Université de Versailles – Saint-Quentin 2015–2016
Licence 1

IN 200 – Polycopié de TD

Sandrine Vial – Sandrine.Vial@uvsq.fr
Version du 7 mars 2016.

Table des matières

1 TD 01 – Rappels et premiers rebonds **2**
1.1 Rappels . . . 2
1.2 Premiers rebonds . . . 3

2 TD 02 – Tableaux et Dégradés **4**

3 TD 03 et TD 04 – Balles et raquettes **5**
3.1 Tableau de balles . . . 6
3.2 Pong . . . 6

4 TD 05 et 06 – Horloge Analogique et Digitale **8**
4.1 Horloge analogique . . . 8
4.2 Horloge Digitale. . . . 8
4.3 Chronomètre . . . 9

A Annexe : types, variables constantes et fonctions disponibles **10**
A.1 Types, Variables, Constantes . . . 10
A.2 Affichage . . . 10
A.3 Gestion d'événements clavier ou souris . . . 11
A.4 Dessin d'objets . . . 11
A.5 Écriture de texte . . . 12
A.6 Lecture d'entier . . . 12
A.7 Gestion du temps . . . 12
A.8 Valeur aléatoires . . . 13
A.9 Divers . . . 13

Introduction

Environnement de programmation

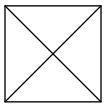
— La machine virtuelle à utiliser pour cette UE est Licence Info(Linux) qui est disponible sur le site du cartable numérique.
— Le présent poly est disponible sur l'espace e-campus de l'IN200.
— L'environnement de programmation basé sur SDL est dans la machine virtuelle.

1 TD 01 – Rappels et premiers rebonds

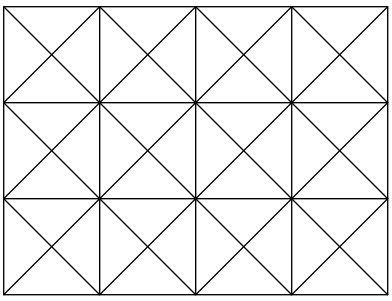
Le but de cette partie est de se familiariser avec l'environnement de programmation en faisant quelques fonctions élémentaires.

1.1 Rappels

Exercice 1 *Des Croix*
On souhaite effectuer le dessin suivant :



1. Écrire une fonction qui, étant donnés un point P et une longueur $long$, affiche la croix dessinée ci-dessus, avec : le point en bas à gauche du carré est P et la longueur du carré est $long$.
2. Écrire une fonction qui, étant donnés un point P, une longueur $long$, un nombre de lignes l et un nombre de colonnes c, affiche l lignes contenant le dessin en croix c fois.
   Par exemple, si $l = 3$ et $c = 4$, la fonction affiche :



   Le point P est le point situé le plus en bas à gauche dans le dessin et $long$ est la longueur d'un côté d'un petit carré. Vous utiliserez la fonction écrite dans la question 1.

Exercice 2 *Triangle*
Écrire une fonction : `void mon_draw_triangle(POINT P1, POINT P2, POINT P3, COULEUR C);`
qui prend en argument trois points et qui affiche un triangle non rempli de la couleur passée en argument.

Écrire une fonction : `void mon_draw_triangle_from_clic(COULEUR C);`
qui attend 3 clics et qui affiche un triangle non rempli de la couleur passée en argument.

Points importants à bien comprendre :
— Les deux fonctions ne font pas exactement la même chose, la première fonction prend des points en argument, il est donc inutile et sans intérêt de faire des `wait_clic()` dans la fonction.
— La deuxième peut être écrite en utilisant la première

— Dans la fonction `main` faites une boucle qui appelle dix fois la première fonction, les points passés en arguments étant issus de `wait_clic()`.
— Dans la fonction `main` rajouter une boucle qui appelle dix fois la deuxième fonction.

1.2 Premiers rebonds

Exercice 3 *J'efface, j'affiche*
Écrire une fonction :
`void efface_affiche(POINT ancien, POINT nouveau);`
qui affiche un cercle noir de rayon 25 centré sur le point `ancien` et qui affiche un cercle rouge de rayon 25 centré sur le point `nouveau`.

Exercice 4 *J'affiche, j'affiche pas*
On considère le programme suivant (déjà en place dans la machine virtuelle ou à récupérer sur e-campus (`exo3.c`) :

```
#include "graphics.h"

int main()
{
POINT p;

init_graphics(900,900);

// Par défaut on est en "affiche_auto_on"
p = wait_clic();
draw_fill_circle(p,200,bleu);
p = wait_clic();
draw_fill_circle(p,200,rouge);


// On passe en off
affiche_auto_off();
p = wait_clic();
draw_fill_circle(p,200,vert);
p = wait_clic();
draw_fill_circle(p,200,jaune);
// Il faut faire affiche_all pour que ca s'affiche
affiche_all();
// On est toujours en off
p = wait_clic();
draw_fill_circle(p,200,gris);
p = wait_clic();
draw_fill_circle(p,200,orange);
// Il faut faire affiche_all pour que ca s'affiche
affiche_all();

// On passe en on
// Plus besoin d'affiche_all
affiche_auto_on();
```

```
p = wait_clic();
draw_fill_circle(p,200,cyan);
p = wait_clic();
draw_fill_circle(p,200,magenta);


wait_escape();
exit(0);
}
```

1. Comprenez le lien entre le code et ce qui se passe lors de l'exécution.

Exercice 5 *Ça bouge*
Dans la fonction `main` simuler le déplacement d'un cercle en faisant des appels successifs à la fonction `efface_affiche`.

- Le cercle doit se déplacer de 2 en x et de 3 en y à chaque itération.
- Au départ le cercle est centré en $(50, 50)$.
- Utilisez les fonctions `void affiche_auto_off();` et `affiche_all();` (voir la section A.2 pour avoir un affichage plus fluide).

Exercice 6 *Pareil en mieux*
Écrire une fonction : `POINT deplace_balle(POINT ancien, int dx, int dy);`
qui prend en argument un point et renvoie le point déplacé de `dx` en `x` et de `dy` en `y`.

Réécrire la fonction `main` avec cette nouvelle fonction.

Exercice 7 *Encore plus propre*
Écrire une fonction : `POINT init_balle(int x, int y);`
qui prend en argument des coordonnées et renvoie un point ayant ces coordonnées.

Réécrire la fonction `main` avec cette nouvelle fonction.

Exercice 8 *Sortir c'est rebondir*
Écrire une fonction : `int trop_haut (POINT P, int r);`
qui prend en argument un point et un rayon et qui renvoie 1 si le cercle centré en `P` et de rayon `r` dépasse en haut de l'écran et 0 sinon.

Réécrire la fonction `main` avec cette nouvelle fonction en testant à chaque affichage si la balle dépasse. Si elle dépasse, modifier le déplacement en y qui devra alors être de -2.

2 TD 02 – Tableaux et Dégradés

Le but de ce TD est d'afficher des dégradés de gris puis de couleur. On rappelle que la fonction :
`COULEUR couleur_RGB(int r, int g, int b)`
prend en argument trois entiers dans l'intervalle [0..256[et renvoie la couleur correspondante.

Exercice 9 *Déclaration*
Déclarer en variable globale un tableau `T1` à une dimension de taille 256 et contenant des cases de type `COULEUR`.

Exercice 10 *Remplissage du tableau*
Écrire une fonction :
`void remplir_gris()`
qui remplit la $i^{\text{ème}}$ case de `T1` avec la couleur qui a comme valeur $(r, g, b) = (i, i, i)$.

Exercice 11 *Affichage du tableau*
Écrire une fonction :
`void afficher_horizontal()`
qui pour chaque case i du tableau affiche un rectangle de largeur 2 et de hauteur la hauteur de la fenêtre. Le coin en bas à gauche du rectangle de la case i est aux coordonnées $(2 \times i, 0)$.

Exercice 12 *Enchaînement 1*
Écrire le programme principal qui initialise la fenêtre graphique de taille 512×512 puis qui appelle la fonction `remplir` puis la fonction `afficher`. N'oubliez pas `wait_escape()`.

Exercice 13 *Enchaînement 2*
Écrire une fonction :
`void afficher_vertical()`
qui affiche le tableau à la verticale.

Exercice 14 *En rouge*
Écrire une fonction :
`void remplir_rouge()`
qui remplit la $i^{\text{ème}}$ case de `T1` avec la couleur qui a comme valeur $(r, g, b) = (i, 0, 0)$.
Afficher ce tableau.

Exercice 15 *En 2D*
Définir un tableau `T2` à deux dimensions chacune de taille 512. Écrire une fonction :
`void remplir_rouge_bleu()`
qui remplit la case (i, j) avec la couleur $(r, g, b) = (i, 0, j)$.
Afficher le tableau en deux dimensions.

3 TD 03 et TD 04 – Balles et raquettes

Le but de ce TD est de comprendre la notion de `struct` *et de bien comprendre les passages d'arguments et retour de fonction.*

Exercice 16 *Avec un struct c'est plus propre*
Déclarer le type :

```
struct balle
       {
       POINT centre;
       int rayon;
       COULEUR coul;
       int dx, dy;
       };
```

Déclarer le synonyme de type :

```
typedef struct balle BALLE;
```

Écrire les fonctions :
- `BALLE init_balle ();` qui initialise une balle avec des valeurs qui vous semblent pertinentes.
- `BALLE deplace_balle (BALLE B);` qui déplace la balle en ajoutant aux champs `centre.x` et `centre.y` les valeurs des champs `dx` et `dy`.
- `void efface_balle (BALLE B);` qui efface la balle.
- `void affiche_balle (BALLE B);` qui affiche la balle.
- `BALLE rebond_balle (BALLE B);` qui modifie les champs `dx` et `dy` si la balle dépasse d'un bord.

3.1 Tableau de balles

Exercice 17 *Tableau de balles*

- Initialiser une fenêtre de 400×400.
- Déclarer un tableau de balles (la définition du type `BALLE` est la même que pour l'exercice précédent).
- L'initialisation de valeurs pour chaque balle est aléatoire :
  - position : aléatoire entre 100 et 300 en `x` et en `y`.
  - couleur : aléatoire.
  - rayon : aléatoire entre 5 et 30.
  - dx et dy : aléatoire entre -5 et $+5$ mais non nuls.
- Écrire un programme principal qui fait se déplacer $N = 100$ balles.
- Ajouter un mur vertical sans épaisseur, sur toute la hauteur, centré de tel sorte que les balles rebondissent sur ce mur.

3.2 Pong

Exercice 18 *Déclaration de type*
On veut pouvoir stocker une raquette rectangulaire comme dans un casse-brique élémentaire. La raquette ne se déplace pas toute seule, elle n'a donc pas besoin de stocker d'informations de déplacement comme le type `BALLE`.

Déclarer un type `RAQUETTE` en commentant l'usage de chacun des champs.

Exercice 19 *Effacer afficher la raquette*
Écrire deux fonctions, une qui affiche et une qui efface la raquette.

Exercice 20 *La fonction `get_arrow()` de la bibliothèque `graphics.h`*
Lorsque la fonction `get_arrow()` est appelée, cette fonction renvoie un `POINT`, donc un appel correct est :

```
POINT P;
...
P = get_arrow();
```

- Si l'utilisateur a appuyé une fois sur la flèche gauche, le champ `x` du point `P` vaudra -1.
- Si l'utilisateur a appuyé n fois sur la flèche gauche, le champ `x` du point `P` vaudra $-n$.
- Si l'utilisateur a appuyé une fois sur la flèche droite, le champ `x` du point `P` vaudra $+1$.
- Si l'utilisateur a appuyé n fois sur la flèche droite, le champ `x` du point `P` vaudra $+n$.

— Si l'utilisateur a appuyé ni sur la flèche droite ni sur la flèche gauche, le champ `x` du point `P` contiendra 0.

La fonction n'est pas bloquante, c'est à dire qu'elle n'attend pas que l'utilisateur appuie sur une flèche, elle renvoie juste quelle flèche a été appuyée ou pas.

Écrire une fonction `RAQUETTE deplace_raquette(RAQUETTE R, int dx)` qui déplace la raquette en fonction du `dx` passé en argument.

Écrire un programme principal qui fait se déplacer la raquette, chaque appui sur une flêche faisant se déplacer la raquette de -1 ou $+1$.

Exercice 21 *Jolie raquette*

Modifier le déplacement de la raquette pour que :

1. elle soit plus rapide.
2. elle ne puisse pas sortir de l'écran.

Exercice 22 *Balle et Raquette*

Reprenez le code des exercices précédents pour qu'une balle et la raquette s'affichent en même temps. La balle doit rebondir et la raquette doit se déplacer avec les flèches.

Exercice 23 *Choc balle raquette*

Écrire une fonction `choc` qui prend en argument une balle et une raquette et qui renvoie 1 si le cercle de la balle et le rectangle de la raquette ont une intersection et qui renvoie 0 sinon.

En reprenant le code qui a été écrit dans les exercices précédents, faites rebondir la balle sur la raquette.

Exercice 24 *Le jeu de base*

Programmer le jeu pour qu'il respecte les règles suivantes :

— Au départ la balle va vers le haut.
— Au départ le joueur dispose de 3 vies.
— La balle ne rebondit pas en bas mais disparait et dans ce cas, le joueur perd une vie.
— Chaque rebond de la balle sur la raquette fait marquer un point.
— Quand le joueur n'a plus de vie, le jeu se termine.

En utilisant la fonction :

```
void aff_int(int n, int taille, POINT p, COULEUR C);
```

qui affiche l'entier `n` de la taille de police `taille`, positionné en `p` et de couleur `C`
afficher le nombres de vies restantes et les points du joueur de manière jolie.

Exercice 25 *Options de jeu*

Implémenter ces differentes options

1. Au bout de 10 rebonds avec la même balle, une vie de plus.
2. Au bout de 20 rebonds avec la même balle, deux nouvelles balles apparaissent pour jouer simultanément avec 3 balles
3. À chaque rebond la balle accélère
4. À chaque rebond sur la raquette, la raquette diminue de taille (sans jamais être nulle)
5. À chaque rebond sur la raquette, le plafond descend.
6. À chaque rebond la balle a son rayon qui diminue
7. Inventez une option et programmez là.

4 TD 05 et 06 – Horloge Analogique et Digitale

Vous allez programmer une horloge analogique et une horloge digitale. Dans une fenêtre de taille 900×600, vous partagerez votre écran en deux parties : le tiers du bas servira à afficher l'horloge digitale, les deux tiers du haut, l'horloge analogique.

Votre programme consistera en une boucle infinie qui réaffichera les 2 horloges à intervalles réguliers.

4.1 Horloge analogique

1. Ecrire une fonction `void afficher_cadran()` qui affiche le cadran de votre horloge analogique. Dans cette fonction vous ferez apparaitre les 12 repères de de votre horloge avec leurs chiffres. Il existe les fonctions `sinf()` (sinus) et `cosf()` (cosinus) qui prenne en paramètre un angle en radians et renvoie la valeur du sinus ou du cosinus de cet angle. Tester votre fonction.
2. Ecrire une fonction `void afficher_aiguille_heure(int h, int m, int s)` qui affiche la petite aiguille des heures en fonction de `h, m` et `s` sur le cadran. Tester votre fonction.
3. Ecrire une fonction `void afficher_aiguille_minute(int h, int m, int s)` qui affiche la grande aiguille des minutes en fonction de la valeur de `h, m` et `s` sur le cadran. Tester votre fonction.
4. Ecrire une fonction `void afficher_aiguille_seconde(int h, int m, int s)` qui affiche l'aiguille des secondes en fonction de la valeur de `h, m` et `s` sur le cadran. Tester votre fonction.
5. Ecrire une fonction `void afficher_aiguilles(int h,int m,int s)` qui affichent les 3 aiguilles sur votre cadran. Tester votre fonction.
6. Le main de votre programme doit ressembler à cela.

```
int main()
{
   int h,m;


  while(1)
  {
     h = heure(); m = minute();s = seconde();
     afficher_cadran();
     afficher_aiguilles(h,m,s);
   }
   return 1;
}
```

4.2 Horloge Digitale.

Dans cette partie, vous devez afficher la même heure que pour l'horloge analogique mais sous forme digitale c'est à dire : hh :mm :ss

Chaque chiffre de cette horloge sera représenté comme des segments allumés parmi les 7 du diagramme suivant :

1. Ecrire une fonction `void transforme(int a, int segments[7])` qui pour un entier `a` entre 0 et 9, remplit le tableau `segments` avec des `0` pour les segments éteints et des `1` pour les segments allumés.
2. Ecrire les fonctions :

(a) `void afficher_dizaine_heure(int segments[7])`

(b) `void afficher_unite_heure(int segments[7])`

(c) `void afficher_dizaine_minute(int segments[7])`

(d) `void afficher_unite_minute(int segments[7])`

(e) `void afficher_dizaine_seconde(int segments[7])`

(f) `void afficher_unite_seconde(int segments[7])`

et les tester une par une.

3. Ecrire une fonction `void deux_points()` qui affiche et éteint des " :" clignotants entre les heures et les minutes et entre les minutes et les secondes.
4. Ecrire une fonction `void affiche_digitale(int h, int m, int s)` qui affiche l'heure sous forme analogique sur votre écran.
5. Modifier votre `main` pour qu'il appelle cette fonction et affiche vos deux horloges simultanément.

4.3 Chronomètre

Nous allons rajouter une fonctionnalité à cette horloge : un chronomètre. Pour cela, vous allez rajouter un bouton dans la partie digitale qui va permettre de passer de la fonctionnalité "horloge" à la fonctionnalité chronomètre. Pour bien utiliser cette fonctionnalité nous allons devoir utiliser la structure de données suivante :

```
struct chrono {
     int h;
     int m;
     int s;
};

typedef struct chrono CHRONO;
```

Vous écrirez pour cela les fonctions suivantes :

1. `void afficher_bouton()` qui permet de passer d'un mode "horloge" à un mode "chronomètre".
2. `CHRONO init_chronomètre()` qui initialise un chronomètre à 0 ;
3. `void afficher_chrono(CHRONO c)` qui affiche sous forme digitale la valeur d'un chronomètre.
4. Lorsque vous êtes en mode "chronomètre", vous allez afficher 2 boutons supplémentaires un bouton `START/STOP` qui permet de démarrer et d'arrêter un chronomètre et un bouton `TOUR` qui permet d'afficher pendant 5 secondes le temps d'un tour.

A Annexe : types, variables constantes et fonctions disponibles

A.1 Types, Variables, Constantes

Types

```
typedef struct point int x,y; POINT;
typedef Uint32 COULEUR;
typedef int BOOL;
```

Variables

La largeur et la hauteur de la fenêtre graphique :

```
int WIDTH;
int HEIGHT;
```

Ces deux variables sont initialisées lors de l'appel à `init_graphics()`.

Constantes

Déplacement minimal lorsque l'on utilise les flèches :

```
#define MINDEP 1
```

Les constantes de couleur :

```
#define noir    0x000000
#define gris    0x777777
#define blanc   0xffffff
#define rouge   0xff0000
#define vert    0x00ff00
#define bleu    0x0000ff
#define jaune   0x00ffff
#define cyan    0xffff00
#define magenta 0xff00ff
```

Les constantes booléennes :

```
#define TRUE 1
#define True 1
#define true 1
#define FALSE 0
#define False 0
#define false 0
```

A.2 Affichage

Initialisation de la fenêtre graphique

```
void init_graphics(int W, int H);
```

Affichage automatique ou manuel

Sur les ordinateurs lent, il vaut mieux ne pas afficher les objets un par un, mais les afficher quand c'est nécessaire. c'est aussi utilme pour avoir un affichage plus fluide quand on fait des animations.

On a donc deux modes d'affichage, automatique ou non. Quand l'affichage automatique est activé, chaque dessin d'objet l'affiche automatiquement. Quand il n'est pas automatique c'est l'appel à la fonction `affiche_all();` qui affiche les objets.

Pour basculer de l'affichage automatique au non automatique, il y a deux fonctions :

```
void affiche_auto_on();
void affiche_auto_off();
```

Quand on est en mode non automatique l'affichage se fait lorsque l'on appelle la fonction :
`void affiche_all();`
Par défaut on est en mode automatique.

Création de couleur

`COULEUR couleur_RGB(int r, int g, int b);` prend en argument les 3 composantes rouge (`r`), vert (`g`) et bleue (`b`) qui doivent être comprises dans l'intervalle : [0..255].

A.3 Gestion d'événements clavier ou souris

Gestion des flêches

`POINT get_arrow();`
Si depuis le dernier appel à `get_arrow()`, il y a eu G appuis sur la flèche gauche, D appuis sur la flèche droite H appuis sur la flèche haut et B appuis sur la flèche bas.
Le point renvoyé vaudra en `x` $D - B$ et en `y` $H - B$.
Cette instruction est non bloquante, c'est à dire que si aucune flèche n'a été appuyée les champs x et y vaudront 0.

Gestion des déplacements la souris

`POINT get_mouse();` renvoie le déplacement de souris avec la même sémantique que `get_arrow()`. Cette instruction est non bloquante : si la souris n'a pas bougé les champs x et y vaudront 0.

Gestion des clics de la souris

`POINT wait_clic();` attend que l'utilisateur clique avec le bouton gauche de la souris et renvoie les coordonnées du point cliqué. Cette instruction est bloquante.
`POINT wait_clic_GMD(char *button);` attend que l'utilisateur clique et renvoie dans `button` le bouton cliqué :
— `*button` vaut 'G' (pour Gauche) après un clic sur le boutton gauche,
— `*button` vaut 'M' (pour milieu) après un clic sur le boutton du milieu,
— `*button` vaut 'D' (pour Droit) après un clic sur le boutton droit.
Cette instruction est bloquante.

Fin de programme

`POINT wait_escape();` attend que l'on tape Echap et termine le programme.

A.4 Dessin d'objets

`void fill_screen(COULEUR color);` remplit tout l'écran.
`void pixel(POINT p, COULEUR color);` dessine un pixel.
`void draw_line(POINT p1, POINT p2, COULEUR color);` dessine un segment.
`void draw_rectangle(POINT p1, POINT p2, COULEUR color);` dessine un rectangle non rempli. Les deux points sont deux sommets quelconques non adjacents du rectangle
`void draw_fill_rectangle(POINT p1, POINT p2, COULEUR color);` dessine un rectangle rempli Les deux points sont deux sommets quelconques non adjacents du rectangle
`void draw_circle(POINT centre, int rayon, COULEUR color);` dessine un cercle non rempli.
`void draw_fill_circle(POINT centre, int rayon, COULEUR color);` dessine un cercle rempli.

```
void draw_circle_HD(POINT centre, int rayon, COULEUR color);
void draw_circle_BD(POINT centre, int rayon, COULEUR color);
void draw_circle_HG(POINT centre, int rayon, COULEUR color);
void draw_circle_BG(POINT centre, int rayon, COULEUR color);
```
dessinent des quarts de cercle.

`void draw_fill_ellipse(POINT F1, POINT F2, int r, COULEUR color);` dessine une ellipse remplie.

`void draw_triangle(POINT p1, POINT p2, POINT p3, COULEUR color);` dessine un triangle non rempli.

`void draw_fill_triangle(POINT p1, POINT p2, POINT p3, COULEUR color);` dessine un triangle rempli.

A.5 Écriture de texte

L'affichage de texte n'est pas en standard dans SDL. Il faut donc que la librairie `SDL_ttf` soit installée.

La configuration fournie teste (grâce au `Makefile`) si `SDL_ttf` est installée ou pas. Si elle est installée, l'affichage se fait dans la fenêtre graphique sinon il se fait dans la fenêtre shell.

`void aff_pol(char *a_ecrire, int taille, POINT p, COULEUR C);` affiche du texte avec

- le texte est passé dans l'argument `a_ecrire`
- la police est celle définie par la constante `POLICE_NAME` dans `graphics.c`
- la taille est passée en argument
- l'argument p de type POINT est le point en haut à gauche à partir duquel le texte s'affiche
- la COULEUR C passée en argument est la couleur d'affichage

`void aff_int(int n, int taille, POINT p, COULEUR C);` affiche un entier. Même sémantique que `aff_pol()`.

Les fonctions suivantes affichent dans la fenêtre graphique comme dans une fenêtre shell. Commence en haut et se termine en bas :

`void write_text(char *a_ecrire);` écrit la chaîne de caractère passée en argument.

`void write_int(int n);` écrit l'entier passé en argument.

`void write_bool(BOOL b);` écrit le booléen passé en argument.

`void writeln();` renvoie à la ligne.

A.6 Lecture d'entier

`int lire_entier_clavier();` renvoie l'entier tapé au clavier. Cette fonction est bloquante

A.7 Gestion du temps

Chronomètre élémentaire

Ce chronomètre est précis à la microseconde.

`void chrono_start();` déclenche le chrono. Le remet à zéro s'il était déjà lancé.

`float chrono_val();` renvoie la valeur du chrono et ne l'arrête pas.

Attendre

`void attendre(int millisecondes);` attend le nombre de millisecondes passé en argument.

L'heure

`int heure();` envoie l'heure de l'heure courante.
`int minute();` renvoie le nombre de minutes de l'heure courante
`int seconde();` renvoie le nombre de secondes de l'heure courante.

A.8 Valeur aléatoires

`float alea_float();` renvoie un `float` dans l'intervalle $[0; 1[$.
`int alea_int(int N);` renvoie un `int` dans l'intervalle $[0..N[$ soit N valeurs différentes de 0 à $N-1$.

A.9 Divers

`int distance(POINT P1, POINT P2);` renvoie la distance entre deux points.