;
   for (i=0; i<NB_SPRITE; i++)
      for (j=0; j<tab[i]->nb_image; j++)
         destroy_bitmap(tab[i]->anim[j]);
```

```
      free(tab[i]->anim);
      free(tab[i]);
}
/*****************************************************************************
*****************************************************************************/
void avance_sprite(BITMAP*dest,t_sprite *s)
{
    // MODIFIER POSITION

// attention à une possible division par 0 si wx,wy ou maxtmps sont à
// 0 avec ce style de code :
//   s->xcmpt=(++s->xcmpt)%s->wx;
//   if (!s->xcmpt)
//      s->x+=s->px;
// on peut lui préférer celui-ci :

   // petit compte pour retard avance en x
   if (++s->xcmpt > s->wx){
      s->x+=s->px;
      s->xcmpt=0;
   }
   // idem en y

   if (++s->ycmpt>s->wy){
      s->y+=s->py;
      s->ycmpt=0;
   }

   // CONTROLE DES BORDS
   // controle mouvement x
   if (s->x+s->tx<0)
      s->x=SCREEN_W-1;

   if (s->x>SCREEN_W)
      s->x=-s->tx;

   // controle mouvement y
   if (s->y+s->ty<0)
      s->y=SCREEN_H-1;

   if (s->y>SCREEN_H)
      s->y=-s->ty;

   // CONTROLE IMAGE
   // compte pour affichage image
   if ( ++s->tmps > s->maxtmps){
      // si inférieur à 0 on repart à l dernière et si supérieur au
      // nombre d'image on repart à 0;
      s->imcourante= ((s->imcourante+s->dir)+s->nb_image)%s->nb_image;
      s->tmps=0;
   }

   // AFFICHAGE SPRITE
   draw_sprite(dest,s->anim[s->imcourante],s->x,s->y);
}
```

3. Détecter des collisions

Repérer les collisions permet de savoir si une image rencontre une autre image et c'est essentiel dans le domaine des sprites : un missile atteint sa cible, une voiture reste sur la route, un bonhomme tape sur un mur, attrape quelque chose etc.

Identifier la couleur du fond est une première approche qui permet de détecter quelque chose sur le fond du fait d'une couleur différente. Il est également facile de savoir si un point est dans un rectangle ou s'il est dans un triangle, et à partir de plusieurs points sélectionnés judicieusement sur une image nous pouvons établir une collision avec une autre image. Mais comme les images sont toujours sous forme de rectangles souvent il s'agit de regarder s'il y a ou non une intersection de rectangles.

3.1 Recherche d'une couleur sur un fond

Cette option peut servir en différentes circonstances.
Imaginons un jeu où la souris soit un viseur et le clic un tir. Le tir peut être caractérisé par un cercle de couleur à l'écran autour de la position de la souris, juste avant l'affichage du double buffer. Dans ce cas savoir si un objet a été touché par le tir revient à faire un test sur la couleur de l'écran à la position de cet objet. Si le résultat est la couleur du cercle, l'objet est dans le cercle et a été touché. Le double buffer réaffiché ensuite efface ce cercle.

Cette méthode pour repérer des interactions a l'avantage d'être immédiate. Il y a un test à faire à partir d'une position pour repérer dans une bitmap de référence s'il y a quelque chose ou non. Cette bitmap devra être rudimentaire, par exemple noir si rien et blanc aux endroits où il y a quelque chose. Lorsqu'un personnage trouve du blanc il est sur quelque chose. De plus, pour savoir si un personnage touche quelque chose, il sera probablement nécessaire de tester plusieurs pixels de l'image du personnage. Pour les autres méthodes il faut en général passer en revue tous les objets présents et voir, pour un objet en particulier, s'il est en interaction avec un autres à partir de la position et de la taille. Lorsque le nombre d'objets est important ça peut devenir lent. Du point de vue du codage, aucune difficulté particulière il suffit d'appeler la fonction getpixel() qui pour une position donnée dans une bitmap donne la couleur du pixel.

Cette méthode peut être utilisée pour gérer un personnage dans un décor. Si l'image du décor comprend de nombreuses couleurs avec des motifs complexes, il est néanmoins possible de doubler cette bitmap par une bitmap de même taille mais simplifiée où apparaîtront les formes importantes avec des couleurs unies. Plusieurs bitmap de ce type peuvent être créées et une peut être associée aux mouvements des personnages. Chaque personnage y laisse comme une empreinte, une trace identificatrice là où il se trouve et peut à partir de sa position repérer l'empreinte des autres.

3.2 Un point dans un rectangle

Dans différentes circonstances nous pouvons avoir besoin de savoir si un point est dans une zone rectangulaire, pour identifier une zone de clic souris par exemple. Un point est dans un rectangle si sa position horizontale est à l'intérieure de la position plus la taille horizontale du rectangle, et, si sa position verticale est à l'intérieur de la position verticale plus la taille verticale du rectangle. Il y a un test à faire pour montrer soit qu'il est dedans soit qu'il est dehors et la réponse est retournée. Le mieux est d' écrire une fonction qui a comme paramètres la position (x,y) à tester, le coin haut-gauche et le coin bas-droite du rectangle. La fonction retourne 1 si le point est dans le rectangle et 0 sinon.

```
int  is_in_rect(int x, int y, int top, int left, int right, int bottom)
{
int res=0;
   if ( x > left && x < right && y > top && y < bottom)
      res=1;
   return res;
}
```

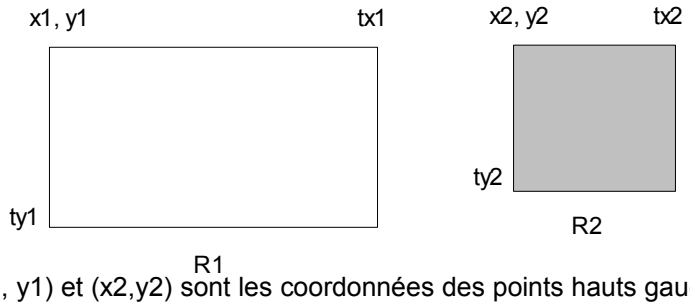
Dans cette fonction, nous posons que le point est en dehors (res=0) et il s'agit de prouver que le point est dedans. A l'inverse nous pouvons poser que le point est dedans (int res=1) et le test prouve

éventuellement que c'est faux dés que x ou y se trouvent à l'extérieur, ce qui donne :

```
int  is_in_rect(int x, int y, int top, int left, int right, int bottom)
{
int res=1;
   if ( x < left || x > right || y < top || y > bottom)
      res=0;
   return res;
}
```

3.3 Intersection de rectangle

Autre option pour la détection de collision, l'intersection de rectangles. Soit deux rectangles R1 et R2 :



(x1, y1) et (x2,y2) sont les coordonnées des points hauts gauche de chaque rectangle, tx1, ty1 et tx2, ty2 correspondent aux tailles respectives des deux rectangles, les coordonnées des points bas droite sont respectivement (x1+tx1, y1+ty1) et (x2+tx2, y2+ty2). Il y a plusieurs façons de s'y prendre pour savoir s'il y a collision entre R1 et R2.
Nous pouvons chercher à savoir si R1 est totalement à gauche ou à droite ou au dessus ou en dessous de R2 soit le test suivant :

Si (x1+tx1 < x2 OU x1 > x2+tx2 OU y1+ty1 < y2 OU y1 < y2+ty2)
  Pas de collision
Sinon
  Collision

Il suffit qu'une des expressions soit vraie pour que le test soit vrai et les quatre expressions ne seront vérifiées qu'en cas de collision entre R1 et R2.

Nous pouvons regarder également de la façon suivante :

Si (x1+tx1 > x2 ET x1 < x2+tx2 ET y1+ty1 > y2 ET y1 < y2+ty2)
  Collision
Sinon
  Pas de collision

Le programme suivant met en œuvre ces tests à partir d'un rectangle aux coordonnées fixes et d'un rectangle mobile avec la souris. Lorsqu'il y a interaction le rectangle mobile change de couleur.

Test 3.10 : intersection de rectangles

```
#include <allegro.h>
#include <stdio.h>
#include <time.h>

#define ERREUR(msg){\
   set_gfx_mode(GFX_TEXT,0,0,0,0);\
   allegro_message("err %s\nligne %d\nfile %s\n",msg,__LINE__,__FILE__);\
   allegro_exit();\
   exit(EXIT_FAILURE);\
```

```
}
// pour informations des deux rectangles
typedef struct FORME{
   int x,y;        //position
   int tx,ty;      //taille
   int couleur;    //couleur
}FORME;
/*********************************************************************
*********************************************************************/
int main()
{
BITMAP*page;
FORME f1,f2;
int color;

   allegro_init();
   install_keyboard();
   install_mouse();
   srand(time(NULL));

   if (set_gfx_mode(GFX_AUTODETECT_WINDOWED, 800, 600, 0, 0) != 0)
      ERREUR("gfx_mode");

   // buffer
   page=create_bitmap(SCREEN_W,SCREEN_H);
   if (!page)
      ERREUR("create_bitmap page");

   // une forme fixe
   f2.tx=rand()%200+100;
   f2.ty=rand()%200+100;
   f2.x=(SCREEN_W-f2.tx)/2;
   f2.y=(SCREEN_H-f2.ty)/2;
   //une forme mobile, sa taille (prend la position de la souris)
   f1.tx=40;
   f1.ty=40;

   // boucle event
   while (!key[KEY_ESC]){
      // si la position de la forme mobile est différente de celle de la
      // souris
      if (f1.x!=mouse_x || f1.y!=mouse_y){

         // prendre position de la souris
         f1.x=mouse_x;
         f1.y=mouse_y;

         // effacer buffer
         clear_bitmap(page);

         // si collision afficher forme mobile en vert, sinon en rouge
         if (collision_rect(&f1, &f2))
            color=makecol(0,255,0);
         else
            color=makecol(0,0,255);

         // form fixe
         rectfill(page,f2.x,f2.y,f2.x+f2.tx,f2.y+f2.ty,makecol(255,0,0));
         // forme mobile
         rectfill(page,f1.x,f1.y,f1.x+f1.tx,f1.y+f1.ty,color);
```

```
         // affichage du tout à l'écran
         blit(page,screen,0,0,0,0,page->w,page->h);
      }
   }
   // sortie
   destroy_bitmap(page);
   return 0;
}
END_OF_MAIN();
/*****************************************************************
Collision de deux rectangles
*****************************************************************/
int collision_rect(FORME *f1, FORME *f2)
{
int res=1;   // par défaut collision

   if ( (f1->x > f2->x + f2->tx) ||   // f1 à droite de f2
        (f1->y > f2->y + f2->ty) ||   // f1 en dessous
        (f2->x > f1->x + f1->tx) ||   // f2 à droite de f1
        (f2->y > f1->y + f1->ty)   ){ // f2 en dessous

                res=0;         // preuve pas de collision
   }
   return res;
}
/*****************************
Nous pouvons écrire également :

int collision_rect(FORME *f1, FORME *f2)
{
int res=0;   // par défaut pas collision

   if ( (f1->x < f2->x + f2->tx) &&
        (f1->x + f1->tx > f2->x) &&
        (f1->y < f2->y + f2->ty) &&
        (f1->y + f1->ty > f2->y)   ){

                res=1;         // preuve collision
   }
   return res;
}
*/
```

3.4 Un point dans un triangle

Pour connaître si un point est dans un triangle, nous utilisons la règle du déterminant. Elle permet de savoir si un point est à gauche ou à droite d'un vecteur. La règle du déterminant est la suivante : soit 3 points PO, PI et P, P est à gauche de POPI si le déterminant vxi*vyp-vxp*vyi des vecteurs POPI(vxi,vyi) et POP(vxp,vyp) est positif, et à droite dans le cas contraire.
Ensuite, soit le triangle ABC si le point P est à gauche de AB, de BC et de CA alors il est dans le triangle. Le triangle est considéré dans le sens inverse des aiguilles d'une montre (triangle direct).

Pour ce faire nous procédons avec deux fonctions. Une, la fonction gauche() pour savoir si un point est à gauche d'un segment. Cette fonction a comme paramètre la position à tester, et les deux points nécessaires au segment. L'autre, la fonction in_triangle(), prend la position à tester et les trois points du triangle à tester. Le test ci-dessous permet de générer des triangles aléatoires en appuyant sur la touche enter. Si la souris se trouve dans le triangle affiché le triangle change de couleur.

Test 3.11 : un point dans un triangle

```
#include <allegro.h>
#include <stdio.h>
#include <time.h>

#define ERREUR(msg) {\
   set_gfx_mode(GFX_TEXT,0,0,0,0);\
   allegro_message("Error : \n%s\n", msg);\
   allegro_exit();\
   return 1;\
}

int      gauche         (int px,int py,int ox,int oy,int ix,int iy);
int      in_triangle    (int x, int y, int ax,int ay,int bx,int by,int
                         cx,int cy);
/*********************************************************************
*********************************************************************/
int main()
{
int sommet[10];
int res,x1,y1,x2,y2,x3,y3,color;
BITMAP*page;
   allegro_init();
   install_keyboard();
   install_mouse();
   srand(time(NULL));

   if (set_gfx_mode(GFX_AUTODETECT_WINDOWED, 800, 600, 0, 0) != 0)
      ERREUR("gfx_mode");

   // buffer
   page=create_bitmap(SCREEN_W,SCREEN_H);
   if (!page)
      ERREUR("create_bitmap page");
   // pour voir le curseur souris à l'écran
   show_mouse(screen);

   // boucle event
   while (!key[KEY_ESC]){

       // génération d'un triangle aléatoire
       if ( key[KEY_ENTER]){
          clear_bitmap(page);
          x1=rand()%SCREEN_W/2;
          y1=SCREEN_H/2+rand()%SCREEN_H/2;
          x2=SCREEN_W/2+rand()%SCREEN_W/2;
          y2=SCREEN_H/2+rand()%SCREEN_H/2;;
          x3=rand()%SCREEN_W;
          y3=rand()%SCREEN_H/2;
          color=makecol(rand()%255, rand()%255,128);
          triangle(page,x1,y1,x2,y2,x3,y3,color);
       }

       // position souris dans le triangle ?
       if (in_triangle(mouse_x, mouse_y, x1,y1,x2,y2,x3,y3)){
         // si oui changement de couleur du triangle
         clear_bitmap(page);
         color=makecol(rand()%255, rand()%255,0);
         triangle(page,x1,y1,x2,y2,x3,y3,color);
       }
```

```
        blit(page,screen,0,0,0,0,SCREEN_W,SCREEN_H);

    }

    // sortie
    destroy_bitmap(page);
    return 0;
}
END_OF_MAIN();
/*********************************************************************
fonction principale qui permet de savoir si un point est à gauche d'un
vecteur.Nous utilisons la règle du déterminant : soit trois points po, pi
et p, p est à gauche de popi si le déterminant vxi*vyp-vxp*vyi des vecteurs
popi(vxi,vyi) et pop(vxp,vyp) est positif et à droite sinon.
*********************************************************************/
int gauche(int px,int py,int ox,int oy,int ix,int iy)
{
int vxi,vyi,vxp,vyp,determinant;

    // avoir les vecteurs oi et op
    vxi=ix-ox;
    vyi=iy-oy;
    vxp=px-ox;
    vyp=py-oy;

    // calculer le déterminant
    determinant= vxi*vyp-vxp*vyi;

    // inversion pour le point de vue de l'observateur, retourne 0 si
positif et
    // à droite et 1 si négatif et à gauche pour l'observateur

    return (determinant>0) ? 0 : 1;
}
/*********************************************************************
à partir de la fonction gauche, en tournant dans le sens inverse des
aiguilles d'une montre, voyons si le point (x,y) est ou pas dans triangle
a-b-c-
*********************************************************************/
int in_triangle(int x, int y, int ax,int ay,int bx,int by,int cx,int cy)
{
int res=0;
    // le point (x,y) est-il à gauche de chaque segment ?
    res+=gauche(x,y,ax,ay,bx,by);
    res+=gauche(x,y,bx,by,cx,cy);
    res+=gauche(x,y,cx,cy,ax,ay);
    return (res==3) ? 1 : 0;
}
```

Remarque :

A partir de la fonction gauche il est possible de savoir si un point se trouve dans un polygone constituée de plusieurs segments à condition qu'il soit convexe. Il suffit de vérifier que le point est à gauche de tous les segments constitutifs de la forme en tournant autour d'elle dans le sens inverse des aiguille d'une montre.

Résumé C3

1. AVOIR DES ACTEURS, FORMES OU PERSONNAGES, LES ANIMER, LES AFFICHER

Définir des acteurs à l'aide de structures

PRG 3.1

Quelles sont les variables dont j'ai besoin pour définir un player ? Un ennemi ? Sur quelles « structures de données » mon programme va reposer ?

Exemple de définition d'un type d'acteur dans notre prg : l'entité du type "t_entite"

```
typedef struct entite{
   int type;                  // type de l'entité : 0=rect, 1=cercle,
2=triangle
   int x,y;                   // position horizontal et verticale
   int tx,ty;                 // variables utilisées pour construire la
forme de l'entité
   int px,py;                 // pour déplacement de l'entité
   int color;                 // couleur de l'entité
}t_entite;

#define NBMAX      50          // Pour avoir plusieurs acteurs du même
t_entite *tab[NBMAX] ;         // type : faire un tableau
                               // - de structures t_entite ou
                               // - de pointeurs t-entite*
```

Remarque langage C :
Possibilité de définir une suite de constantes par des mots avec un **enum** de la façon suivante :

```
enum{ RECT, CERCLE, TRIANG, FIN};
```

Par defaut les mots de l'enum sont incrémentés de 1 en 1 à partir de 0 : RECT vaut 0, CERCLE 1, TRIANG 2, FIN 3. L'incrémentation est toujours de 1 en 1 mais il est possible de définir un ou plusieurs départ avec des affectations, par exemple :

```
enum{ RECT=3, CERCLE, TRIANG=7, FIN};
```

ici RECT vaut 3, CERCLE vaut 4, TRIANG vaut 7 et FIN vaut 8.

Affichage, méthode double buffer

Le principe du double buffer est simplement d'agencer l'ensemble des formes que l'on veut mouvoir dans une image bitmap intermédiaire (le double buffer) puis d'afficher d'un seul bloc cette image à l'écran en écrasant ce qui se trouvait précédemment affiché à l'écran. Ce qui donne les étapes suivantes :

1) Effacer l'image intermédiaire.
2) Pour n objets, modifier les positions des objets en fonction de leurs vitesses et trajectoires respectives etc.
3) Eventuellement contrôler les bords
4) Afficher les formes (cercles, rectangles triangles, dessins, petites images etc.) à leurs nouvelles positions dans la bitmap intermédiaire.
5) Plaquer cette image à l'écran.

Ajouter une image en fond

PRG 3.2

BITMAP* load_bitmap(const char *filename, RGB *pal);
Cette fonction charge un fichier bitmap (BMP, LBM, PCX, et TGA) en mémoire à partir du nom et chemin spécifié par la chaîne « filename » et retourne un pointeur sur une structure BITMAP. Si l'image est en 8 bits ses couleurs sont stockées dans une pallette « pal », pour les autres formats de

couleur l'argument « pal » peut être NULL. Si le format de l'image ne correspond pas au format courrant du programme l'image est convertie dans le format courant. Retourne NULL si erreur.

Pratiquer une homothétie

PRG 3.3

```
void stretch_blit(      BITMAP *source, BITMAP *dest,
                        int source_x, source_y, source_width,source_height,
                        int dest_x, dest_y, dest_width, dest_height);
```

La fonction stretch_blit ()permet de donner à une image les dimensions que l'on souhaite en réalisant sur elle une homotétie.
« source » et « dest » sont respectivement les images source et destination,
les variables source_... correspondent à la zone de l'image source que l'on veut copier mais attention il ne faut pas dépasser la taille de l'image source.
Les variables dest_... la taille de la zone de destination qui peut elle dépasser la taille de l'image destination.
Une homotétie est réalisée de la taille source à la taille destination. Source et destination doivent être de la même color depth, la source doit être une bitmap mémoire.

Remplacer les formes (rect, cercles etc.) par des images

Un affichage avec ou sans mode masque

PRG 3.4

Dans une image tous les pixels de la couleur du masque peuvent être ignorés lors des affichages. Ces couleurs sont 0 en 8 bits (en général le noir) et rose (maximun rouge et bleu, 0 vert) en true color. Les fonctions d'affichage en mode masque sont :

```
void masked_blit(     BITMAP *source, BITMAP *dest,
                int source_x, int source_y,
                int dest_x, int dest_y,
                int width, int height);
```

Identique à la fonction blit() mais les pixels de la couleur du masque sont comme transparents. Les deux bitmaps source et destination doivent être dans le même mode couleur (8,15,16,24 ou 32 bits).

```
void masked_stretch_blit(     BITMAP *source, BITMAP *dest,
                        int source_x, source_y, source_w, source_h,
                        int dest_x, dest_y, dest_w, dest_h);
```

Identique à la fonction stretch_blit() mais en mode masque.

2. SPRITES ET ANIMATION

Remplacer une image par une séquence d'images

Base de l'animation

PRG 3.5

La base d'un dessin animé consiste à afficher chronologiquement la succession des images qui décompose un mouvement. La succession des images est stockée dans un tableau et l'algorithme pour une petite séquence « sprite » affichée avec un double buffer est le suivant :

Au départ n_i = première image de l'animation

1) Effacer le buffer .
2) Afficher l'image n_i courante dans le buffer à la position voulue.
3) Incrémenter n_i pour passer à l'image suivante. Si n_i égale l'indice de la dernière image, la valeur de l'indice de la première lui est affectée.
4) Afficher le buffer à l'écran.
5) Mettre le processus en boucle, retourner en 1.

Bonhomme déplacé au clavier

PRG 3.6

Pour déplacer un personnage avec le clavier (haut, bas, gauche, droite) il faut plusieurs séquences d'images selon les vues souhaitées, par exemple de dos il monte , de face il descend, vers gauche il va à gauche et vers droite il va à droite. Chaque vue est une petite animation, et en fonction du pilotage au clavier il faut en gérer l'évolution de l'image courante.

A chaque appuie sur une touche pilote (les quatre flèches par exemple) on a besoin d'un compteur d'image qui permettra d'afficher les images de la séquence concernée dans l'ordre. Mais également à chaque fois que l'on appuie sur une de ces touches la position de l'image est modifiée, ce qui donne :

Dans la boucle d'évènements
Si une touche est appuyée
1) Effacer le buffer .
2) récupérer la touche
3) envoyer la touche dans un switch et pour chaque cas :
- vérifier si l'image concernée reste dans la zone de jeu
- si oui modifier les coordonnées en fonction de la touche appuyée (en x ou y)
- afficher l'image courante désignée par le compteur d'image dans le buffer à la nouvelle position
- incrémenter le compteur d'image pour la direction concernée
4) Afficher le buffer à l'écran.

Contrôler la vitesse de l'animation

PRG 3.7

Il est important de pouvoir découpler le mouvement du sprite dans l'espace (modification des coordonnées) du mouvement intrinsèque de l'animation (modification de l'image courante). Pour ce faire il est nécessaire de pouvoir avancer dans l'espace sans avancer dans la séquence d'animation. Il suffit d'ajouter un compteur de tours afin de réduire l'avancement de l'animation à une image tous les n tours.
Pour ce faire, soit une variable « tmps » qui compte la durée d'attente, une variable « maxtmps » qui définit l'attente maximum et une variable « imcourante » qui donne l'indice de l'image courante (toutes les images de l'animation sont sensées être stockées dans un tableau) on a :

```
if (++tmps>maxtmps){
   tmps=0;
   // imcourante va jusque NB_IMAGE-1 et retourne à 0
   imcourante=(imcourante+1)%NB_IMAGE;
}
```

« imcourante » est incrémenté de 1 uniquement lorsque « tmps » atteint « maxtemps », à chaque fois « tmps » est réinitialisé à 0. Il y a ainsi possibilité d'agir sur la vitesse de l'animation en fonction d'un déplacement donné.
Autre écriture plus compacte :

```
if ( !(++tmps%maxtemps))
   ++imcourante%=NB_IMAGE ;
```

Attention dans ce deuxième cas à ce que maxtemps ne soit JAMAIS égal à 0 sinon plantage cause division par 0.

Généralisation du contrôle d'un sprite

PRG 3.8

```
// intégration du controle vitesse dans la structure de données
typedef struct SPRITE{

  // le deplacement
  float x,y;          // position
  float px,py;        // pas du déplacement
```

```
  int tx,ty;          // taille

  // l'image
  int imcourante;     // l'image courante
  int nb_image;       // le nombre max des images de l'animation
  int dir;            // sens de l'animation (à reculons ou vancer)
  int tmps;           // pour compter le temps d'affichage de l'image
  int maxtmps;        // temps maxi pour chaque image

}t_sprite ;
```

Récupérer une série d'images stockée dans un seul fichier

Les images encombrent vite le répertoire d'un programme qui utilise plusieurs animations sprites. Le mieux est de les regrouper sur un fichier unique et de les récupérer une à une ensuite dans le programme avec par exemple une fonction du type :

```
BITMAP* recup_sprites( BITMAP*scr,     // bitmap origine
                       int w, int h,   // taille image de l'animation
                       int startx, int starty, // début de la série à
                                               //partir de la position x et y
                       int col,      // nombre de colonnes
                       int element)  // numéro de l'image dans la série
{
BITMAP *bmp;
int x,y;

            bmp=create_bitmap(w,h);
            if (bmp!=NULL){
                   x= startx+(element%col)*w;
                   y= starty+(element/col)*h;
                   blit(scr,bmp,x,y,0,0,w,h);
            }
            return bmp;
}
```

Animer plusieurs séquences simultanément

PRG 3.9

Quelles structure de données ?

```
// Pour controler un sprite :
typedef struct SPRITE{

   // le deplacement (sans float mais avec le même mécanisme de retard que
   // celui utilisé pour pour l'image)
   int x,y;            // position
   int px,py;          // pas du deplacement
   int tx,ty;          // taille
   int wx,wy;          // pour retarder avancement en x et y
   int xcmpt,ycmpt;    // pour compter le retard

    // l'image
    int imcourante;  // l'image courante
    int nb_image;    // le nombre max des images de l'animation
    int maxtmps;     // temps maxi pour chaque image
    int tmps;        // pour compter le temps d'affichage de l'image
    int dir;         // sens de l'animation (à reculons ou avancer)
    BITMAP**anim;    // un tableau de pointeurs pour stocker chaque anime
}t_sprite ;
```

```
// Pour contrôler NB_SPRITE, un tableau de t_sprite* :

#define NB_SPRITE  7
t_sprite* tab[NB_SPRITE] ;

// pour identifier l'indice de chaque sprite par un texte, un enum :

enum{DRAGON,POISSON,CRABE,ABEILLE,MOUSTIQUE,SERPENT};
```

3. DETECTER DES COLLISIONS

Test simple rectangle à rectangle

PRG 3.10

La collision a lieu lorsqu'une image rencontre une autre image. Pour le savoir on peut :
- utiliser une fonction « getpixel() » qui donne la couleur d'un pixel pour identifier sur quoi un pixel ou plusieurs pixels d'une image avance (le fond noir ou autre chose par exemple)
- faire un test d'intersection entre rectangles par exemple :

(x1, y1) et (x2,y2) sont les coordonnées des points hauts gauche de chaque rectangle, tx1, ty1 et tx2, ty2 correspondent aux tailles respectives des deux rectangles, les coordonnées des points bas droite sont respectivement (x1+tx1, y1+ty1) et (x2+tx2, y2+ty2).

Nous pouvons chercher à savoir si R1 est totalement à gauche ou à droite ou au dessus ou en dessous de R2 soit les tests équivalents suivants :

Si (x1+tx1 < x2 OU x1 > x2+tx2 OU y1+ty1 < y2 OU y1 > y2+ty2)
  Pas de collision
Sinon
  Collision

Si (x1+tx1 > x2 ET x1 < x2+tx2 ET y1+ty1 > y2 ET y1 < y2+ty2)
  Collision
Sinon
  Pas de collision

Par ailleurs dans différentes circonstances nous pouvons avoir besoin de savoir si un point est dans une zone rectangulaire, pour identifier une zone de clic souris par exemple, soit la fonction suivante :

```
// La fonction renvoie 1 si le point (x,y) passé en argument est dans le
// rect (top, left,right,bottom)
int dedans(int x, int y, int top,int left,int right,int bottom)
{
int res=0;
            if (x>left && x<right && y>top && y<bottom)
                  res=1;
            return res;
}
```

ANNEXE 1 : Allegro pour Microsoft Visual C++ 8

1. Installer Allegro pour Microsoft Visual C++ 8

Sur le site http://www.allegro.cc/files/ à la rubrique « Binary » télécharger :
- le zip prévu pour Visual C++ 8 et le zip Tools & examples .

Vous devez également vous procurer le SDK de DirectX. Le SDK complet fait prêt de 450 M° mais nous avons besoin juste du dossier include avec les fichiers d'entêtes (*).h et du dossier lib qui contient différentes librairies compilées (*).lib. Ces dossiers pour la version 7 de direct X sont diffusés sur le site http://alleg.sourceforge.net/wip.fr.html.

Cependant tout est regroupé sur le site http://fdrouillon.free.fr et à la rubrique « ressources » vous pouvez
- télécharger le package maison Visual C++ 8. Ce package contient tout ce qui est nécessaire : exemples, doc, include, lib, tools et tests et les dossiers include et lib du SDK de DirectX 9.

Ensuite il suffit :
1) De copier
   - le contenu du dossier 'include' de la librairie dans le dossier 'include' de Visual C++
   - le contenu du dossier 'include' du SDK de Direct X dans le dossier 'include' de Visual C++

2) De copier
   - le contenu du dossier 'lib' de la librairie allegro (alld42.pdb, alld.lib, alld_s.lib, alleg.lib, alleg_s.lib, alleg_s_crt.lib, allp.lib, allp_s.lib) dans le dossier 'lib' de Visual C++.
   - Le contenu du dossier 'lib' du SDK de Direct X dans le dossier 'lib' de Visual C++

3) Copier la ou les DLLs (alleg42.dll, alld42.dll, allp42.dll) dans le dossier WINDOWS/system ou WINDOWS/system32 (pour XP, Vista). Eventuellement on peut préférer copier la DLL choisie dans le répertoire où se trouve le programme qui en a besoin (voir 1.2.2 linkage)

2. Projet et configuration sous Visual C++ 8

Cette section s'appuie au départ sur un article du wiki Allegro écrit par Matthew Leverton que vous pouvez retrouver à http://wiki.allegro.cc/Visual_C%2B%2B_Express_2005.

2.1 Utiliser le templates C++ 8 (un projet tout fait)

Télécharger ce projet sur http://fdrouillon.free.fr le mettre de côté et, plutôt que d'avoir à faire et configurer soi-même un nouveau projet, le copier pour chaque nouveau projet. Changer ensuite les noms de projet et de fichier dans visual C++.

2.2 Faire un projet avec Visual C++

1) Ouvrir Visual C++

2) Dans le menu fichier sélectionner Nouveau / Projet, une fenêtre s'ouvre.

3) Sur le côté gauche de la fenêtre, sélectionner « Visual C++ », du contenu s'affiche à droite, dans « Modèles Visual Studio installés » sélectionner « Projet vide ».

4) En bas, dans le champ prévu, taper un nom pour le projet.

5) Juste en dessous choisir le répertoire dans lequel le projet sera.
Visual crée automatiquement un dossier pour le projet. Mais un projet peut éventuellement contenir plusieurs « solutions » c'est-à-dire plusieurs programmes sur lesquels il est alors possible de travailler simultanément. Dans ce cas chaque solution devra avoir son propre répertoire à l'intérieur du projet et pour ce faire il faut cocher la case « créer le répertoire pour la solution ». En revanche s'il n'y a qu'une seule solution pour le projet (un seul programme à travailler à la fois) un sous répertoire est inutile et il vaut mieux décocher cette case.

6) Cliquer OK, le projet vide est constitué.

7) A gauche dans l'explorateur, sélectionner le dossier « fichiers sources » en cliquant avec le bouton droit. Un menu contextuel s'ouvre : sélectionner « Ajouter / Nouvel élément ». Une fenêtre s'ouvre.

8) Dans la partie gauche sélectionner « code » et dans le contenu qui s'affiche à droite sélectionner « Fichier C++ (.cpp) ». En bas entrer un nom pour le fichier source et cliquer sur « ajouter ».

Maintenant il y a tout ce qui faut pour écrire du code, reste à configurer le projet pour qu'il compile.

2.2 Configurer le projet avec Visual C++

Par défaut Visual C++ suppose deux configurations : le mode « debug » et le mode « release ». Mais il est possible de créer d'autres configurations, notamment celles dites « static ». Nous allons détailler celles qui sont possibles avec Allegro. Pour commencer voici un test de la configuration « debug »

2.3.1 Test de la configuration « debug »

1) Vérifier que la configuration actuellement sélectionnée est bien sur « debug » (une petite fenêtre dans la barre d'outils, si vous cliquez sur la petite flèche à droite un menu s'ouvre avec la liste « debug, release, gestionnaire de configurations », sélectionner « debug ».)

2) Dans le menu « Projet » en haut, sélectionner « propriétés ». Une fenêtre « page de propriétés » s'ouvre. A gauche ouvrir « Propriétés de configuration ».

3) Ouvrir C/C++
- sélectionner « général » à gauche et à droite, à « format des informations de débogage » sélectionner dans le menu déroulant « base de données du programme pour modifier et continuer (/ZI) ».

- sélectionner « optimisation » à gauche et à droite, à « optimisation » sélectionner « désactivé (/Od) »

- sélectionner « génération de code » à gauche et à droite, à « bibliothèque runtime » sélectionner « DLL de débogage multithread (/MDd)

4) ouvrir « éditeur de liens » sur le côté gauche.
- sélectionner « entrée » à gauche et à droite écrivez « alld.lib » à « Dépendance supplémentaires »

- sélectionner « débogage » à gauche et à droite, à « Génération des informations de débogage » sélectionner « Oui (/DEBUG) »

5) Cliquer sur « Appliquer » en bas de la fenêtre, puis sur OK.

Maintenant pour tester cette configuration copier coller le code ci-dessous dans votre fichier source :

```
#include <allegro.h>
int main()
{
   allegro_init();
```

```
   allegro_message("coucou");
   return 0;
}
END_OF_MAIN();
```

Presser F7 pour construire le projet. S'il n'y a pas d'erreur presser cntl-F5 pour lancer le programme. Vous devez voir apparaître une petite fenêtre avec « coucou » écrit au dessus d'un bouton « ok »

2.3.2 A propos de l'utilisation du débogueur

Le débogueur est un point fort du logiciel Visual C++. Pour pouvoir l'utiliser avec Allegro votre projet doit être initialisé en mode fenêtre (voir dans section : Entrer dans le mode graphique, Initialiser un mode graphique, la fonction set_gfx_mode() avec le paramètre GFX_AUTODETECT_WINDOWED) et la configuration active doit être la configuration « Debug ». La liste des configurations possibles est à droite de la petite flèche verte « démarrer le débogage » dans la barre d'outils standard de l'interface de Visual C++.

Pour démarrer une cession de débogage, juste pressez F5 ou dans le menu « Déboguer » sélectionner « Démarrer le débogage ». L'application est alors lancée et tourne normalement tant qu'il n'y a pas de crash. En particulier le débogueur de Visual C+ trace tous les pointeurs et met en évidence les erreurs d'utilisation.

Comme dans tout débogueur il est possible de placer dans le code source des « break points ». Ils permettent d'arrêter le programme dans son fonctionnement là où se trouve un break point et de basculer dans le code source correspondant et de l'examiner : contenu des variables, adresses des pointeurs etc. Il est ainsi possible de vérifier le fonctionnement d'un algorithme et des variables prévues avec. Pour relancer l'exécution il suffit de réappuyer sur F5. Pour placer un break point il suffit de cliquer dans la marge à gauche de la ligne où l'on veut arrêter le programme.

2.3.3 Différentes configurations possibles avec Allegro

A part le mode débug que nous venons de voir, il y a au total 6 configurations possibles :

Linkage dynamique (avec dll) :
- **Release**
- Debug
- Pofile

Linkages static (sans dll) :
- **Static Release** et Static Release + Static Runtime (voir section diffusion du programme)
- Static Debug
- Static Profile

Les trois premières font appel à un linkage dynamique et les programmes résultants ont besoin d'une DLL pour fonctionner. Les trois suivantes font appel à un linkage static. Elles n'ont pas besoin de DLL pour fonctionner et le code correspondant est inclus dans l'exécutable.

Le mode debug permet le débogage. Il est à utiliser lorsque l'on travailler sur l'exécutable, pendant son développement, mais il est déconseillé pour la version finale que vous souhaitez distribuer. Pour un programme fini que vous voulez distribuer compilez votre programme en mode « release ».

Il reste un troisième mode, le mode « profile ». C'est un mode qui permet d'avoir plus d'informations sur l'ensemble du code source utilisé dans le programme mais il rare d'avoir à l'utiliser et nous ne le détaillerons pas davantage ici.

Le mode release (dynamique ou static), en caractère gras ci-dessus, est probablement le plus important dans la mesure où il est possible de se passer des autres et que celui-ci est indispensable pour la distribution de son programme.

2.3.4 Ajouter une configuration

Ce n'est pas obligé mais dans le projet il peut y avoir plusieurs configurations. Celle définie comme active est alors utilisée à la compilation. Pour rendre active une configuration il suffit de la sélectionner dans la liste des configurations à droite de la petite flèche verte dans la barre d'outils standard de l'interface de Visual C++.
Visual C++ suppose que l'on aura au moins deux configurations dûment paramètrées : une « debug » et une « release ». Il est possible d'en ajouter d'autres par exemple une « static debug » et une « static release ».

Pour ce faire dans le menu « projet » sélectionner « propriétés de [nom du projet] ». La fenêtre des propriétés s'ouvre. En haut à gauche il y a la liste des configurations du projet et la configuration active est mentionnée « Active ». En haut à droite il y a le gestionnaire des configuration, pour ajouter une configuration cliquer sur ce bouton.

Une nouvelle fenêtre s'ouvre, en haut à gauche dans « Configuration de la solution active » sélectionner « nouveau ». Dans la boite de dialogue qui apparaît entrer le nom pour la nouvelle configuration et éventuellement le nom d'une configuration existante à partir de laquelle copier des paramètres. Ensuite cliquer sur fermer. Il reste à paramétrer chaque configuration.

2.3.5 Paramétrages des configurations

Dans la fenêtre « page de propriétés » accessible via le menu « projet / Propriétés de [nom du projet] », sélectionner dans la liste en haut à gauche la configuration à configurer. Ensuite entrer les valeurs voulues selon les indications données dans le tableau ci-dessous.

Configurations	Rubriques	Valeurs
1 **Realease**	C/C++ / Génération de code / Bibliothèque runtime	DLL multithread (/MD)
	Editeur de liens / Entrée / Dépendances supplémentaires	alleg.lib
2 **Debug**	C/C++ / Général / Format des informations de débogage	Base de données du programme pour Modifier & Continuer (/ZI)
	C/C++ / Optimisation / Optimisation	Désactivé (/Od)
	C/C++ / Génération de code / Bibliothèque runtime	DLL de débogage multithread (/MDd)
	Editeur de liens / Entrée / Dépendances supplémentaires	alld.lib
	Editeur de liens / Débogage / Génération des informations de débogage	Oui (/DEBUG)
3 **Profile**	C/C++ / Génération de code / Bibliothèque runtime	DLL multithread (/MD)
	Editeur de liens / Entrée / Dépendances supplémentaires	allp.lib
4 **Static Release**	C/C++ / Préprocesseur / Définitions du préprocesseur	ALLEGRO_STATICLINK
	C/C++ / Génération de code / Bibliothèque runtime	DLL multithread (/MD)

	Editeur de liens / Entrée / Dépendances supplémentaires	alleg_s.lib kernel32.lib user32.lib gdi32.lib comdlg32.lib ole32.lib dinput.lib ddraw.lib dxguid.lib winmm.lib dsound.lib
	Editeur de liens / Ligne de commande / options supplémentaires	/LTCG
5 Static Release, Static Runtime	C/C++ / Préprocesseur / Définitions du préprocesseur	ALLEGRO_STATICLINK
	C/C++ / Génération de code / Bibliothèque runtime	Multithread (/MT)
	Editeur de liens / Entrée / Dépendances supplémentaires	alleg_s_crt.lib kernel32.lib user32.lib gdi32.lib comdlg32.lib ole32.lib dinput.lib ddraw.lib dxguid.lib winmm.lib dsound.lib
	Editeur de liens / Ligne de commande / options supplémentaires	/LTCG
6 Static Debug	C/C++ / Préprocesseur / Définitions du préprocesseur	ALLEGRO_STATICLINK
	C/C++ / Général / Format des informations de débogage	Base de données du programme pour Modifier & Continuer (/ZI)
	C/C++ / Optimisation / Optimisation	Désactivé (/Od)
	C/C++ / Génération de code / Bibliothèque runtime	DLL de débogage multithread (/MDd)
	Editeur de liens / Entrée / Dépendances supplémentaires	alld_s.lib kernel32.lib user32.lib gdi32.lib comdlg32.lib ole32.lib dinput.lib ddraw.lib dxguid.lib winmm.lib dsound.lib
	Editeur de liens / Débogage / Génération des informations de débogage	Oui (/DEBUG)
7 Static Profile	C/C++ / Préprocesseur / Définitions du préprocesseur	ALLEGRO_STATICLINK
	C/C++ / Génération de code / Bibliothèque runtime	DLL multithread (/MD)
	Editeur de liens / Entrée / Dépendances supplémentaires	allp_s.lib kernel32.lib user32.lib gdi32.lib comdlg32.lib ole32.lib dinput.lib ddraw.lib dxguid.lib winmm.lib dsound.lib

L'idéal serait bien sur de faire un modèle de projet préconfiguré avec Visual C++…

2.3.6 Diffusion du programme : attention aux DLL requises

Microsoft a introduit une nouvelle façon d'utiliser les DLL et il ne suffit plus de mettre une DLL dans le même folder qu'un exécutable pour que tout fonctionne sous XP ou les versions ultérieures. Peu de personne ont les DLLs du runtime de Visual C++ 8 installées sur leur machine et il peut s'avérer nécéssaire de les fournir avec le programme en plus de la DLL Allegro.

Dans ce cas il faut ajouter un dossier nommé Microsoft.VC80.CRT dans le répertoire du programme.

Ce dossier est inclus dans le package Visual C++ 8 fourni à la rubrique « ressources » sur http://fdrouillon.free.fr Mais il peut également être à téléchargé sur :
http://members.allegro.cc/matthew/Microsoft.VC80.CRT.zip

Ce dossier contient les DLLs supplémentaires nécessaires. Il doit être placé tel quel avec le même nom dans le prépertoire du programme ce qui donne par exemple une arborescence du type :

C:\mon_jeu
- Alleg42.dll
- Mon_jeu.exe
- Microsoft.VC80.CRT
    - Microsoft.VC80.CRT.manifest
    - msvcm80.dll
    - msvcp80.dll
    - msvcr80.dll

Encore un inconvénient, sous windows 98 et 2000 les utilisateurs devront sortir les DLL du dossier Microsoft.VC80.CRT et les placer au même niveau que l'exécutable.

Configuration static
Le plus simple est d'avoir une version static du programme qui incorpore toutes les DLLs, Allegro et C Runtime et fonctionne sur toutes les plateformes Windows. Dans ce cas il n'y a plus besoin d'installer aucune DLL. Pour ce faire il faut choisir la configuration n°5 « Static Release, Static Runtime »