Objectif

Créer un jeu qui vous permettra d'amasser des **$$$$** et replonger dans votre dépendance au développement de logiciel, et de maîtriser le concept de tableaux multidimensionnels. L'objectif est aussi de vous initier aux concepts de type énuméré et de type structure. Ce travail devrait aussi vous faire réaliser l'importance de bien isoler, dans son code, l'input (les commandes) du programme, le traitement des règles du jeu et l'affichage (l'output) graphique.

Contexte de réalisation et de remise

Ce travail **individuel** sera effectué pendant les périodes de laboratoire. Les remises seront faites par LÉA à la date mentionnée pour chaque livrable.

Livrables

1. *C21-TP1-Nom-Prenom.cpp*
   - Un seul fichier contenant tout le code source. **Utiliser** le fichier ***C21-TP1-Base.cpp*** et **renommer** le. Vous devez absolument utiliser les définitions (enum, struct, const et variables) de ce fichier.

2. *C21-TP1-Nom-Prenom.exe*
   - Le programme.

Spécifications

Le jeu est composé d'un damier de 96 cases (8 lignes, 12 colonnes) sur lesquelles le joueur déplace un curseur jaune à l'aide des flèches du clavier. Chaque case du jeu est visible sous l'un ou l'autre des 4 aspects suivants: *case bleue*, *case dollars*, *case rose,* ou *case noire.* Au début, toutes les cases sont soit une *case noire*, soit une *case bleue*.

Lorsque le curseur se pose sur une case bleue, celle-ci change et devient une case rose ou bien une case *dollars* ($$$$). Cette case deviendra visible seulement quand le curseur aura bougé de nouveau. Quand une case *dollars* devient visible à la suite d'un premier passage du curseur, le joueur doit passer sur la case à nouveau afin d'amasser les $$$$ et gagner 1 point. La case *dollars* $$$$ deviendra alors une case rose. Finalement, si le curseur se pose sur une case rose celle-ci devient une case noire et il ne sera plus possible d'y repasser par la suite.

Remarque importante : Le positionnement initial du curseur, au début de la partie, n'est pas considéré comme un déplacement, et n'a donc aucun effet sur l'état de la case de départ.

Le jeu se termine quand les 15 points ont été ramassés ou si le curseur ne peut plus se déplacer. Vous devez détecter, immédiatement après un déplacement, si le curseur est bloqué.

Le fichier **C21-TP1-Thiboutôt-Alain.exe** représente le programme à réaliser. Il faut le reproduire fidèlement.

Spécifications techniques

1. Les touches valides sont les 8 flèches de déplacement (ex : ➔, ➔, ➔, etc.)

```
enum Arrowkeys // Code ascii décimal des touches fléchées du clavier
{
    up_left       = 71,
    up            = 72,
    up_right      = 73,
    left          = 75,
    right         = 77,
    down_left     = 79,
    down          = 80,
    down_right    = 81
};

using Ak = Arrowkeys; // un alias plus concis
```

Voir le fichier « **arrow keys.cpp** » pour un exemple.

2. L'ensemble des cases possibles du damier est spécifié par le type énuméré suivant :

```
enum Case { CO, CS, CD, CF, CV };

CO --> Case Ordinaire       bleue
CS --> Case Surprise        bleue
CD --> Case Dollars         verte
CF --> Case Fragile         rose
CV --> Case Vide            noire
```

3. Le damier et les cases initiales au début du jeu. Ce tableau doit incarner l'état du jeu en tout temps.

```
Case damier[LIG][COL] =
{
    { CO, CO, CO, CO, CO, CO, CO, CO, CV, CO, CO, CS },
    { CO, CO, CV, CV, CO, CO, CO, CO, CO, CV, CO, CV },
    { CO, CO, CV, CS, CV, CO, CO, CO, CO, CO, CV, CS },
    { CO, CO, CV, CS, CV, CO, CO, CV, CV, CO, CV, CS },
    { CS, CO, CV, CV, CV, CS, CV, CO, CV, CO, CV, CO },
    { CS, CO, CS, CS, CO, CS, CV, CS, CV, CO, CV, CO },
    { CS, CO, CS, CO, CO, CO, CV, CV, CV, CO, CV, CO },
    { CS, CS, CO, CO, CO, CO, CO, CO, CO, CO, CO, CO }
};
```

4. Listes des transformations possibles des cases du damier

```
Case futur[5] = { CF, CD, CF, CV, CV };   // ex: futur[CO] ==> CF

Deux chemins possibles de transformations :
   1) CO ==> CF ==> CV
   2) CS ==> CD ==> CF ==> CV
```

5. Le déplacement (Move) du curseur dans le tableau damier

```
struct LC            // ligne et colonne (l,c) d'une case du damier[l][c]
{
   size_t l, c;
};

struct Move          // coordonnées des 2 cases impliquées dans un déplacement
{
   LC from, to;      // (case from ==> case to)
};

Move m;
```

Au début du jeu, le curseur est positionné dans le coin supérieur gauche (m.from = {0,0}) du damier.

Un déplacement est autorisé si la case d'arrivée (m.to) est à l'intérieur des frontières du damier et que cette case est différente de CV (case vide).

6. Chaque case du damier doit être affichée avec un caractère spécial et une couleur

```
const size_t CASE_X = 6;       // largeur d'une case : min 2
const size_t CASE_Y = 3;       // hauteur d'une case : min 2

struct Style                   // le style pour l'affichage d'une case
{
   Color color; char c;
};

Style map[5] =                 // les styles pour chaque case
{
   { Color::blu, '\xB2' },     // ex: map[CO].c ==> '\xB2'
   { Color::blu, '\xB2' },
   { Color::grn, '\x24' },     // ex: map[CD].color ==> Color::grn
   { Color::pur, '\xB0' },
   { Color::blk, '\x00' }
};
```

Voir le fichier « **case en couleur.cpp** » pour un exemple.

7. Le curseur doit être affiché en jaune avec les caractères spéciaux suivants :

```
char cursor[3][3] =
{
   { '\xC9', '\xCB', '\xBB' },
   { '\xCC', '\xCE', '\xB9' },
   { '\xC8', '\xCA', '\xBC' }
};
```

Les 4 caractères de coins sont obligatoires et les autres servent au remplissage en fonction du nombre de colonnes (CASE_X) de du nombre lignes (CASE_Y) d'une case.

8. À l'affichage, les cases du jeu sont espacées par des colonnes et des lignes vides

```
const size_t SPACE_X = 2;   // colonnes vides entre les cases, 1 minimum
const size_t SPACE_Y = 1;   // lignes vides entre les cases, 1 minimum
```

9. À l'affichage, Le coin supérieur gauche du damier est à la position :

```
const size_t START_X = 10;  // x du coin supérieur gauche
const size_t START_Y = 5;   // y du coin supérieur gauche
```

10. Attention : il y a deux systèmes de coordonnées dans ce programme :

    1) Un système **logique** (l,c) pour accéder les cases dans le tableau damier

```
struct LC // une coordonnée dans le tableau damier[l][c]
{
   size_t l, c;
};
```

    2) Un système **graphique** (x,y) pour afficher les cases dans la console

```
struct XY // une coordonnée (coin supérieur gauche) d'une case dans la console
{
   size_t x, y;
};
```

    Calculs pour déterminer la position graphique (x,y) d'une case à partir de sa position logique (l,c) :

```
LC lc = {0,0}; // position logique

XY xy = { START_X + (lc.c * DELTA_X) , START_Y + (lc.l * DELTA_Y) };
```

11. Attention : chaque déplacement du curseur implique de rafraichir à la console seulement deux cases et non la totalité du damier. Seule la case de départ (m.from) qui vient d'être libéré par le curseur et le curseur au nouvel endroit (m.to) doivent être réaffichés.

12. Attention : à la suite d'un déplacement, le programme doit détecter si le curseur est désormais bloqué, auquel cas la partie est terminé. Si les 15 points sont accumulés, il s'agit tout de même d'une victoire.

Diagramme d'action (ébauche)

```
Afficher le damier

Initialiser la position de départ du curseur en : ligne = 0 et colonne = 0
Afficher le curseur

Faire

   Faire
      Lire Touche
   Tant que Touche != flèche

   Calculer la position d'arrivée du curseur avec la direction de la flèche et
   sa position de départ

   Si la position d'arrivée est règlementaire

      Vérifier si des dollars sont disponibles à la case d'arrivée
      Modifier l'état de la case d'arrivée selon les règles de transformation

      Afficher le curseur à la position d'arrivée
      Réafficher la case de départ où était le curseur

Tant que Points < 15 ET curseur non bloqué

Écrire le rapport
```