Contrôle informatique : Librairie graphique Allegro

ECE – INGE1

03/05/2014

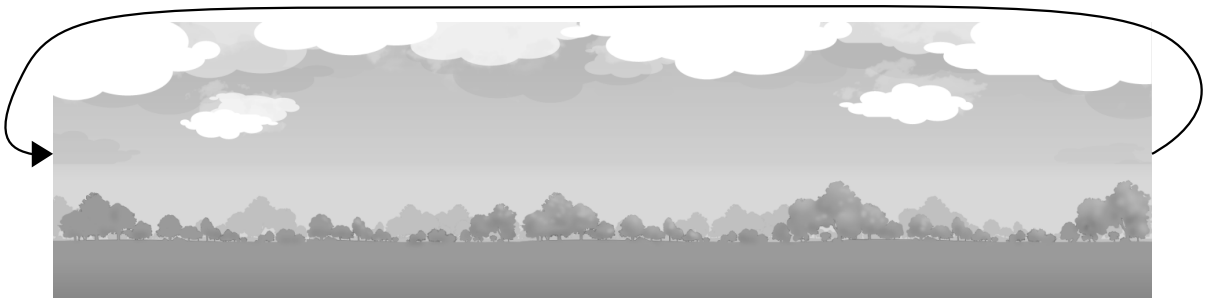
Durée : 1H30

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

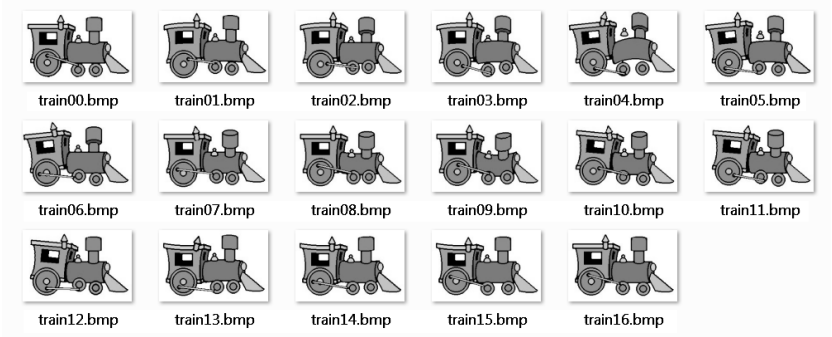
Un "squelette" de projet Allegro est en annexe : les différentes parties sont indiquées par des lettres A B C … : **il est inutile de recopier ce squelette**. Pour les différentes questions vous préciserez dans quelles sections (A B C...) votre code devra s'écrire. Une liste des fonctions Allegro utiles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

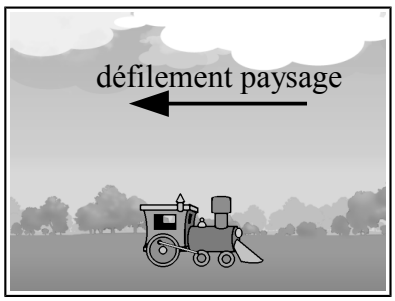
1. Animation, blit, défilement *8 points (2 + 6)*

L'objectif est de réaliser une animation d'une locomotive avançant sur un paysage qui défile de droite à gauche. La locomotive reste à une position fixe par rapport à l'écran mais est animée par affichage successif et cyclique de 17 dessins (principe du dessin animé).
L'animation est "infinie" : le paysage tourne "en boucle". Touche echap. pour quitter.



paysage.bmp : arrivé au bout à droite on "recolle" le même graphisme au début à gauche





résultat écran

Le fichier image paysage.bmp a la même hauteur que l'écran, il est plus large que l'écran.
Les fichiers images du train sont toutes de même dimension, les graphismes y sont dessinés sur fond magenta (255,0,255).
Les tailles exactes de ces images ainsi que de l'écran ne sont pas connues au moment d'écrire le code : le code doit être "paramétrique" et s'adapter automatiquement à ces différentes valeurs selon les éléments fournis par le graphiste, et il est possible que le projet final ne soit pas sur un écran 800x600 mais dans une autre résolution graphique (autres valeurs pour set_gfx_mode...)
La locomotive doit être centrée horizontalement sur l'écran, et placée verticalement de telle sorte que la base de l'image (les roues) soit à 50 pixels au dessus de la base de l'écran.

TSVP ↳

L'animation doit se faire à environ 50 images par seconde (fonction **`rest`** : paramètre en millisecondes). La vitesse de défilement sera de 100 pixels par seconde. L'animation de la locomotive doit se faire à 25 dessins par seconde : on ne passe d'un graphisme locomotive au suivant qu'une fois sur deux.
Pour éviter tout risque de clignotement, l'animation sera réalisée en "double-buffer".

Le nombre d'images de locomotive est défini en section A :
```
#define NBLOCO 17
```

Vous disposez des sous-programmes suivants, **supposés déjà écrits par un collègue**

// Charger image à partir d'un fichier, avec gestion de l'échec éventuel (abandon de programme)
```
BITMAP *chargerImage(char *nomFichierImage);
```

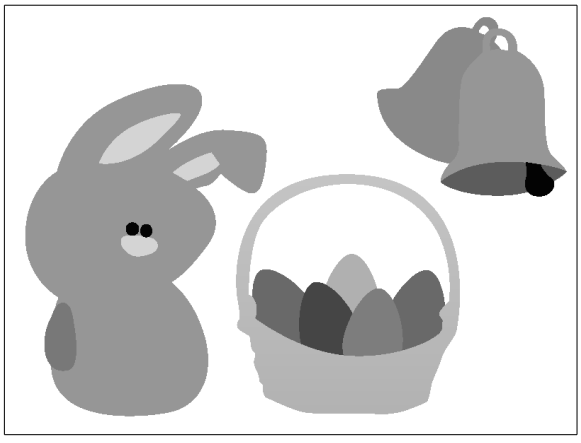
// Chargement de tous les fichiers "train00.bmp" "train01.bmp" … dans le tableau de BITMAP
```
void chargerLocos(BITMAP *locos[NBLOCO]);
```

a) Dessinez un ou plusieurs schémas représentant la géométrie du projet, en indiquant des cotes (mesures) se rapportant aux noms de variables qui seront utilisés dans le programme. Explicitez les formules qui vous permettrons de connaître toutes les valeurs utiles à partir des valeurs de départ (dans l'énoncé ou récupérables à l'exécution avec Allegro)

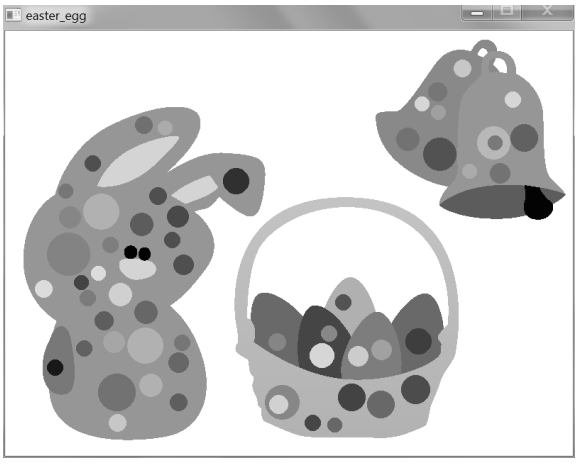
b) Ecrivez le code, à ajouter au squelette Allegro, pour obtenir le résultat décrit ci dessus. Précisez bien dans quelle(s) section(s) vous ajoutez vos lignes de code. Il n'est pas indispensable de découper en sous-programmes (les 2 sous-programmes chargerImage et chargerLocos sont supposés disponibles, ils ne sont pas demandés)

2. Détection niveau pixel et traitement d'image *6 points (3 + 3)*

On dispose d'un fichier image "paques.bmp" dans le répertoire de projet. Le graphisme est constitué de zones de couleurs unies sur fond blanc. On souhaite réaliser un programme qui charge ce fichier, applique un algorithme qui dessine 57 disques de couleur aléatoires dans les zones de couleur (pas sur le fond blanc), et affiche le résultat à l'écran. Echap. pour quitter.

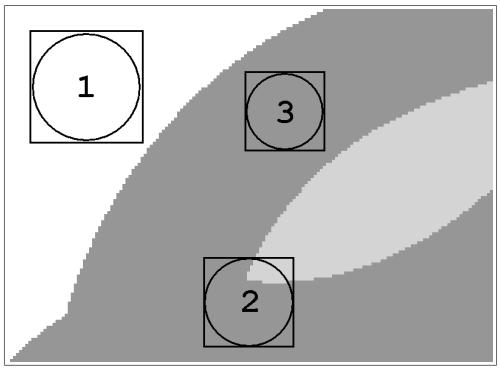


paques.bmp



résultat écran (moins vilain en couleurs !)

Les 57 disques sont à des positions aléatoires, ils ont un rayon aléatoire entre 10 et 30 pixels. Pour éviter qu'un disque soit "à cheval" entre 2 zones de couleurs différentes on testera que la zone envisagée est bien de couleur uniforme. Pour simplifier on pourra se contenter d'étudier les pixels du carré circonscrit au disque. Voir illustration page suivante.



Une fonction **`placeOk`** renvoie un booléen Faux si le carré circonscrit au disque dont les coordonnées du centre et le rayon qui lui sont donnés en paramètre contient des pixels de couleurs distinctes (cas 2) ou si la couleur est unie mais est blanche (cas 1). Elle retourne Vrai sinon (cas 3). L'image est nécessairement reçue en paramètre (mais n'est pas modifiée).
Pour cette fonction la gestion des bords image n'est pas demandée (les accès hors image ne plantent pas)

a) ***Ecrire la fonction `placeOk` selon les indications ci dessus.***

b) ***Ecrire la procédure `decorer` qui reçoit une BITMAP (de taille arbitraire) et qui dessine dessus 57 disques qui respectent les contraintes de placement (en utilisant `placeOk`)***

Le reste du programme n'est pas demandé.

3. Cinématiques et structures acteurs *6 points (4 + 2)*

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)

```
typedef struct acteur       // chaque acteur qui se déplace
{
    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;

typedef struct listeActeurs    // Une collection d'acteurs (vague d'assaut)
{
    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.

a) ***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.***

b) ***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)