Stéph

Zeste de savoir

IOCCC - Décortiquons le code de Peter EASTMAN, vainqueur Best Ball 2011.

16 janvier 2021

Table des matières

Pour les fans du langage dont s'agit ou les simples curieux, vous avez du entendre parler du *IOCCC*. Comme l'idée est d'aller rapidement au cœur du sujet dans ce billet, je renvoie ceux désireux d'en savoir plus, à faire un tour sur la page dédiée chez WIKIPEDIA ou de visiter le site directement.

Bon vous l'avez compris on va parler `C` avec pour objectif de comprendre et d'appendre. Le but recherché sera tout d'abord de décortiquer le code source qui est volontairement écrit sans autre formatage ou indentation que le dessin ou le symbole que son auteur a voulu représenter. Nous en ferons au fur et à mesure un code lisible pou un humain en expliquant les parties qui méritent des éclaircissements (presque tout en fait).

Ce billet peut être lu et compris par les pratiquants de ce langage (et tous les autres, bien entendu), mais soyons réaliste, les débutants risquent de rencontrer quelques difficultés. Les explications ne leur sont pas destinés et je ne serai pas très didactique.

1. Le source d'origine

J'ai choisi donc de vous parler du programme du vainqueur 2011, dans la section *Best Ball*, *Peter Eastman*. Les plus avides pourront tout de suite se jeter sur la vidéo qui montre le résultat en version animée (pardonnez la qualité, c'est une capture et c'est pas mon truc).

Je leur recommande cependant un peu de patience et la lecture du source au préalable.

Voir l'animation:

Contenu masqué n°1

Voici le code tel que Eastman l'a présenté:

```
#include <stdio.h>
#include <math.h>
#include <unistd.h>
#include <sys/ioctl.h>

            main() {
         short a[4];ioctl
      (0,TIOCGWINSZ,&a);int
    b,c,d=*a,e=a[1];float f,g,
  h,i=d/2+d%2+1,j=d/5-1,k=0,l=e/
 2,m=d/4,n=.01*e,o=0,p=.1;while (
printf("\x1b[H\x1B[?25l"),!usleep(
79383)){for (b=c=0;h=2*(m-c)/i,f=-
.3*(g=(l-b)/i)+.954*h,c<d;c+=(b=++
b%e)==0)printf("\x1B[%dm ",g*g>1-h
*h?c>d-j?b<d-c||d-c>e-b?40:100:b<j
||b>e-j?40:g*(g+.6)+.09+h*h<1?100:
 47:((int)(9-k+(.954*g+.3*h)/sqrt
  (1-f*f))+(int)(2+f*2))%2==0?107
    :101);k+=p,m+=o,o=m>d-2*j?
      -.04*d:o+.002*d;n=(l+=
         n)<i||l>e-i?p=-p
             ,-n:n;}}
```

Violent, vu comme cela, mais il montre qu'il est bien dans le sujet, c'est certainement une balle ou un ballon.

Avant de comprendre un tel charabia, mais après une lecture attentive tout de même, je vous invite à le compiler tel quel. Récompense pour ceux qui ont su attendre un peu avant de visionner la vidéo. Que voient-ils: une console qui affiche une balle, de deux couleurs, bondissante et rotative, suivie par son ombre. Sachez que cela tient en 655 octets dans le `main`, il a fait compact, le gars.

En voici le résultat:

Vous, je sais pas mais moi, comprendre, en détail, comment ce code fonctionne en lisant ce que contient cette sphère de caractères, je n'y arrive pas. Nous allons donc éclaircir tout cela. Vous avez sans doute une idée de la méthode qu'il convient d'appliquer:

— repérons les points virgules afin de passer à la ligne;
— isoler le code entre accolades;
— indenter pour faire clair;
— renommer au besoin (et franchement il faudra) tout ce qui est abscons;
— insérer quelques manques aux bonnes règles.

Mes oreilles sifflent déjà au sujet de l'indentation. Pas de débat possible, je fais comme je veux pour la démonstration.

Après chacun fait comme il aime.

2. Premier formatage

2.1. Premier toilettage.

Le premier jet donne:

```
#include <stdio.h>
#include <math.h>
#include <unistd.h>
#include <sys/ioctl.h>

main()
{
        short a[4];
        ioctl(0,TIOCGWINSZ,&a);
        int b,c,d=*a,e=a[1];
        float f,g,h,i=d/2+d%2+1,j=d/5-
            1,k=0,l=e/2,m=d/4,n=.01*e,o=0,p=.1;

        while (printf("\x1b[H\x1B[?25l"),!usleep(79383))
        {
                for (b=c=0;
                h=2*(m-c)/i,f=-.3*(g=(l-b)/i)+.954*h,c<d;
                c+=(b=++b%e)==0)
                printf("\x1B[%dm ",g*g>1-h
                                *h?c>d-j?b<d-c||d-c>e-b?40:100:b<j
                                ||b>e-j?40:g*(g+.6)+.09+h*h<1?100:
                                47:((int)(9-k+(.954*g+.3*h)/sqrt
                                (1-f*f))+(int)(2+f*2))%2==0?107
                                :101);
                k+=p,m+=o,o=m>d-2*j?-.04*d:o+.002*d;
                n=(l+=n)<i||l>e-i?p=-p,-n:n;
        }
}
```

Partant de là, même si on est encore loin du compte, on peut repérer plus facilement:

— les variables;
— la présence de `TIOCGWINSZ` (la console n'est pas loin);

3. Explorons le while().

— un `while()` "musclé" (qui sent un peu la boucle infinie) contenant un `for()` tout aussi balèze;
— ce `for()` contient un `printf()` qui passe son temps à modifier un tas de truc;

Que conclure à cette étape: certainement qu'on affiche quelque chose dans le terminal.

Pour aller plus loin, mieux vaut réviser un peu les appels système. En effet, cela n'a pas échappé aux plus aguerris un `ioctl (0,TIOCGWINSZ,&a)` n'est pas utilisé tous les jours.

C'est le premier morceau sérieux auquel il faut s'attaquer, c'est parti:

— `ioctl` permet de travailler sur les fichiers ouverts et les caractères particuliers. Grâce à la page `man` on apprend notamment:
— que le premier paramètre (`0`) est le descripteur. Il s'agit de l'entrée standard;
— que le deuxième (`TIOCGWINSZ`) nous permet de connaître la taille du terminal;
— que le dernier (`(&a)`) s'apparente à un pointeur sur une structure.

Très bien et on en fait quoi de tout cela? Allons voir ladite structure avant:

```
1 struct winsize {
2         unsigned short ws_row ;
3         unsigned short ws_col ;
4         unsigned short ws_xpixel ;
5         unsigned short ws ypixel ;
6         };
```

Voici donc que l'on passe un pointeur sur un tableau de short (`&a` qui vaut `a`). On aura donc dans notre appel à `ioctl` la hauteur du terminal et sa largeur.

Bon çà, c'est fait. Il faut regarder un peu comment est utilisé `a`. Et bien elle donne une valeur aux variables `d` et `e` et … c'est tout.

Cela nous éclaire sur notre terminal et nous permet de renommer nos variables en hauteur `d` et largeur `e` que l'on vient d'identifier.

Poursuivons le renommage.

3. Explorons le while().

3.1. Jetons un coup d'oeil à la boucle principale.

Continuons à faire propre, renommer, compléter les manques et rendre cela plus clair.

Rajoutons un `int` et `return` au `main` et occupons nous de la boucle `while()`. Nous en profitons pour la définir officiellement comme infinie (ce qu'elle était) puis reportons le `printf()`. Comment l'auteur a procédé pour cette boucle infinie?

L'expression testée contient le `printf()`, une virgule et l'appel à `usleep()`. Rappelons que l'opérateur virgule évalue le membre de gauche en ignorant son résultat puis évalue son membre de droite et en renvoie sa valeur. Nous avons donc dans l'ordre:

3. Explorons le while().

— exécution du `printf()` ;
— exécution de `usleep(),` celle-ci renvoyant zéro;
— l'opérateur `!` remplace ce dernier par 1.

Le test du `while()` vaut donc toujours vrai ce qui en fait une boucle infinie. La valeur donnée à `usleep()` pourrait être de 80 000 (se sont des millisecondes) pour le rendre plus confortable visuellement, cela n'a pas d'importance à ce niveau.

Cela nous donne:

```
1  int main()
2  {
3          short a[4];
4          ioctl(0,TIOCGWINSZ,&a);
5          int b,c;
6          int hauteur = a[0];
7          int largeur = a[1];
8          float f,g,h,i=hauteur/2+hauteur%2+1,j=hauteur/5-
               1,k=0,l=largeur/2;
9          float m=hauteur/4,n=.01*largeur,o=0,p=.1;
10
11         while (1)
12         {
13                 printf("\x1b[H\x1B[?25l");
14                 usleep(79383));
15
16                 for (b=c=0;
17                 h=2*(m-c)/i,f=-.3*(g=(l-b)/i)+.954*h,c<hauteur;
18                 c+=(b=++b%largeur)==0)
19                         printf("\x1B[%dm ",g*g>1-h
20                                 *h?c>hauteur -j?b<hauteur
                                       c||hauteur-c>largeur-
                                       b?40:100:b<j
21                                 ||b>largeur-
                                       j?40:g*(g+.6)+.09+h*h<1?100:
22                                 47:((int)(9-k+(.954*g+.3*h)/sqrt
23                                 (1-f*f))+(int)(2+f*2))%2==0?107
24                                 :101);
25                 k+=p,m+=o,o=m>hauteur-2*j?-
                       .04*hauteur:o+.002*hauteur;
26                 n=(l+=n)<i||l>largeur-i?p=-p,-n:n;
27         }
28         return 0;
29 }
```

Sautons le premier `printf()` pour le moment et voyons maintenant notre `for()`.

4. Examinons le for ().

4.1. Que nous dit ce for().

Observons bien la boucle, nous constatons que nous y retrouvons l'opérateur virgule par trois fois, séparant trois expressions. Ne traitant pas les membres de gauche intéressons nous à la dernière. Le test se fait donc sur `c < hauteur` alors que les expressions précédentes servent aux affectations. Partant de ce constat on peut les ramener au début du `for()` ce qui donne:

```
for (y = 0; y < hauteur ; y++)
{
        for (x= 0; x < largeur; x++)
        {
                g = (1 - x) / i;
                h = 2 * (m - y) / i;
                f = -.3 * g + .94 * h;
                printf ("../...");
        }
}
```

Vous commencez sans doute à y voir un peu mieux sans doute. Alors trouvez la bonne réponse parmi ces trois propositions: Le programme déclenche:

1. Une boucle contenant une autre boucle définissant l'emplacement du terminal en y affichant un truc;
2. Une boucle infinie contenant deux boucles définissant chacun des emplacements du terminal et en y affichant un truc à chaque fois;
3. Une boucle infinie contenant trois boucles pour afficher le terminal et un truc dedans.

Contenu masqué n°2

Poursuivons l'analyse et remontons au premier `printf()` qui nous parle certainement de couleurs et de caractères particuliers.

5. Parlons un peu couleurs.

5.1. Voici un gros morceau.

Il débute par `\x1B[`. Mais c'est quoi ce truc?

J'espère que vous avez révisé votre table *ASCII* et surtout que vous maîtrisez bien le terminal Linux. Sinon, il faut s'y replonger pour tout saisir. Allez, je vous aide.

Procédons par ordre:

— `\x` annonce un caractère dont la valeur *ASCII* sera hexadécimale;
— `1B` correspond caractère de contrôle "escape";
— Oui mais suivi du `\[` il commence une séquence d'échappement.

Oui mais pour faire quoi? Contrôler le terminal (quelle merveille, tout çà en cinq caractères).

Les séquences de contrôle se placent après le crochet et permettront donc d'agir sur le terminal (effacements, déplacements …). Je laisse à ceux que cela intéresse d'explorer plus avant toutes ces possibilités.

Poursuivons donc après le crochet: `H\x1B[?25l`.

— `H` (sans paramètres) déplace le curseur au haut à gauche;
— `?25L` le cache.

Ce `printf()` sert donc à cacher le curseur.

Le second se charge de la couleur. Attention çà pique!

Il faut avant tout apprendre à coder ces séquences, voici donc quelques explications:

— Commencez par ouvrir la séquence avec le `\x1B[`;
— Renseignez ensuite la lettre référençant l'action;
— Inscrivez **AVANT** cette lettre ses paramètres éventuels;
— Chaque paramètre doit être séparé par un point virgule.

Notre second `printf()` ouvre une séquences avec `m`, lettre appliquant le changement de la couleur. Son paramètre est complexe et se termine par un espace. C'est ce dernier qui sera affiché en prenant la couleur demandée.

L'animation appelle le noir, le blanc, le rouge et deux nuances de gris, l'ensemble pour le premier plan et l'arrière plan. Les codes utilisés sont ceux de la spécification SGR-1 (4 bits) compris entre 0 et 128 (important pour la suite). Allez, je suis en forme, je vous donne la correspondance entre les numéros de couleurs et leur code *VGA* pour l'arrière plan:

— 40 -> 0,0,0;
— 47 -> 170,170,170;
— 100 -> 85,85,85;
— 101 -> 255,85,85;
— 107 -> 255,255,255.

Revenons au code pour voir comment cela s'articule. Notre séquence commence donc par le `\x1B[m` pour ensuite afficher le nombre correspondant au rendu souhaité (gras, couleur …). Comprenant mieux ce code abscons on peut rendre tout cela un peu plus clair et j'ai choisi ceci (extraits):

```
1 int main()
2 {
3         int i ;
4
5         for (i = 1; i < 128; i++)
6         {
7                 printf("\x1B[%dm", i) ;
```

```
 8                 printf("%d", i) ;
 9                 printf("\x1B[0m\n");
10         }
11         return 0;
12 }
```

Allons plus loin avec quelques `#define` pour ces couleurs.

```
 1 int main()
 2 {
 3         #define NOIR                40
 4         #define GRIS_CLAIR        47
 5         #define GRIS                100
 6         #define ROUGE                101
 7         #define BLANC                107
 8
 9         .../...
10
11         while (1)
12         {
13                 printf("\x1B[H");
14                 printf("\x1B[?251");
15
16                 for (x = 0; x < largeur; x++)
17                 {
18                         g = (1 - x) / i;
19                         h = 2 * (m - y) / i;
20                         f = -.3 * g + .954 * h;
21
22       int couleur = g * g > 1 - h *h ? y > hauteur - j  ? x <
             hauteur - y ||
23       hauteur - y > largeur - x ? NOIR : GRIS  : x < j ||
24       x > largeur - j ? NOIR : g * (g + .6) + .09 + h * h < 1 ?
             GRIS : GRIS_CLAIR :
25       (int ) 9 - k + (0954 * g + .3 * h) / sqrt (1 - f*f)) + (int)
             (2 + f * 2) %2 == 0 ? BLANC : ROUGE;
26       printf("\x1B[%dm ", couleur);
27                 }
28         }
29         .../...
30 }
```

Bien, ceci fait, nous pouvons voir l'usage intensif de l'opérateur ternaire. C'est bien entendu voulu par l'auteur afin de rendre son code compact et illisible. Afin de comprendre il faut repérer ses `:` et ses `?` afin de les isoler. Je choisis la solution de transformer tout cela en une suite de "if/else", ce qui nous donne:

```
.../...
        int couleur;

        if (g * g > 1 - h * h)
        {
                if (y > hauteur - j)
                {
                        if (x < hauteur - y || hauteur - y >
                            largeur - x )
                                couleur = NOIR ;
                        else
                                couleur = GRIS;
                }
                else
                {
                        if (x < j || x > largeur - j)
                                couleur = NOIR ;
                        else
                        {
                                if (g * (g+.6) + .09 + h* h < 1)
                                        couleur = GRIS ;
                                else
                                        couleur = GRIS_CLAIR ;
                        }
                }
        }
        else
        {
                if ((int) 9 - k + (.954 * g + .3 * h) / sqrt (1 - f
                    * f)
                        + (int) (2 + f * f) %2 == 0 )
                        couleur = BLANC ;
                else
                        couleur = ROUGE ;
        }
  .../...
```

On avance mais il reste du taf. Il faut notamment comprendre à quoi correspondent certaines variables qui sont encore des énigmes. Regarder de nouveau la vidéo et observez les couleurs. On constate que seule la sphère est rouge et blanche et les autres couleurs servent au reste. Oui c'est évident à l'image mais essentiel quant à la compréhension du code.

Vous l'avez sans doute compris, ce programme localise le curseur afin de savoir ce qu'il doit afficher, la sphère ou son environnement.

Examinons les variables y afférentes.

6. Paufinons quelques variables

6.1. Travaillons encore sur les variables.

Là il va falloir vous remémorer comment déterminer l'équation d'un cercle.

Si nous regardons le test de notre premier `if()`, et je vous rappelle que la place du curseur est la base du bon affichage, il est fort probable qu'il détermine si celui-ci est dans ce cercle ou non. Si nous voulons rendre ce test plus parlant nous pouvons le remplacer par `g * g + h* h > 1` dont il est l'équivalent.

Maintenant regardez les expressions correspondantes, soit: `((l-x)/i)2+(2*(m-y)/i)2 > 1`. Cela ne vous rappelle rien?

Jetons un coup d'œil à l'équation d'un cercle. Si par exemple il est de centre:

$$(ax, ay)$$

et de rayon:

$$r$$

je peux obtenir:

$$((ax - x)/r)2 + ((ay - y)/r)2 = 1$$

Par comparaison on identifie trois variables, le rayon `i` , l'abscisse `l` et l'ordonnée `m`. Voilà qui va faciliter leur renommage et améliorer la compréhension. Les plus affûtés vont comprendre seuls à quoi correspond `2*`. Pour les autres, voici une piste: pensez terminal, pensez caractère …

Pour `g` et `h`, je vous revoie quelques lignes plus haut ou la réponse a été donnée. Il s'agit des coordonnées du curseur, bien sûr, qui seront également renommées.

> *i* Et de cinq, çà avance bien.

Examinons le test du cinquième `if()`: `(g * (g+.6) + .09 + h* h < 1)` et réécrivez-le en guise d'exercice. Il est en effet possible d'arriver à cela: `(g+0.3)2+h2<1`.

Il permet de décaler le cercle par rapport à la sphère … hum serait-ce l'ombre qui pointe son nez? Mais oui, on duplique en décalant vers la droite de 30%.

Il reste encore que la cas de la variable `j`. Celle-ci est initialisée à `hauteur/5-1` sans jamais être modifiée. Elle est utilisée uniquement avec `hauteur` ou `largeur`. Elle sert donc de délimitation à l'affichage.

Réécrivons tout cela de manière plus compréhensible. Commençons à commenter l'essentiel.

```
1 while (1)
2 {
3         printf("\x1B[H");                          // curseur placé
4         printf("\x1B[?251");                 // invisibilité
5
```

```
6           for (y = 0; y < hauteur; y++)
7           {
8                   for (x = 0; x < largeur; x++)
9                   {
10                          int couleur;
11                          curseur_x = (centre_x - x) / rayon;
12                          curseur_y= 2*(centre_y - y) / rayon;
13                          f = .3 * curseur_x + .954 *curseur_y;
14
15                          if (curseur_x *curseur_x + curseur_y *
                                curseur_y > 1) // si on se trouve hors
                                de la sphère
16                          {
17                                  if (y > hauteur - bord) // si on se
                                        trouve en bas
18                                  {
19                                          if (x < hauteur - y ||
                                                hauteur - y > largeur -
                                                x) // si on est à
                                                gauche ou a droite
20                                          {
21                                                  couleur = NOIR;
22                                          }
23                                          else
24                                                  couleur = GRIS;
25                                  }
26                                  else // si on se trouve plus haut
27                                  {
28                                          if (x < bord || x > largeur
                                                - bord) // si on se
                                                trouve sur les côtés
29                                          {
30                                                  couleur = NOIR;
31                                          }
32                                          else // on se trouve dans
                                                la zone
33                                          {
34                                                  if (curseur_x+0.3)
                                                        *
                                                        (curseur_x+0.3)
                                                        + curseur_y *
                                                        curseur_y < 1)
35                  // si on se trouve dans l'ombre de la sphère
36                                                  {
37                                                          couleur =
                                                                GRIS;
38                                                  }
39                                                  else
40                                                          couleur =
                                                                GRIS_CLAIR;
```

```
41                                                  }
42                                          }
43                                  }
44                                  else // on se trouve dans la sphère on
                                        affiche alors blanc ou rouge
45                                  {
46                                          if
                                                (((int)(9-k+(.954*curseur_x+.3*curseur_y)
                                                / sqrt(1-f*f)) + (int)(2+f*2))
                                                % 2 == 0)
47                                                  couleur = BLANC;
48                                          else
49                                                  couleur = ROUGE;
50                                  }
51                                  printf("\x1B[%dm ", couleur);
52                          }
53                  }
54                  k+p= centre_y+=o,
                        o=centre_y>hauteur-2*bord?-.04*hauteur:o+.002*hauteur;
55                  n=(centre_x+=n)<rayon||centre_x>largeur-rayon?p=-p,-n:n;
56  }
```

Ceci fait nous pouvons nous intéresser maintenant à notre balle, ses couleurs et son mouvement.

7. Damier rouge et blanc de la balle

7.1. La balle, une histoire de couleurs.

Notre balle affiche un damier blanc et rouge. Où trouver le code correspondant à l'affichage de ces couleurs? Regardons une nouvelle fois notre animation. Nous pouvons percevoir qu'elle est inclinée, cela peut peut-être nous aider, même si nous verrons son déplacement plus tard. Nous n'avons pas regardé de prêt la variable `f`, il est temps de le faire. Observez les deux lignes suivantes:

```
f = -.3 * curseur_x + .954 * curseur_y;
```

et

```
if (((int)(9-k+(.954*curseur_x+.3*curseur_y) / sqrt(1-f*f)) + (int)(2+f*2))
% 2 == 0)
```

La variable `f` n'est testée que dans ce `if()` mais que traduisent les nombres 0.3 et 0.954? Souvenez vous, le programmeur veut de la compacité mais aussi rendre le code illisible. Il utilise donc certainement des nombres magiques qu'il faut comprendre.

En fait la somme de leur carré est égal à 1. Il peut s'agir des sinus et cosinus d'un angle. En tripatouillant un peu on trouve que 0.3 est le sinus de 17.5° et 0.954 le cosinus de … 17.5°. Tiens, cela ressemble à l'inclinaison de la balle, non?

Nous transformons donc cette valeur d'angle en constante.

Ces nombres sont utilisés ailleurs pour le calcul de notre variable et notamment pour l'abscisse et l'ordonnée du point obtenu pour permettre une rotation d'angle. Notre variable `f` sera renommée `rotation_y` et l'expression `(.954*curseur_x+.3*curseur_y)` par `rotation_x`.

Pas très facile à appréhender, il faut l'avouer. Reformulons en disant que `rotation_x` et `rotation_y` correspondent au point que nous voulons colorer seulement dans la balle inclinée de 17.5°, ce qui pourrait se traduire par:

```
#define ANGLE = 17.5 / 180 * M_PI
rotation_x = cos(ANGLE) * curseur_x + sin(ANGLE) * curseur_y;
rotation_y = sin(ANGLE) * curseur_x + cos(ANGLE) * curseur_y;
if (((int)(9 - k + rotation_x / sqrt(1 - rotation_y *
    rotation_y))+(int)(2 + rotation_y * 2)) %2 ==0)
```

Au fait, on ne travaille pas à plat, c'est important pour notre damier, mais sur une sphère. Nous allons donc utiliser des noms plus parlant et mieux adaptés en remplaçant `x` et `y` par `longitude` et `latitude`. Notre `rotation_y` (fournit entre -1 et 1) est plutôt sympa car il donne directement la latitude calculée de la manière suivante: `2 + rotation_y * 2` dont le résultat est compris entre 0 et 4.

La longitude nécessite un calcul plus étoffé. Plus on s'approche de son NORD et de son SUD et plus les cases se resserrent. Cette sphère empile des disques de rayon 1 à l'équateur et 0 aux pôles. Le rayon du disque ne dépend que de `rotation_y` qui vérifie `rayon_disque2 + rotation_y2 = 1`. Donc `rayon_disque` = `sqrt(1 - rotation_y * rotation_y)`. C'est pourquoi l'abscisse est de la forme: `(int)(9 - k + rotation_x / rayon_disque)`. Le `9` sert à disposer d'une valeur toujours positive et le `k` ayant une valeur changeant régulièrement permettant de délivrer le damier suivant les longitudes.

Nous approchons de la fin mais avant de lire notre nouveau code je vous propose maintenant de traiter du mouvement de notre balle.

8. Mouvement

8.1. L'animation

Vous l'avez deviné, le code restant concerne l'animation. Nous poursuivons tout d'abord la réécriture du code afin de le rendre plus clair:

```
.../...
k += p;
centre_y += o;
if (centre_y > hauteur - 2 * bord)
        o = -.04 * hauteur;
```

```
else
        o = o + .002 * hauteur;

centre_x += n;
if (centre_x < rayon || centre_x > largeur - rayon)
{
        p = -p;
        n = -n;
}
```

Nous renommons et comprenons mieux. En effet, nous savons que `k` participe à la rotation du damier. Sa valeur est modifiée par `p` que l'on nommera vitesse_rotation. Quant à `o` elle permet de modifier la position (verticale) de la balle et nous l'appellerons `vitesse_verticale`. Enfin, nous terminerons par transformer `n` en `vitesse_horizontale`.

Terminons donc cette partie:

```
.../...
offset_rotation += vitesse_rotation; //mise à jour de la rotation
centre_y += vitesse_verticale; // idem pour la position verticale
    de la balle
if (centre_y > hauteur - 2 * bord)
{        //si la balle est trop basse, la vitesse devient négative
    pour le rebond
        vitesse_verticale = vitesse_vertical = -0.04 * hauteur;
}
else
{        //la vitesse vers le bas augmente linéairement en simulant
    l'accélération de la gravité
        vitesse-verticale = vitesse_vertical + 0.002 * hauteur;
}

// Position horizontale
centrre_x += vitesse_horizontale;
if (centre_x < rayon || centre_x  > largeur - rayon)
{        // si impact sur un bord on inverse la vitesse de rotation
        vitesse_rotation = - vitesse_rotation;
        // ainsi que la vitesse horizontale
        vitesse_horizontale = - vitesse_horizontale;
}
```

On est pas mal, je crois. Il reste à finaliser en rassemblant nos morceaux:

Contenu masqué n°3

Bien entendu, le fait de chercher à rendre ce charabia compréhensible pour un humain en rallonge le code source. Il faut cependant admettre que sa compacité, si l'on fait abstraction des règles habituelles de nommage et de formatage, démontre la puissance du langage utilisé. Allez, j'invite les joueurs à faire plus court et plus léger (hors assembleur) dans un autre langage.

Bon courage, merci de m'avoir lu et clin d'œil à Taurre 🍊

Ce billet était long, j'en convient, bien plus que ne l'est le source écrit par *Peter EASTMAN*. Une preuve, s'il en était encore besoin, de la puissance du `C` et de la compacité du code qu'il est possible d'atteindre en l'utilisant. Nous l'avons vu, il permet aussi d'écrire du code imbuvable et illisible si on s'y amuse. En dehors d'une recherche de compacité extrême justifiée par un besoin spécifique, il n'est pas recommandé d'user d'artifices sans une maîtrise parfaite de l'outil.

Contenu masqué

Contenu masqué n°1

Élément externe (Video) —
Consultez cet élément à l'adresse `https://www.youtube.com/embed/TvYBPRr5HaI?feature=oembed`.

Retourner au texte.

Contenu masqué n°2

La bonne réponse = 2.

Retourner au texte.

Contenu masqué n°3

```
#include <stdio.h>
#include <math.h>
#include <unistd.h>
#include <sys/ioctl.h>

#define NOIR                40
#define GRIS_CLAIR        47
#define GRIS                100
```

Contenu masqué

```
#define ROUGE                101
#define BLANC                107
#define ANGLE       17.7 * M_PI / 180.0

int main()
{
        short a[4];
        ioctl(0, TIOCGWINSZ, &a);
        int x;
        int y;
        int hauteur = a[0];
        int largeur = a[1];
        float curseur_x;
        float curseur_y;
        float rayon = hauteur / 2 + hauteur % 2 + 1;
        float bord = hauteur / 5 - 1;
        float offset_rotation = 0;
        float centre_x = largeur / 2;
        float centre_y = hauteur / 4;
        float vitesse_horizontale = 0.01 * largeur;
        float vitesse_verticale = 0;
        float vitesse_rotation = 0.1;

        while (1)
        {
                printf("\x1b[H");
                printf("\x1B[?25l");
                usleep(80000);

                for (y = 0; y < hauteur; y++)
        {
            for (x = 0; x < largeur; x++)
            {
                int couleur;
                curseur_x = (centre_x - x) / rayon;
                curseur_y = 2 * (centre_y - y) / rayon;
                if (curseur_x *curseur_x + curseur_y * curseur_y >
                    1)
                {
                    if (y > hauteur - bord)
                    {
                        if (x < hauteur - y || hauteur - y >
                            largeur - x)
                        {
                            couleur = NOIR;
                        }
                        else
                        {
                            couleur = GRIS;
                        }
```

Contenu masqué

```
57                      }
58                      else
59                      {
60                          if (x < bord || x > largeur - bord)
61                          {
62                              couleur = NOIR;
63                          }
64                          else
65                          {
66                              if ((curseur_x + 0.3) * (curseur_x +
                                    0.3) + curseur_y * curseur_y < 1)
67                              {
68                                  couleur = GRIS;
69                              }
70                              else
71                              {
72                                  couleur = GRIS_CLAIR;
73                              }
74                          }
75                      }
76                  }
77                  else
78                  {
79                      float rotation_x = cos(ANGLE) * curseur_x +
                            sin(ANGLE) * curseur_y;
80                      float rotation_y = sin(ANGLE) * curseur_x +
                            cos(ANGLE) * curseur_y;
81                      float rayon_disque = sqrt(1 - rotation_y *
                            rotation_y);
82                      if (((int)(9 - offset_rotation + rotation_x /
                            rayon_disque) +
83                          (int)(2 + rotation_y * 2)) %2 ==0)
84                      {
85                          couleur = BLANC;
86                      }
87                      else
88                      {
89                          couleur = ROUGE;
90                      }
91                  }
92                  printf("\x1B[%dm ", couleur);
93              }
94          }
95
96          offset_rotation += vitesse_rotation;
97          centre_y += vitesse_verticale;
98          if (centre_y > hauteur - 2 * bord)
99          {
100             vitesse_verticale = vitesse_verticale = -0.04 *
                    hauteur;
```

Contenu masqué

```
101         }
102         else
103         {
104             vitesse_verticale = vitesse_verticale + 0.002 *
                    hauteur;
105         }
106
107         centre_x += vitesse_horizontale;
108         if (centre_x < rayon || centre_x  > largeur - rayon)
109         {
110             vitesse_rotation = - vitesse_rotation;
111             vitesse_horizontale = - vitesse_horizontale;
112         }
113     }
114         return 0;
115 }
```

Retourner au texte.