Contrôle informatique : Librairie graphique Allegro

ECE – INGE1
R. FERCOQ
semaine du 08/04/2013

Durée : 1H30

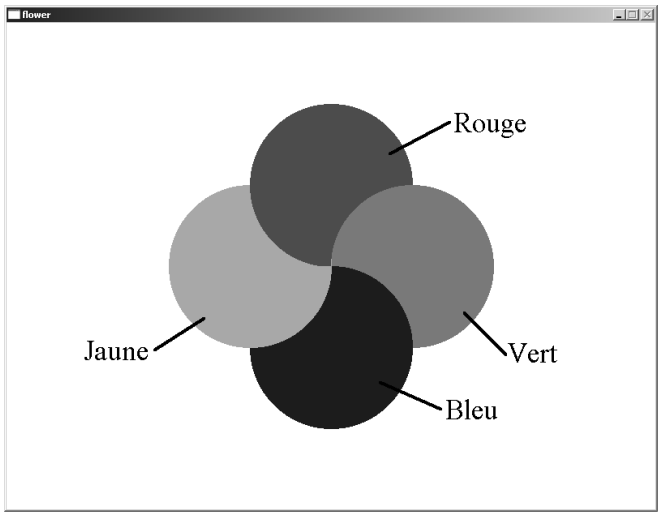
Calculatrices non autorisées. Aucun document.
Répondez sur votre copie, pas sur l'énoncé.

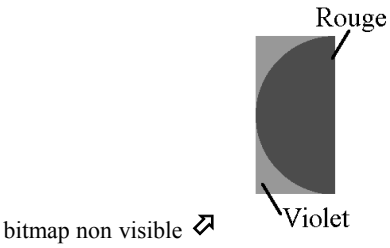
Un "squelette" de projet Allegro est en annexe : les différentes parties sont indiquées par des lettres A B C … : **il est inutile de recopier ce squelette**. Pour les différentes questions vous préciserez dans quelles sections (A B C...) votre code devra s'écrire.

Une liste des fonctions Allegro utiles et de leurs paramètres est en annexe.

1. Dessin statique: géométrie, couleurs, cadres bitmap *4 points*

Quelles seraient les instructions à ajouter au squelette fourni en annexe pour obtenir le dessin suivant **directement à l'écran** sur **fond blanc** :





Les 4 disques font tous 200 pixels de **diamètre**. Les disques opposés (rouge/bleu et jaune/vert) sont tangents au centre de l'écran (de résolution 800 par 600). Pour obtenir l'effet de superposition cyclique il est suggéré de faire en 2 étapes :

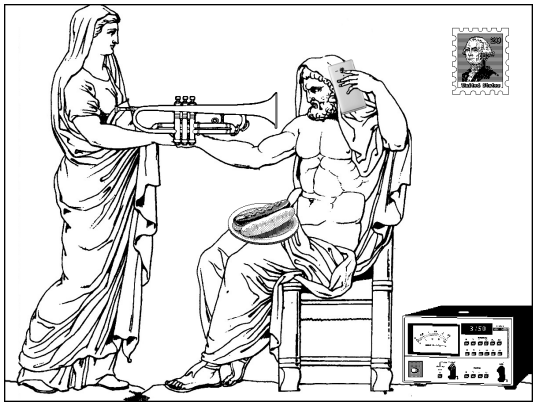
- D'abord afficher directement à l'écran les 4 disques dans l'ordre rouge, vert, bleu, jaune. C'est pas mal mais à la fin de cette séquence le jaune va "mordre" sur le disque rouge.
- Ensuite on va reconstituer la partie gauche du disque rouge, sans abimer le jaune, en "blittant avec transparence" un demi-disque rouge qui aura été préalablement dessiné sur fond violet dans une bitmap temporaire, non visible (figure de droite). Cette bitmap temporaire est à créer et à libérer `void destroy_bitmap(BITMAP *bitmap);` dès que le dessin est terminé.

Pour rappel allegro coupe automatiquement tout ce qui dépasse d'un cadre de dessin : pour obtenir un demi-disque il suffit de dessiner un disque au bon endroit sur une bitmap temporaire de bonne dimension.

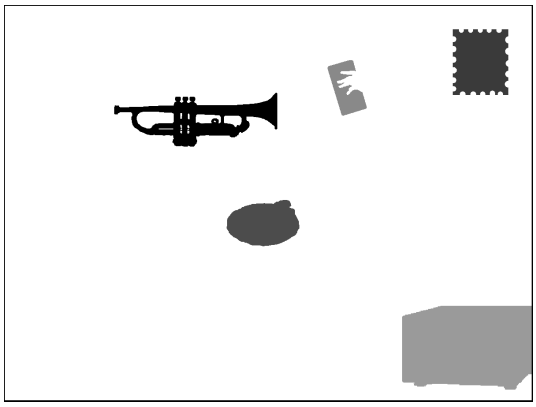
Dernière remarque, le dessin est fait **une fois**, lors du lancement du programme (pour ce projet la "boucle de jeu" ne sert qu'à attendre l'appui sur la touche échap. pour fermer le programme)

2. Interaction souris et traitement pixel par pixel *10 points*

On dispose de 2 fichiers images 800 par 600 dans le répertoire de projet :



mytho.bmp



anachron.bmp

Comment à partir du squelette fourni en annexe obtenir un programme qui :

- Charge dans les bitmaps **`mytho`** et **`anachron`** les fichiers "mytho.bmp" et "anachron.bmp"
  Vous coderez la vérification de bon chargement, qui affiche un message et termine l'exécution avec **`allegro_exit();`** et **`exit(EXIT_FAILURE);`** en cas de problème.
- Affiche à l'écran l'image **`mytho`**
- Permet ensuite à l'utilisateur d'interagir avec la souris pour découvrir les anachronismes de la scène : lors d'un clic si il y a un objet anachronique à cet endroit alors cet objet est colorisé selon la couleur présente en correspondance sur **`anachron`**.

Le test de présence d'un objet anachronique à l'endroit du clic pourra se faire à l'aide d'un ***`getpixel`*** *à ces coordonnées mais sur la bitmap anachron : le fond de l'image anachron est blanc (parfaitement : 255,255,255), toute autre couleur signale un clic dans une zone anachronique. Une valeur "non blanche" déclenchera donc un balayage complet pixel par pixel: à chaque fois qu'on voit cette valeur sur anachron on fera la moyenne entre cette couleur et la couleur sur mytho, cette moyenne sera directement ré-affectée* ***sur mytho*** *pour ce pixel. C'est la colorisation. Attention : il est nécessaire de moyenner pour chaque composante rouge vert bleu. Enfin on en profitera pour mettre un pixel blanc* ***sur anachron*** *pour "effacer" l'objet et qu'il ne déclenche plus de traitement même si on re-clique dans la même zone.*

- Quand les 5 objets anachroniques ont été découverts un message de félicitation apparaît dans la console (simple printf) et la boucle de jeu se termine (il est nécessaire de modifier le while...)
- Sinon quand on presse échap. le jeu se termine mais sans message de félicitation

Il n'est pas demandé de pouvoir rejouer sans relancer le programme, les bitmaps peuvent donc être directement modifiées, conformément aux indications en italique.
Pour des raisons d'efficacité les traitements pixel par pixel ne se font jamais directement sur screen mais sur les bitmaps (nécessitant éventuellement un ré-affichage à l'écran si le résultat doit être rendu visible).
Astuce : pour **tester l'égalité** d'une couleur (obtenue avec un getpixel, ou un makecol) avec une autre couleur (obtenue avec un autre getpixel) il suffit de tester une seul égalité de valeur, il n'est pas nécessaire de faire 3 tests sur les 3 composantes rouges vertes bleues associées.

Précisez bien dans quelle(s) section(s) vous ajoutez vos lignes de code.

3. Cinématiques et structures acteurs *6 points*

Dans cet exercice on souhaite étudier les sous-programmes de déplacement (pas d'affichage) d'un programme qui fait bouger un nombre variable d'acteurs autonomes, par exemple des soucoupes volantes. Les soucoupes se déplacent en ligne droite selon un mouvement rectiligne uniforme.
Les soucoupes de type 0 rebondissent comme une boule de billard dès qu'un coté touche un bord écran (800 par 600 mais utilisez les globales SCREEN_W et SCREEN_H)
Les soucoupes de type 1 disparaissent complètement par un bord écran avant de ré-apparaitre progressivement par le bord opposé : gestion des bords type PacMan.

Les structures suivantes sont supposées définies en section B (inutile de recopier)

```
// chaque acteur qui se déplace
typedef struct acteur
{
    int x, y;         // coordonnées (du coin supérieur gauche)
    int dx, dy;       // vecteur déplacement
    BITMAP * sprite; // image pour cet acteur
    int type;         // 0 pour un acteur rebond billard, 1 pour un acteur PacMan
} t_acteur;

// Une collection d'acteurs
typedef struct listeActeurs
{
    int max;         // nombre maxi d'éléments = taille du tableau de pointeurs
    t_acteur **tab; // le tableau de pointeurs sur acteurs
} t_listeActeurs;
```

La vague d'assaut est représentée par un tableau de pointeurs sur structures, les cases "non utilisées" de ce tableau ayant par convention la valeur NULL (pas de soucoupe à cette case). Toutes les données utiles sont supposées déjà allouées/chargées/initialisées dans ces structures. Pour connaître la taille (largeur et hauteur) d'une soucoupes on se basera sur son sprite associé.

Ecrire le sous-programme **`actualiserActeur`** qui reçoit **comme seul paramètre** un pointeur sur un t_acteur, qui met à jour la position de cet acteur à partir de son vecteur déplacement et qui gère les bords selon son type. **Aucun affichage**.

Ecrire le sous-programme **`actualiserListeActeurs`** qui reçoit **comme seul paramètre** un pointeur sur un t_listeActeurs et qui actualise tous les acteurs référencés en appelant le sous-programme précédent.

Annexes

SQUELETTE DE PROJET ALLEGRO

Il est inutile de recopier ce code, mais précisez bien dans quelle(s) section(s) les morceaux de programme que vous écrivez doivent se trouver. Ne répondez pas sur l'énoncé, il n'y a pas la place.

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

// SECTION A : CONSTANTES #define
    ⋮

// SECTION B : DEFINITIONS DES STRUCTURES
    ⋮

// SECTION C : PROTOTYPES DES SOUS-PROGRAMMES
    ⋮

// Programme principal
int main(int argc, char *argv[])
{
    // SECTION D : DECLARATIONS DES VARIABLES DU MAIN
    ⋮

    // SECTION E : INITIALISATION ALLEGRO
    srand(time(NULL));
    allegro_init();
    install_keyboard();
    install_mouse();

    // SECTION F : OUVERTURE MODE GRAPHIQUE
    set_color_depth(desktop_color_depth());
    if (set_gfx_mode(GFX_AUTODETECT_WINDOWED,800,600,0,0)!=0)
    {
        allegro_message("probleme mode graphique");
        allegro_exit();
        exit(EXIT_FAILURE);
    }
    show_mouse(screen);

    // SECTION G : AVANT BOUCLE JEU
    ⋮


    // SECTION H : BOUCLE JEU
    while (!key[KEY_ESC])
    {
        ⋮
    }

    // SECTION I : TERMINER LE PROGRAMME

    return 0;
}
END_OF_MAIN();


// SECTION J : DEFINITIONS DES SOUS-PROGRAMMES
    ⋮
```

LISTE DE FONCTIONS, TYPES ET VARIABLES ALLEGRO UTILES

void allegro_message(const char *text_format, ...);
Affiche une popup avec un message.

screen : extern BITMAP *screen;
C'est l'identifiant de l'écran réel

SCREEN_W SCREEN_H
Largeur (Width) et Hauteur (Height) de la sortie graphique Allegro (l'écran Allegro)

void clear_bitmap(BITMAP *bmp);
Efface (en noir) la bitmap en paramètre.

void clear_to_color(BITMAP *bmp, int color);
Efface (en couleur color) la bitmap en paramètre.

int makecol(int r, int g, int b);
Pour fabriquer l'entier représentant une couleur à partir de Red/Green/Blue
Chaque composante est donnée entre 0 (minimum) et 255 (maximum)

int getr(int c); int getg(int c); int getb(int c);
Permet d'accéder aux 3 composantes rouge vert et bleu d'une couleur (entier c)
Chaque composante est entre 0 (minimum) et 255 (maximum)

void putpixel(BITMAP *bmp, int x, int y, int color);
Poser un seul pixel de couleur sur une BITMAP ou sur l'écran (screen) en x y
La couleur précédente du pixel est écrasée.

int getpixel(BITMAP *bmp, int x, int y);
Lire la couleur du pixel de la BITMAP ou de l'écran (screen) en x y
La valeur récupérée est un int, équivalent à une couleur obtenue avec un makecol

void rectfill(BITMAP *bmp, int x1, int y1, int x2, int y2, int color);
Dessine un rectangle plein. (x1,y1) : coin supérieur gauche (x2,y2) : coin inférieur droit

void circlefill(BITMAP *bmp, int x, int y, int radius, int color);
Dessine un cercle plein (disque) (x,y) : coordonnées du centre radius : rayon du disque

typedef struct BITMAP ...
Structure qui contient les images (chargées depuis fichiers ou dessinées par programme)
Les bitmaps sont toujours déclarées et utilisées comme pointeurs : BITMAP *
Pour accéder à la largeur et à la hauteur d'une BITMAP après création ou chargement :
bmp->w // largeur (width) bmp->h // hauteur (height)

BITMAP *create_bitmap(int width, int height);
Allouer et initialiser une BITMAP "vide" de taille width(largeur) x height(hauteur)

BITMAP *load_bitmap(const char *filename, RGB *pal);
Charge l'image d'un fichier .bmp dans une BITMAP créée sur mesure.
Retourne le pointeur sur la BITMAP en cas de succès, NULL en cas d'échec (à tester)
Nous n'utilisons pas les palettes : on indiquera NULL dans le paramètre pal.

int save_bitmap(const char *filename, BITMAP *bmp, const RGB *pal);
Sauve l'image d'une BITMAP dans un fichier .bmp (ou .tga ou .pcx)
Nous n'utilisons pas les palettes : on indiquera NULL dans le paramètre pal.

void blit(BITMAP *source, BITMAP *dest, int source_x, int source_y, int dest_x, int dest_y, int width, int height);
Copie une zone rectangulaire de la BITMAP source vers la BITMAP destination.

void draw_sprite(BITMAP *bmp, BITMAP *sprite, int x, int y);
Dessine la totalité de l'image sprite sur la bitmap bmp, en évitant de dessiner là où les pixels du sprites sont "transparents" (violet...)
Attention par rapport à un blit, ici la destination est indiquée avant la source.

key : extern volatile char key[KEY_MAX];
Tableau de booléens (Vrai/Faux) indiquant si une touche est enfoncée
Entre crochet mettre l'identifiant du scancode de la touche :
KEY_A … KEY_Z KEY_0 … KEY_9 KEY_SPACE KEY_ENTER …

mouse_b : extern volatile int mouse_b;
Contient l'état instantané des boutons de la souris.
mouse_b&1 : Booléen vrai si le bouton gauche est enfoncé, faux sinon
mouse_b&2 : Booléen vrai si le bouton droit est enfoncé, faux sinon

mouse_x : extern volatile int mouse_x; **mouse_y** : extern volatile int mouse_y;
Coordonnées instantanées de la souris (bout du pointeur)