Licence Informatique - 2ème Année
Rapport du Projet d'assembleur

Le jeu Démineur
Fichier source : DEMINEUR.X68

Auteur :

Matthieu COMME

Année universitaire 2024-2025

SOMMAIRE

Introduction

Nous connaissons tous le légendaire jeu du Démineur, popularisé par Windows.
Le but est de localiser des mines cachées dans une grille représentant un champ de mines, avec pour seule indication le nombre de mines dans les zones adjacentes.

Dans ce rapport, nous racontons le développement du projet : le raisonnement, les algorithmes mis en œuvre et les difficultés rencontrées.

Nous sommes partis du principe qu'il fallait d'abord créer un jeu fonctionnel, sans fioritures. Si nous y parvenons dans les délais, recréer une interface familière et des fonctionnalités améliorant le tout.

Vous trouverez en annexe des schémas illustrant les algorithmes et concepts annotés.

I. Créer un jeu fonctionnel

A. Tracer la grille

Une grille de jeu est simple à dessiner. Nous décidons de créer un carré pour simplifier les mesures. Pour tracer les lignes qui la composent, nous allons appliquer l'algorithme suivant :

Soit deux points A (xA,yA) et B (xB,yB), xMax la taille de la grille et LC la largeur d'une case.

[1] Lignes verticales :

On initialise A = (0,0) et B = (0, xMax)
On trace une ligne entre les deux points.
Tant qu'on n'a pas le nombre de lignes souhaité, on augmente de la largeur d'une case les abscisses de nos deux points :
-> A = (xA + LC, 0) et B = (xB + LC, xMax) et on trace une nouvelle ligne.

[2] Lignes horizontales :

Même idée : A = (0,0) et B = (xMax,0) puis A = (0, yA + LC) et B = (xMax, yB + LC)

Nous avons détaillé les calculs de coordonnées pour ce premier algorithme.
Nous ne le ferons plus par la suite pour alléger le compte-rendu.

B. Modélisation

Nous représenterons la grille sous la forme d'une chaîne, le i-ème caractère représente le contenu de la case correspondante.
'B' = mine; '0' = 0 mine adjacente ; '1' = 1 mine adjacente …

Hésitation entre deux formats de grille :

- Séparer chaque caractère de la chaîne avec des 0, pour afficher les nombres avec DRAW_STRING (et non AFFCAR il faut manipuler les numéros de lignes et de colonnes, et nous préférons les coordonnées)
  - ex: 'B',0,'2',0,'B',0,'1',0 …
- Une chaîne simple et indiquer le nombre de mines avec un code couleur
  - ex: 'B2B1'...

Nous avons choisi le premier format car c'est plus agréable pour l'utilisateur de voir le nombre affiché.

Bien plus tard dans le développement, en travaillant sur le choix de la difficulté, nous avons rencontré un problème. En déclarant la chaîne de la grille difficile, une erreur de syntaxe apparaît après un certain nombre de caractères dans la chaîne. Nous supposons que l'instruction est trop longue, car pour chaque caractère, il y a un 0 (pour le DRAW_STRING).

Pour contrer ça, nous avons eu l'idée de "déclarer une variable" en .W qui sera composée du caractère à afficher sur l'octet de poids fort, et d'un 0 sur l'octet de poids faible. Ce sera lui qu'on affichera et non directement le caractère dans le tableau.
Certaines solutions paraissent évidentes mais nous n'y pensons pas directement.

<u>Génération "aléatoire" de grille :</u>
Avantage :

- rejouabilité quasi infinie

Inconvénients:

- facile de placer les mines, mais placer les nombres l'est beaucoup moins.
- doit prendre en compte chaque bordure et chaque coin pour l'incrémentation :
    - si une mine est placée au centre, on doit incrémenter toutes les cases adjacentes,
    - si une mine est au milieu de la bordure gauche, on n'incrémente pas les cases à gauche car celles-ci sont sur la droite de la grille

Toujours dans l'optique de créer en premier lieu un jeu fonctionnel, nous travaillerons d'abord sur une grille prédéfinie.
Si par la suite nous ne savons ou ne pouvons pas générer aléatoirement, nous devrons prévoir un certain nombre de grilles, et une sera sélectionnée au hasard pour la partie en cours.

<u>Conditions de victoire et de défaite :</u>
Victoire : si le compteur de cases restantes est nul
Défaite : si la case cliquée contient une mine.

Problème majeur (plusieurs heures pour le résoudre) : la partie se termine instantanément (victoire) après le 1er clic sur une case qui n'est pas une mine.
Cela n'arrive pas en mode débuggage, donc incompréhension totale.

Après de nombreux, nombreux essais en isolé, puis dans le programme complet, on en déduit que lors d'un simple clic, d'un point de vue humain, la machine en capte une multitude, ce qui diminue très rapidement le compteur à zéro et mène à une victoire.

Pour y remédier :

- Soit un "tableau de booléens" de longueur le nombre de cases. Toutes les valeurs étant initialisées à 0, si on clique sur la case i alors t[i] = 1.
    - Lors des vérifications avant de rentrer dans les calculs, si t[i] = 1 alors on ignore le clic et on retourne au départ.
- Soit on change la couleur de la case jouée. Lorsqu'on clique sur une case, on vérifie la couleur et on ignore le clic si nécessaire.

Cette deuxième option paraît bien plus simple, donc elle sera choisie.

C. Déroulement d'un tour de jeu

Voici la boucle représentant un tour de jeu :

Après un clic gauche, la couleur du pixel cliqué est comparée à la couleur d'une case non-révélée.
Si elle est différente (ex: case déjà révélée, clic hors de la grille…) alors
    -> le clic est ignoré : retour au début de la boucle
Sinon,
    l'index de la case cliquée est calculé et on regarde le contenu de cette case.
    Si c'est une mine alors
        -> défaite : on affiche toutes les mines et fin de partie.
    Sinon
        le nombre de mines adjacentes à cette case est affiché
        le compteur de cases restantes est décrémenté
        si ce compteur est nul alors
            -> victoire : fin de partie
        sinon
            -> retour au début de la boucle

D. Sous-routines

[3] GET_I
A partir d'un pixel cliqué A = (X,Y) et LC la largeur d'une case, calcule l'indice de la case correspondante. On utilise par la suite la division entière.
Diviser X par LC donne la colonne C, diviser Y par LC donne la ligne L.
On obtient l'indice en multipliant L par le nombre de colonnes et en ajoutant C

XY_CASE
A partir de l'indice, calcule les coordonnées du coin supérieur gauche et du coin inférieur droit de la case. Ça commence par le processus inverse de GET_I.
On recalcule la ligne et la colonne de la case, puis on multiplie par LC pour avoir le coin gauche. On ajoute LC à X et Y pour obtenir le coin droit.

COULEUR_NOMBRE
Compare un caractère, en modifie la couleur et l'affiche au milieu de la case.
Ex: '0' = noir; '1' = vert etc.

VICTOIRE
Affiche le message de victoire.

DÉFAITE
Parcours du tableau et révèle en rouge l'emplacement de chaque mine.
Affiche le message de défaite

II. Interface

Nous avons pour référence l'interface du Démineur Windows XP[4].
La grille est entourée d'une bordure grise. En haut, deux compteurs sont présents :

- un chronomètre en secondes
- le nombre de mines supposément restantes

Pour mettre en place les compteurs, on peut afficher des chaînes de caractères.
Mais ce serait plus sympa de créer un afficheur 7 segments, non ?
Enfin, il existe un bouton entre ces deux compteurs qui permet de lancer une nouvelle partie.

A) Interface globale

En annexe se trouve le schéma légendé de l'interface[5].
On trace un rectangle gris de la taille de la grille, en incluant les bordures.
Puis vient le tour des deux rectangles noirs pour les afficheurs.

Jusqu'à présent, l'origine du repère de notre grille était le point (0,0). Nous devons maintenant considérer le décalage occasionné par les bordures de l'interface.
Dorénavant, toutes les "fonctions" utilisant les coordonnées seront corrigées grâce à la sous-routine SET_ORIGINE_GRILLE.

On ajoute un rectangle jaune entre les afficheurs. En cliquant dessus, une nouvelle partie se lance. Réalisable simplement en testant la couleur du pixel lors d'un clic.
Si elle est égale à la couleur du bouton, on réinitialise le jeu en retournant au tout début du programme.

Dans l'idée d'avoir une interface facilement customisable, nous la créons à l'aide de peu de constantes, la majorité des mesures étant calculée à partir des ces quelques valeurs fixes : sous-routines INIT_CONST 1 et 2.

B) Afficheurs 7 segments

1) Écrire un chiffre

Nous avons vu en Expériences Informatiques l'an dernier comment écrire un chiffre avec ce dispositif. Pourquoi ne pas l'adapter en assembleur ?
Un chiffre est constitué de sept segments[6], allumés ou non en fonction de ce qu'on veut afficher. On crée donc une librairie BIB_AFFICHEUR qui va contenir toutes ces sous-routines.

Voici le déroulé type de la création d'un segment :

- initialiser les 2 points X et Y, à l'origine de l'afficheur
- placer X et Y aux bons endroits
- tracer la ligne entre ces deux points

A partir de ces sous-routines, nous pouvons en créer d'autres pour afficher nos chiffres. Exemples :

- '1' : segments B et C
- '2' : segments A, B, G, E et D
- etc.

2) Écrire un nombre

Nous concevons l'afficheur sur 3 chiffres, deux calculs sont nécessaires.
Soit x = 123, le nombre à afficher.

- 123/100 = 1*100 + 23. Nous affichons donc 1 sur le chiffre des centaines puis utilisons le reste pour le calcul suivant.
- 23/10 = 2*10 + 3. Afficher 2 sur le chiffre des dizaines et 3 les unités.

Avant d'afficher chaque chiffre, nous plaçons le repère adéquat. En effet, après avoir affiché le chiffre des centaines, nous devons décaler le repère vers la droite pour le chiffre des dizaines, puis encore une fois pour les unités.

Si nous voulons en écrire un autre, il faudra effacer le nombre actuel. Pour ce faire, on affiche '8' en noir car ce chiffre est composé de tous les segments possibles.

Le premier compteur représente le nombre de mines restantes, et s'actualise lorsqu'on "flag" ou "deflag" une case (cf. sous-chapitre Flag).

Le second est un chronomètre. Il devrait donc s'actualiser chaque seconde, environ. Nous devons trouver le moyen de ne pas interrompre le chrono quand on clique sur une case.

III. Fonctionnalités

Nous avons à présent un jeu fonctionnel. Il est temps d'ajouter des fonctionnalités supplémentaires pour améliorer la qualité du jeu.

A. Sélection de la difficulté

Avant de lancer la partie, un menu[7] apparaît nous permettant de choisir la difficulté. Cela se fait en cliquant sur la case correspondante. On teste la couleur cliquée et on modifie les constantes dépendant de la difficulté comme la taille de la grille, le nombre de mines etc.

B. Flag

Si vous pensez savoir qu'une mine se cache sous une case, vous pouvez poser un "flag" dessus. Cela permet de se souvenir de sa position, ainsi que de désactiver les clics sur cette case. On peut placer autant de flag qu'il y a de mines.
Dans la boucle principale, nous ajoutons un test :

```
Si (un clic droit est détecté) alors
        Si (la case cliquée est cachée) alors
                Si (il n'y a plus de flag disponible) alors
                        -> retour au début de la boucle principale
                Sinon
                        -> on pose un flag
                Finsi
        Sinon si (la case cliquée est déjà flag)
                -> on le retire
        Finsi
Finsi
```

Après avoir (de)flag, on actualise le nombre de mines et l'afficheur. Puis nous imposons un léger délai d'attente pour limiter spam clics.

C. Cases adjacentes

Dans le jeu de base, lorsqu'on clique sur une case '0', toutes les cases adjacentes (notées C_ADJ) sont révélées, récursivement. Sans savoir si on peut le faire dans le temps imparti, on peut d'abord créer une fonction retournant les cases adjacentes à la case cliquée. Elle sera toujours utile pour le prochain sous-chapitre.

Le nombre et la position des C_ADJ varient en fonction de la position de la case initiale. Le coin supérieur n'aura pas les mêmes C_ADJ qu'une case au milieu de la grille. C'est pourquoi nous devons réaliser plusieurs tests afin de classer notre cas.

Pour avoir le numéro de ligne ou de colonnes, nous effectuons les mêmes calculs (division entière) que GET_I[3].

```
Si (la case est sur la première ligne) alors
        si (sur la première colonne) alors
                -> coin supérieur gauche
        sinon
                si (sur la dernière colonne) alors
                        -> coin supérieur droit
                sinon
                        -> bord haut, coins exclus
Sinon
        Si (sur la dernière ligne) alors
                si (sur la première colonne) alors
                        -> coin inférieur gauche
                sinon
                        si (sur la dernière colonne) alors
                                -> coin inférieur droit
                        sinon
                                -> bord bas, coins exclus
        Sinon
                Si (sur la première colonne) alors
                        -> bord gauche, coins exclus
                sinon
                        Si (sur la dernière colonne) alors
                                -> bord droit, coins exclus
                        Sinon
                                -> la case est sur aucun bord
```

Une fois que notre case est classée, nous mettons dans une "file" ses C_ADJ.
Trois exemples sont illustrés en annexe[8].

Nous appliquerons le traitement lorsqu'on défile.

D. Génération “aléatoire” de grille

Avec l’aide du précédent sous-chapitre, nous pouvons créer une grille pseudo-aléatoire. On commence par initialiser une grille remplie de ‘0’.
Voici la boucle se répétant tant que le nombre de mines placées est inférieur au nombre requis.

On tire au sort un nombre entre 0 et (nombre de cases-1).
Si (la case correspondante n’est pas une mine) alors

- elle devient une mine
- on incrémente toutes ses cases adjacentes qui ne sont pas des mines

E. Propagation du clic

On rappelle que dans le jeu de base, lorsqu’on clique sur une case ‘0’, toutes les cases adjacentes sont révélées, récursivement.
L’algorithme utilisé lorsqu’on clique sur un ‘0’ est le suivant :

On calcule toutes ses cases adjacentes.
Puis une par une, on vérifie si elles sont cachées.
Si c’est le cas, on révèle, sauf si c’est une mine.

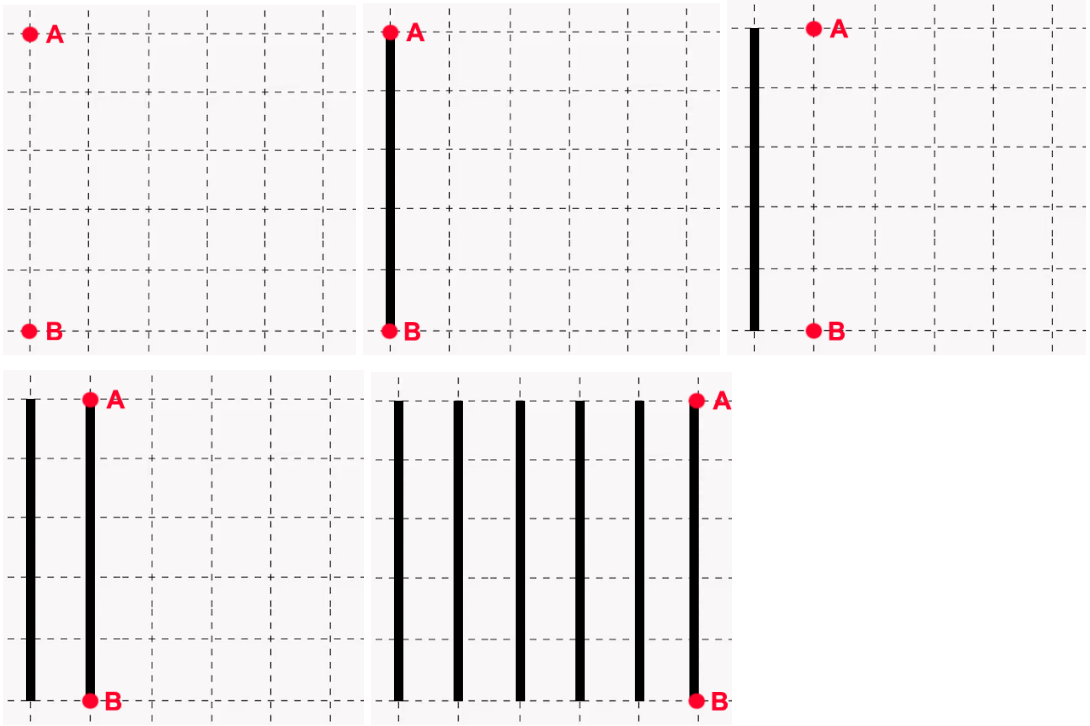
IV. Conclusion

Je suis plutôt fier de ce projet, qui était amusant et souvent prise de tête avec ce langage inconnu il y a 4 mois. Il reste plusieurs améliorations potentielles et des algorithmes simplifiables, mais cette version est, je trouve, convenable.

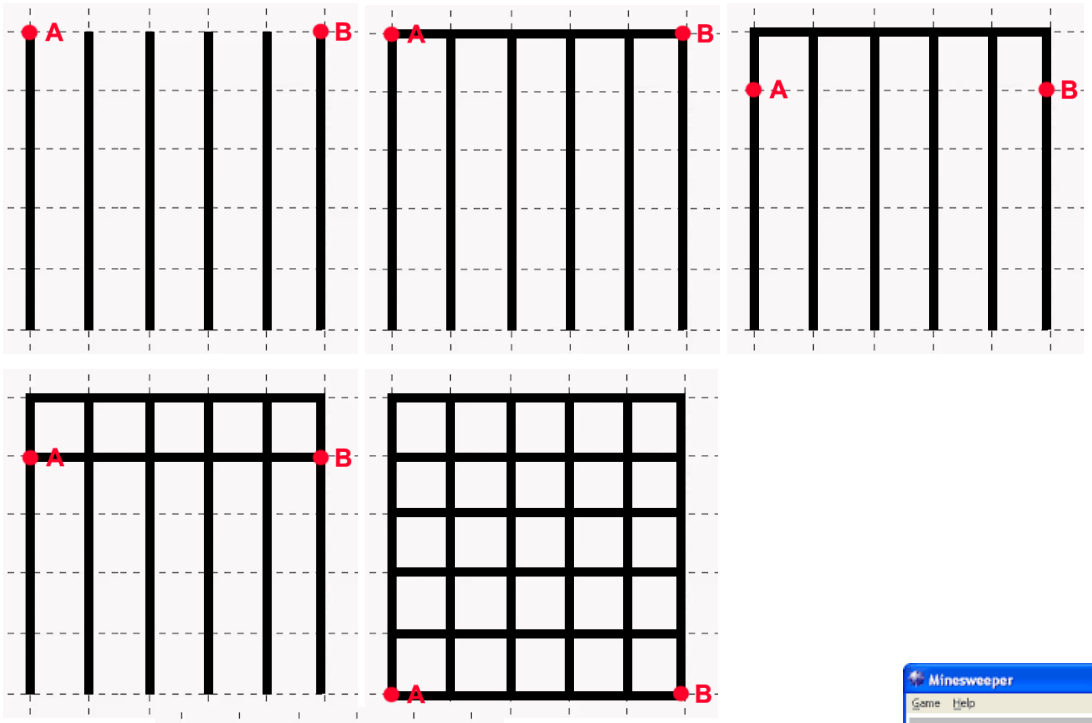
- Meilleur système de chrono, limiter l’arrêt du temps quand on maintient le clic
- Pousser la propagation du clic, qui continue tant qu’elle rencontre un ‘0’
- Bouton “aide” qui indique une case sans danger, ou bien une mine ?
- Une meilleure génération de nombre “aléatoire”
- Enregistrer le meilleur temps et faire un classement pour chaque difficulté

V. ANNEXE

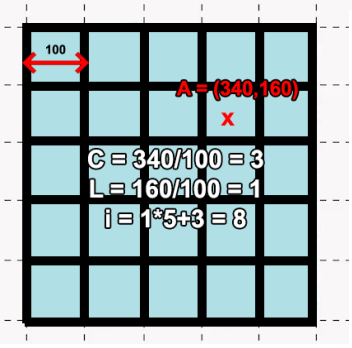
1: Lignes verticales



2: Lignes horizontales

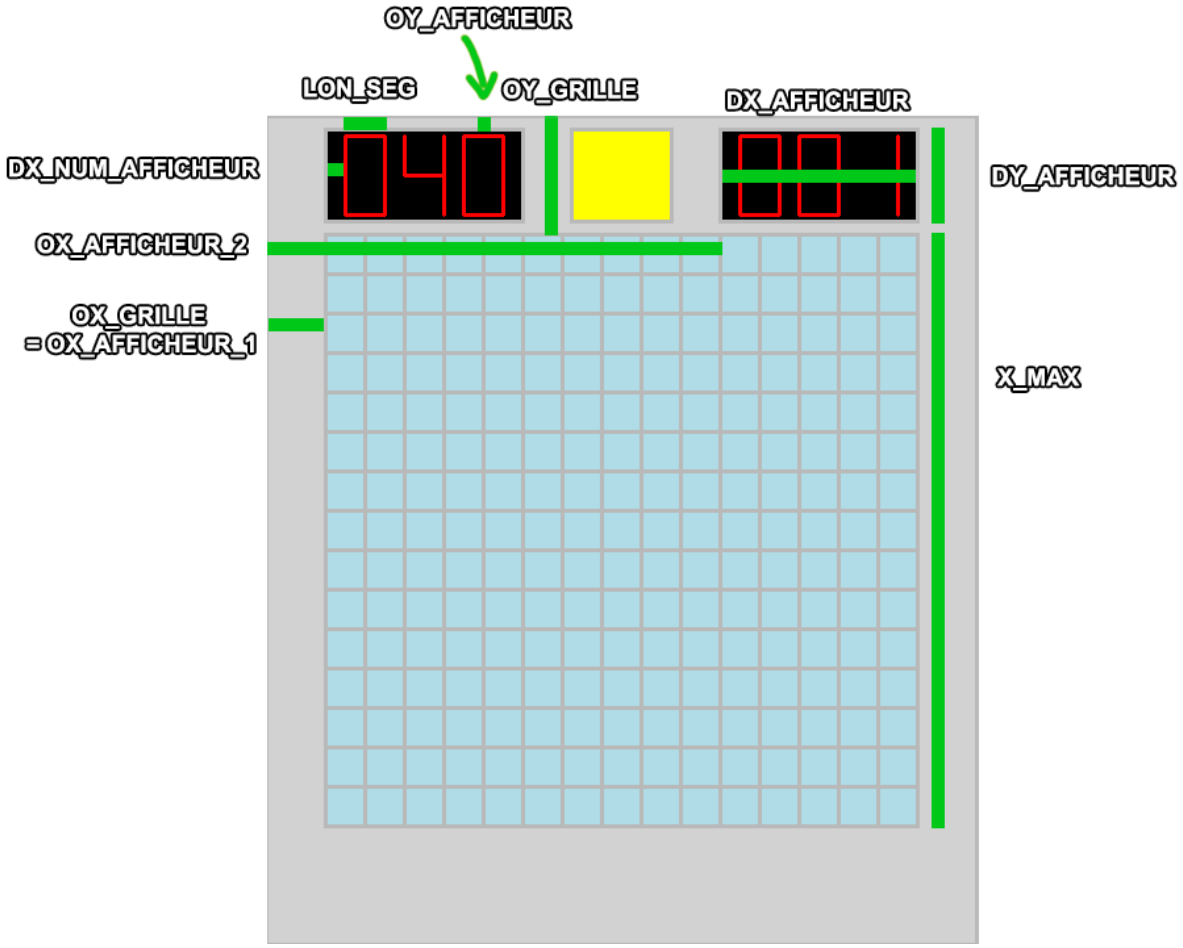


3: GET_I

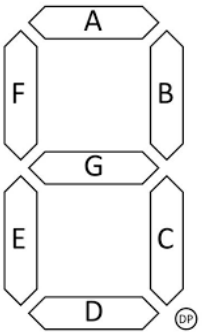


4: Démineur Windows XP

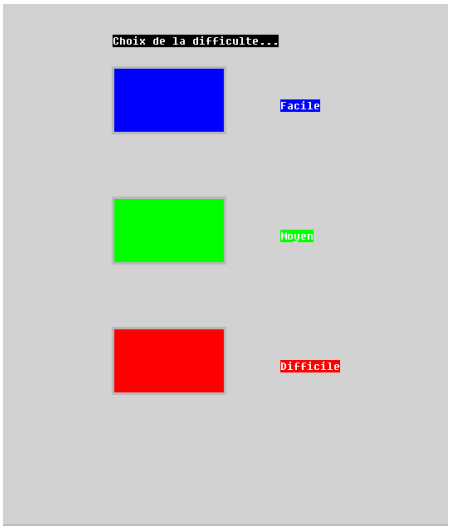
5: Schéma légendé de l'interface



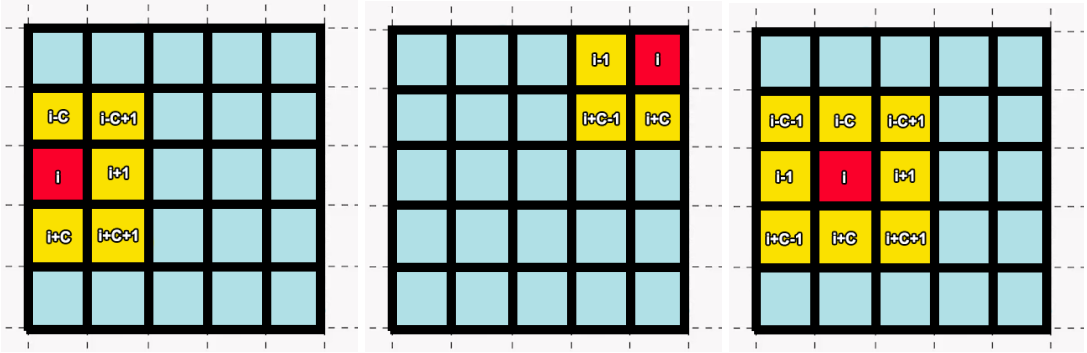
6: Afficheur 7 segments



7: Choix de la difficulté



8: Calcul des cases adjacentes (C = nombre de colonnes)



Code commenté du programme

```
* Fichier source: DEMINEUR.X68
*               D1-4.W = XY ;
*               D6.W = indice tab ; D7 = afficheur
*               A0 = grille
*               A1 = DRAW_STRING
*               A2 = ADJ
*-----------------------------------------------------
            ORG    $1000
START:
            MOVE.W  RES_X,D1            * largeur
            SWAP    D1
            MOVE.W  RES_Y,D1            * hauteur
            JSR RESOLUTION
            JSR INIT_CONST_1
            JSR INTERFACE_DIFFICULTE
CHOIX_DIFFICULTE:
            MOVE.L  #0,D1
            JSR GET_MOUSE
            AND.B   #1,D0
            CMP.B   #1,D0
            BNE CHOIX_DIFFICULTE
            SWAP    D1
            MOVE.W  D1,D2
            SWAP    D1
            JSR GET_PIX_COLOR
            MOVE.L  D0,D1
```

```
            CMP.L  #$00FF0000,D1
            BEQ SETUP_FACILE
            CMP.L  #$0000FF00,D1
            BEQ SETUP_MOYEN
            CMP.L  #$000000FF,D1
            BEQ SETUP_DIFFICILE
            BRA CHOIX_DIFFICULTE
FIN_CHOIX_DIFFICULTE:
            JSR INIT_CONST_2
            MOVE.W  #$FF00,D1
            JSR POS_CURS                * Efface l'ecran
            JSR INTERFACE

BOUCLE_SOURIS:                          * boucle principale
            MOVE.L  BUFFER_CHRONO,D7
            MOVE.L  #0,BUFFER_CHRONO
ATTENTE_BOUCLE_SOURIS:
            ADD.L   #1,D7
            MOVE.L  #0,D1
            JSR GET_MOUSE
            AND.B   #3,D0
            CMP.B   #1,D0
            BEQ CLIC_GAUCHE
            CMP.B   #2,D0
            BEQ CLIC_DROIT
            CMP.L   REF_CHRONO,D7     * $20000 = 1 sec sur mon laptop
            BMI ATTENTE_BOUCLE_SOURIS
            ADD.W   #1,CHRONO
            JSR AFFICHE_CHRONO
            BRA BOUCLE_SOURIS
FIN_BOUCLE_SOURIS:

CLIC_DROIT:
            MOVE.L  D7,BUFFER_CHRONO
            SWAP    D1
            MOVE.W  D1,D2
            SWAP    D1
            JSR GET_PIX_COLOR           * contrôle la couleur de la case cliquée
            JSR GET_I
            CMP.L   COULEUR_CACHEE,D0
            BEQ FLAG
            CMP.L   COULEUR_FLAG,D0
            BEQ DEFLAG
            BRA BOUCLE_SOURIS
```

```
FLAG:                                   * desactive une case
        CMP.W   #0,NB_MINES     * ignore le clic s'il n' y a plus de flag disponible
        BEQ     BOUCLE_SOURIS
        SUB.W   #1,NB_MINES
        MOVE.L  COULEUR_FLAG,D1
        JSR SET_FILL_COLOR
        BRA SUITE_DE_FLAG
DEFLAG:                                 * la réactive
        MOVE.L  COULEUR_CACHEE,D1
        JSR SET_FILL_COLOR
        ADD.W   #1,NB_MINES
SUITE_DE_FLAG:
        MOVE.L  COULEUR_CRAYON,D1
        JSR SET_PEN_COLOR
        JSR XY_CASE
        JSR DRAW_FILL_RECT
        JSR AFFICHE_SCORE
        JSR COOLDOWN                    * eviter les spam clicks
        BRA BOUCLE_SOURIS
FIN_CLIC_DROIT:
CLIC_GAUCHE:
        MOVE.L  D7,BUFFER_CHRONO
        SWAP    D1
        MOVE.W  D1,D2
        SWAP    D1
        JSR GET_PIX_COLOR
        CMP.L   COULEUR_RESTART,D0
        BEQ RESTART
        CMP.L   COULEUR_CACHEE,D0
        BNE ATTENTE_BOUCLE_SOURIS    * ignore si la case n'est pas
    "cliquable"
        JSR GET_I
        MOVE.B  (A0,D6),N               * N = contenu de la case
        MOVE.L  COULEUR_CRAYON,D1
        JSR SET_PEN_COLOR
        MOVE.L  COULEUR_REVELEE,D1
        JSR SET_FILL_COLOR
        JSR XY_CASE
        JSR DRAW_FILL_RECT
CMP_COULEUR_CLIC:
        CMP.B   #66,N                   * = 'B'
        BEQ DEFAITE
        JSR COULEUR_NOMBRE
        SUB.W   #1,CASES_RESTANTES
```

```
TEST_CASES_ADJ:
        CMP.B   #$30,N              * On révèle ADJ si N = '0'
        BEQ REVELE_ADJ
FIN_TEST_CASES_ADJ:
        CMP.W   #0,CASES_RESTANTES
        BNE BOUCLE_SOURIS
        JSR VICTOIRE
        BRA FIN
FIN_CLIC_GAUCHE:

FIN:                                * laisse peu de temps pour relancer une partie
        ADD.L   #1,D6
        MOVE.L  #0,D1
        JSR GET_MOUSE
        AND.B   #1,D0
        CMP.B   #1,D0
        BNE FIN
        SWAP    D1
        MOVE.W  D1,D2
        SWAP    D1
        JSR GET_PIX_COLOR
        CMP.L   COULEUR_RESTART,D0
        BEQ RESTART
        CMP.L   #$30000,D6
        BMI FIN

        JMP FINPRG

   INCLUDE 'BIBLIO.x68'
   INCLUDE 'BIBPERIPH.x68'
   INCLUDE 'BIBGRAPH.x68'
   INCLUDE 'BIB_DEMINEUR.x68'
   INCLUDE 'BIB_AFFICHEUR.x68'

   ORG     $3000

NB_COLONNES:    DS.W 1
LARGEUR_CASE:   DS.W 1              * xmax / nb_col
CENTRE_CASE:    DS.W 1
RES_X:  DC.W 700
RES_Y:  DC.W 700
X_MAX:  DC.W 450                    * (multiple de 45)
OX_GRILLE:  DS.W 1                  * xmax / 10
OY_GRILLE:  DS.W 1                  * xmax / 5
```

```
OX_AFFICHEUR_1: DS.W 1              * = ox_grille
OX_AFFICHEUR_2: DS.W 1              * ox_grille + xmax - dx_aff
OX_AFFICHEUR: DS.W 1
OX_AFFICHEUR_ACTUEL: DS.W 1
OY_AFFICHEUR: DS.W 1                * 2*OY_aff + DY_aff = OY_grille
DX_AFFICHEUR: DC.W 150              * 4*dx_num + 3*lon_seg ; * dc sinon bug ?
DY_AFFICHEUR: DC.W 70
DX_NUM_AFFICHEUR: DS.W 1            * lon_seg / 2
LON_SEG:    DC.W 30
COULEUR_SEG:    DC.L $000000FF
CHRONO:  DC.W    0
BUFFER_CHRONO:  DC.L 0
REF_CHRONO:     DC.L $6000          * définit l'écoulement du temps
COULEUR_CACHEE: DC.L $00E6E0B0
COULEUR_REVELEE:    DC.L $00000000
COULEUR_MINES:  DC.L $000000FF
COULEUR_FLAG:   DC.L $00FF00FF
COULEUR_CRAYON: DC.L $00BBBBBB
COULEUR_FOND:   DC.L $00D3D3D3
COULEUR_AFFICHEUR:  DC.L $00000000
COULEUR_RESTART:    DC.L $0000FFFF
N:  DC.W 0
NB_CASES:   DS.W 1
NB_MINES:  DS.W 1
CASES_RESTANTES: DC.W 0
ADJ:    DS.W 9                      * contient les indices des cases adjacentes
GRILLE: DS.B 230                    * > NB_CASES en difficile
MSG_VICTOIRE:   DC.B 'Bravo pour cette belle victoire !',0
MSG_DEFAITE:    DC.B 'Dommage ! C est perdu...',0
CHAINE_DIFFICULTE:  DC.B 'Choix de la difficulte...',0
CHAINE_FACILE:  DC.B 'Facile',0
CHAINE_MOYEN:   DC.B 'Moyen',0
CHAINE_DIFFICILE: DC.B 'Difficile',0

    END    START
```

*** Fichier : BIB_DEMINEUR**

```
*-------------------------------------------------------

INIT_CONST_1:                   * initialisation de la première partie des constantes
        MOVE.W  X_MAX,D0
        DIVU    #5,D0
        MOVE.W  D0,OY_GRILLE
        ASR.W   #1,D0
        MOVE.W  D0,OX_GRILLE
        MOVE.W  OX_GRILLE,D1
        MOVE.W  D1,OX_AFFICHEUR_1
        ADD.W   X_MAX,D1
        SUB.W   DX_AFFICHEUR,D1
        MOVE.W  D1,OX_AFFICHEUR_2
        MOVE.W  OY_GRILLE,D1
        SUB.W   DY_AFFICHEUR,D1
        ASR.W   #1,D1
        MOVE.W  D1,OY_AFFICHEUR
        MOVE.W  LON_SEG,D1
        ASR.W   #1,D1
        MOVE.W  D1,DX_NUM_AFFICHEUR
        MOVE.W  #0,CHRONO
        MOVE.L  #ADJ,A2
        RTS

INIT_CONST_2:                   * initialisation du reste des constantes
        MOVE.W  NB_CASES,D1
        SUB.W   NB_MINES,D1
        MOVE.W  D1,CASES_RESTANTES
        MOVE.L  #0,D1
        MOVE.W  X_MAX,D1
        DIVU    NB_COLONNES,D1
        MOVE.W  D1,LARGEUR_CASE
        ASR.W   #1,D1
        MOVE.W  D1,CENTRE_CASE
        MOVE.L  #N,A1               * contenu de la case
        MOVE.L  #GRILLE,A0
        JSR REMPLIT_ZERO_GRILLE
        JSR RANDOM_GRILLE
        RTS
```

```
REMPLIT_ZERO_GRILLE:
        MOVE.W  #0,D0
BOUCLE_ZERO_GRILLE:
        MOVE.L  #$30303030,(A0)+
        ADD.W   #4,D0
        CMP.W   NB_CASES,D0
        BMI BOUCLE_ZERO_GRILLE
        MOVE.L  #GRILLE,A0
        RTS

RANDOM_GRILLE:                  * génère une grille aléatoire
        MOVE.L  #0,D2           * compteur de mines
BOUCLE_RANDOM:
        MOVE.L  #0,D3           * compteur attente
RANDOM_ATTENTE_1:               * s'il n' y a pas d'attente, les nombres
                                  sélectionnés seront consécutifs
        ADD.L   #1,D3
        CMP.L   #$00009ABC,D3
        BNE RANDOM_ATTENTE_1
        MOVE.L  #0,D3
        JSR GET_TIME
        AND.L   #$0000FFFF,D1
        DIVU    NB_CASES,D1
        SWAP    D1
        MOVE.W  D1,D6
        CMP.B   #66,(A0,D6)     * si t[D1] = 'B', on recommence
        BEQ BOUCLE_RANDOM
        MOVE.B  #66,(A0,D6)
        JSR CASES_ADJ
RANDOM_ADJ_INCR:
        CMP.L   #ADJ,A2
        BEQ FIN_RANDOM_ADJ
        MOVE.W  -(A2),D0
        CMP.B   #66,(A0,D0)
        BEQ RANDOM_ADJ_INCR
        ADD.B   #1,(A0,D0)
        BRA RANDOM_ADJ_INCR
FIN_RANDOM_ADJ:
RANDOM_ATTENTE_2:
        ADD.L   #1,D3
        CMP.L   #$0000DEF0,D3
        BNE RANDOM_ATTENTE_2
        ADD.W   #1,D2
        CMP.W   NB_MINES,D2
```

```
            BNE BOUCLE_RANDOM
FIN_RANDOM_GRILLE:
            RTS

TRACER_FOND:
            MOVE.L  COULEUR_FOND,D1
            JSR SET_FILL_COLOR
            JSR RESET_D
            MOVE.W  OX_GRILLE,D3
            ASL.W   #1,D3
            ADD.W   X_MAX,D3
            MOVE.W  OY_GRILLE,D4
            ASL.W   #1,D4
            ADD.W   X_MAX,D4
            JSR DRAW_FILL_RECT
            RTS

INTERFACE_DIFFICULTE:
            MOVE.L  COULEUR_CRAYON,D1
            JSR SET_PEN_COLOR
            MOVE.B #3,D1
            JSR WIDTH_PEN
            JSR TRACER_FOND
            * MESSAGE CHOIX *
            MOVE.L  #CHAINE_DIFFICULTE,A1
            MOVE.L  #$00000000,D1
            JSR SET_FILL_COLOR
            MOVE.W  D3,D1
            ASR.W   #1,D1
            MOVE.W  D1,D3
            ASR.W   #1,D1
            MOVE.W  D4,D2
            ASR.W   #2,D2
            MOVE.W  D2,D4
            ASR.W   #2,D2
            JSR DRAW_STRING
            MOVE.W  D1,D5                   * D5 = sauvegarde de D1
            * FACILE *
            MOVE.L  #$00FF0000,D1
            JSR SET_FILL_COLOR
            MOVE.W  D5,D1
            ASL.W   #1,D2
            MOVE.W  D4,D6                   * D6 = espace vertical entre 2 boutons
            MOVE.W  D2,D7                   * D7 = moitie de la hauteur d'une case
```

```
        ASR.W   #1,D7
        JSR DRAW_FILL_RECT
        MOVE.W  D1,D5
        ASR.W   #1,D1
        ADD.W   D3,D1
        ADD.W   D7,D2
        MOVE.L  #CHAINE_FACILE,A1
        JSR DRAW_STRING
        * MOYEN *
        MOVE.L  #$0000FF00,D1
        JSR SET_FILL_COLOR
        MOVE.W  D5,D1
        SUB.W   D7,D2
        ADD.W   D6,D2
        ADD.W   D6,D4
        JSR DRAW_FILL_RECT
        MOVE.W  D1,D5
        ASR.W   #1,D1
        ADD.W   D3,D1
        ADD.W   D7,D2
        MOVE.L  #CHAINE_MOYEN,A1
        JSR DRAW_STRING
        * DIFFICILE *
        MOVE.L  #$000000FF,D1
        JSR SET_FILL_COLOR
        MOVE.W  D5,D1
        SUB.W   D7,D2
        ADD.W   D6,D2
        ADD.W   D6,D4
        JSR DRAW_FILL_RECT
        ASR.W   #1,D1
        ADD.W   D3,D1
        ADD.W   D7,D2
        MOVE.L  #CHAINE_DIFFICILE,A1
        JSR DRAW_STRING
        RTS

SETUP_FACILE:
        MOVE.W  #5,NB_COLONNES
        MOVE.W  #25,NB_CASES
        MOVE.W  #3,NB_MINES
        BRA FIN_CHOIX_DIFFICULTE
```

```
SETUP_MOYEN:
        MOVE.W  #9,NB_COLONNES
        MOVE.W  #81,NB_CASES
        MOVE.W  #10,NB_MINES
        BRA FIN_CHOIX_DIFFICULTE

SETUP_DIFFICILE:
        MOVE.W  #15,NB_COLONNES
        MOVE.W  #225,NB_CASES
        MOVE.W  #40,NB_MINES
        BRA FIN_CHOIX_DIFFICULTE

RESET_D:
        MOVE.L  #0,D0
        MOVE.L  #0,D1
        MOVE.L  #0,D2
        MOVE.L  #0,D3
        MOVE.L  #0,D4
        MOVE.L  #0,D5
        MOVE.L  #0,D6
        *MOVE.L  #0,D7
        RTS

SET_ORIGINE_GRILLE:
    * corrige D1-4 en prenant en compte le décalage lié à l'interface
        ADD.W   OX_GRILLE,D1
        ADD.W   OX_GRILLE,D3
        ADD.W   OY_GRILLE,D2
        ADD.W   OY_GRILLE,D4
        RTS

INTERFACE:                      * dessine l'interface de jeu
        JSR TRACER_FOND
        * AFFICHEUR 1 *
        MOVE.L  COULEUR_AFFICHEUR,D1
        JSR SET_FILL_COLOR
        MOVE.W  OX_AFFICHEUR_1,D1
        MOVE.W  OY_AFFICHEUR,D2
        MOVE.W  D1,D3
        MOVE.W  D2,D4
        ADD.W   DX_AFFICHEUR,D3
        ADD.W   DY_AFFICHEUR,D4
        JSR DRAW_FILL_RECT
        MOVE.W  D3,D5          * D5 et D6 vont aider à tracer le bouton restart
```

```
        * AFFICHEUR 2 *
        MOVE.W  OX_AFFICHEUR_2,D1
        MOVE.W  D1,D3
        ADD.W   DX_AFFICHEUR,D3
        JSR DRAW_FILL_RECT
        MOVE.W  D1,D6                   * D6 = 0.25*ecart entre les 2 afficheurs
        MOVE.W  D1,D3
        * BOUTON RESTART *
        MOVE.L  COULEUR_RESTART,D1
        JSR SET_FILL_COLOR
        MOVE.W  D5,D1
        SUB.W   D5,D6
        ASR.W   #2,D6
        ADD.W   D6,D1
        SUB.W   D6,D3
        JSR DRAW_FILL_RECT
        ADD.W   #5,OY_AFFICHEUR  * ajuste position des compteurs (rustine 1)
        JSR TRACER_GRILLE
        JSR AFFICHE_CHRONO
        JSR AFFICHE_SCORE
        RTS

TRACER_GRILLE:
        MOVE.L  COULEUR_CACHEE,D1
        JSR SET_FILL_COLOR
        MOVE.W  OX_GRILLE,D1
        MOVE.W  OY_GRILLE,D2
        MOVE.W  D1,D3
        MOVE.W  D2,D4
        ADD.W   X_MAX,D3
        ADD.W   X_MAX,D4
        JSR DRAW_FILL_RECT
PRINT_LIGNES_GRILLE:
        MOVE.L COULEUR_CRAYON,D1
        JSR SET_PEN_COLOR
        MOVE.B #3,D1
        JSR WIDTH_PEN
        ADD.W   #1,NB_COLONNES                  * nb_lignes = nb_col + 1
        JSR RESET_D
        MOVE.W X_MAX,D4
        JSR SET_ORIGINE_GRILLE
LIGNES_VERTICALES:
        JSR DRAW_LINE
        ADD.W LARGEUR_CASE,D1
```

```
            ADD.W LARGEUR_CASE,D3
            ADD.W #1,D5
            CMP.W NB_COLONNES,D5
            BNE LIGNES_VERTICALES
FIN_LIGNES_VERTICALES:
            JSR RESET_D
            MOVE X_MAX,D3
            JSR SET_ORIGINE_GRILLE
LIGNES_HORIZONTALES:
            JSR DRAW_LINE
            ADD.W LARGEUR_CASE,D2
            ADD.W LARGEUR_CASE,D4
            ADD.W #1,D5
            CMP.W NB_COLONNES,D5
            BNE LIGNES_HORIZONTALES
            JSR RESET_D
            SUB.W   #1,NB_COLONNES
            RTS

GET_I:                          * D1/D2 .W -> D6.W = indice de la case cliquée
            SUB.W   OX_GRILLE,D1
            SUB.W   OY_GRILLE,D2
            AND.L   #$0000FFFF,D1
            AND.L   #$0000FFFF,D2
            DIVU    LARGEUR_CASE,D1         * indice x
            DIVU    LARGEUR_CASE,D2         * indice y
            MULU    NB_COLONNES,D2
            ADD.W   D2,D1
            MOVE.W  D1,D6
            RTS

XY_CASE:                        * i dans D6 -> X/Y dans D1,2/D3,4
            MOVE.L  #0,D1
            MOVE.L  D1,D2
            MOVE.W  D6,D1
            DIVU    NB_COLONNES,D1
            MOVE.W  D1,D2
            MULU    LARGEUR_CASE,D2
            SWAP    D1
            AND.L   #$0000FFFF,D1
            MULU    LARGEUR_CASE,D1
            MOVE.W  D1,D3
            ADD.W   LARGEUR_CASE,D3
            MOVE.W  D2,D4
```

```
            ADD.W   LARGEUR_CASE,D4
            JSR SET_ORIGINE_GRILLE
            RTS

COULEUR_NOMBRE:
* attribue une couleur en fonction de N et l'affiche au milieu de la case
            MOVE.W  D1,D3                       * pour recuperer D1 plus tard
            CMP.B   #$30,N
            BEQ COULEUR_ZERO
            CMP.B   #$31,N
            BEQ COULEUR_UN
            CMP.B   #$32,N
            BEQ COULEUR_DEUX
            CMP.B   #$33,N
            BEQ COULEUR_TROIS
            MOVE.L  #$0D30094,D1 * N >= 4, à continuer si je décide de mettre N>4 ?
            BRA FIN_COULEUR_NOMBRE
COULEUR_ZERO:
            MOVE.L  #$00000000,D1
            BRA FIN_COULEUR_NOMBRE
COULEUR_UN:
            MOVE.L  #$00FF1010,D1
            BRA FIN_COULEUR_NOMBRE
COULEUR_DEUX:
            MOVE.L  #$0090EE90,D1
            BRA FIN_COULEUR_NOMBRE
COULEUR_TROIS:
            MOVE.L  #$0000A5FF,D1
            BRA FIN_COULEUR_NOMBRE
FIN_COULEUR_NOMBRE:
            JSR SET_FILL_COLOR
            MOVE.W  D3,D1
            ADD.W   CENTRE_CASE,D1
            ADD.W   CENTRE_CASE,D2
            JSR DRAW_STRING                     * ecrit N
            RTS
```

```
REVELE_ADJ:                          * révèle les cases non-mines adjacentes
        JSR CASES_ADJ
BOUCLE_REVELE_ADJ:
          MOVE.L  COULEUR_REVELEE,D1
          JSR SET_FILL_COLOR
          CMP.L   #ADJ,A2
          BEQ FIN_BOUCLE_REVELE_ADJ
          MOVE.W  -(A2),D6
          MOVE.B  (A0,D6),N
          CMP.B   #65,N
          BEQ BOUCLE_REVELE_ADJ
          JSR XY_CASE
          ADD.W   CENTRE_CASE,D1
          ADD.W   CENTRE_CASE,D2
          JSR GET_PIX_COLOR   * vérification de la couleur de la case adjacente
          CMP.L   COULEUR_CACHEE,D0
          BNE BOUCLE_REVELE_ADJ
          SUB.W   CENTRE_CASE,D1
          SUB.W   CENTRE_CASE,D2
          JSR DRAW_FILL_RECT
          JSR COULEUR_NOMBRE
          SUB.W   #1,CASES_RESTANTES
          CMP.W   #0,CASES_RESTANTES
          BNE BOUCLE_REVELE_ADJ
          JSR VICTOIRE
          BRA FIN
FIN_BOUCLE_REVELE_ADJ:
          BRA FIN_TEST_CASES_ADJ

COOLDOWN: * permet de temporiser
          MOVE.L  #0,D7
BOUCLE_COOLDOWN:
          ADD.L   #1,D7
          CMP.L   #$00018000,D7         * compteur seconde = environ $30000
          BMI BOUCLE_COOLDOWN
          ADD.L   D7,BUFFER_CHRONO
          MOVE.L  BUFFER_CHRONO,D0
          CMP.L   REF_CHRONO,D0
          BMI FIN_COOLDOWN
          MOVE.L  REF_CHRONO,BUFFER_CHRONO
FIN_COOLDOWN:
   RTS
```

```
VICTOIRE:
        MOVE.L  #$00FF00FF,D1
        JSR SET_FILL_COLOR
        MOVE.L  #MSG_VICTOIRE,A1
        MOVE.W  X_MAX,D1
        ASR.W   #2,D1
        MOVE.W  X_MAX,D2
        ADD.W   #20,D2
        JSR SET_ORIGINE_GRILLE
        JSR DRAW_STRING
        RTS

DEFAITE:
        MOVE.L  COULEUR_MINES,D1
        JSR SET_FILL_COLOR
        MOVE.W  #0,D6                   * indice tableau
REVELE_MINES:
        CMP.B   #66,(A0,D6)
        BEQ MINE
SUITE_REVELE_MINES:
        ADD.W   #1,D6
        CMP.W   NB_CASES,D6
        BNE REVELE_MINES
        MOVE.L  #MSG_DEFAITE,A1
        MOVE.W  X_MAX,D1
        ASR.W   #2,D1
        MOVE.W  X_MAX,D2
        ADD.W   #20,D2
        JSR SET_ORIGINE_GRILLE
        JSR DRAW_STRING
FIN_DEFAITE:
        BRA FIN

MINE:
        JSR XY_CASE
        JSR DRAW_FILL_RECT
        JSR COOLDOWN
        BRA SUITE_REVELE_MINES

RESTART:
        SUB.W   #5,OY_AFFICHEUR         * (rustine 2)
        BRA START
```

```
*-------------------------------------------------------
* Retourne dans (A2).W l'indice des cases adjacentes à la case dont l'indice est dans D6.W
* D5.L = etat GD..HB -> 10..10 = coin haut gauche ; 00..00 -> pas en bordure ; 01..00 -> bord gauche etc.
* D7.W = dernière ligne ( nb_col - 1)
*-------------------------------------------------------

CASES_ADJ:
            MOVE.L  #0,D7
            MOVE.W  NB_COLONNES,D7
            SUB.W   #1,D7
TEST_BORD:
            MOVE.L  #0,D5
            MOVE.L  D6,D0
            DIVU    NB_COLONNES,D0
TEST_BORD_HB:
            CMP.W   #0,D0
            BEQ ADD_BORD_HAUT
            CMP.W   D7,D0
            BEQ ADD_BORD_BAS
FIN_TEST_BORD_HB:

TEST_BORD_GD:
            SWAP D5
            SWAP    D0
            CMP.W   #0,D0
            BEQ ADD_BORD_GAUCHE
            CMP.W   D7,D0
            BEQ ADD_BORD_DROIT
FIN_TEST_BORD_GD:
CMP_BORD:
            MOVE.W  D6,D0
            SWAP    D5
            CMP.B   #2,D5
            BEQ BORD_HAUT
            CMP.B   #1,D5
            BEQ BORD_BAS
            SWAP    D5
            CMP.B   #2,D5
            BEQ BORD_G
            CMP.B   #1,D5
            BEQ BORD_D
            * 8 CASES ADJACENTES *
```

```
        SUB.W   #1,D0
        MOVE.W  D0,(A2)+
        SUB.W   NB_COLONNES,D0
        MOVE.W  D0,(A2)+
        ADD.W   #1,D0
        MOVE.W  D0,(A2)+
        ADD.W   #1,D0
        MOVE.W  D0,(A2)+
        ADD.W   NB_COLONNES,D0
        MOVE.W  D0,(A2)+
        ADD.W   NB_COLONNES,D0
        MOVE.W  D0,(A2)+
        SUB.W   #1,D0
        MOVE.W  D0,(A2)+
        SUB.W   #1,D0
        MOVE.W  D0,(A2)+
FIN_CMP_BORD:
        RTS

BORD_HAUT:
        SWAP    D5
        CMP.B   #2,D5
        BEQ COIN_HG
        CMP.B   #1,D5
        BEQ COIN_HD
        JSR CASES_INF
        JSR CASES_GD
        BRA FIN_CMP_BORD

BORD_BAS:
        SWAP    D5
        CMP.B   #2,D5
        BEQ COIN_BG
        CMP.B   #1,D5
        BEQ COIN_BD
        JSR CASES_SUP
        JSR CASES_GD
        BRA FIN_CMP_BORD

COIN_HG:
        ADD.W   #1,D0
        MOVE.W  D0,(A2)+
        ADD.W   NB_COLONNES,D0
        MOVE.W  D0,(A2)+
```

```
        SUB.W   #1,D0
        MOVE.W  D0,(A2)+
        BRA FIN_CMP_BORD

COIN_HD:
        SUB.W   #1,D0
        MOVE.W  D0,(A2)+
        ADD.W   NB_COLONNES,D0
        MOVE.W  D0,(A2)+
        ADD.W   #1,D0
        MOVE.W  D0,(A2)+
        BRA FIN_CMP_BORD

COIN_BG:
        ADD.W   #1,D0
        MOVE.W  D0,(A2)+
        SUB.W   NB_COLONNES,D0
        MOVE.W  D0,(A2)+
        SUB.W   #1,D0
        MOVE.W  D0,(A2)+
        BRA FIN_CMP_BORD

COIN_BD:
        SUB.W   #1,D0
        MOVE.W  D0,(A2)+
        SUB.W   NB_COLONNES,D0
        MOVE.W  D0,(A2)+
        ADD.W   #1,D0
        MOVE.W  D0,(A2)+
        BRA FIN_CMP_BORD

BORD_G:
        MOVE.W  D6,D0
        SUB.W   NB_COLONNES,D0
        MOVE.W  D0,(A2)+
        ADD.W   #1,D0
        MOVE.W  D0,(A2)+
        ADD.W   NB_COLONNES,D0
        MOVE.W  D0,(A2)+
        ADD.W   NB_COLONNES,D0
        MOVE.W  D0,(A2)+
        SUB.W   #1,D0
        MOVE.W  D0,(A2)+
        BRA FIN_CMP_BORD
```

```
BORD_D:
        MOVE.W  D6,D0
        SUB.W   NB_COLONNES,D0
        MOVE.W  D0,(A2)+
        SUB.W   #1,D0
        MOVE.W  D0,(A2)+
        ADD.W   NB_COLONNES,D0
        MOVE.W  D0,(A2)+
        ADD.W   NB_COLONNES,D0
        MOVE.W  D0,(A2)+
        ADD.W   #1,D0
        MOVE.W  D0,(A2)+
        BRA FIN_CMP_BORD

ADD_BORD_HAUT:
        ADD.B  #2,D5
        BRA FIN_TEST_BORD_HB

ADD_BORD_BAS:
        ADD.B  #1,D5
        BRA FIN_TEST_BORD_HB

ADD_BORD_GAUCHE:
        ADD.B  #2,D5
        BRA FIN_TEST_BORD_GD

ADD_BORD_DROIT:
        ADD.B  #1,D5
        BRA FIN_TEST_BORD_GD

CASES_SUP:                      * i dans D6.W, retourne les cases sup dans (A2)
        MOVE.L  #0,D0
        MOVE.W  D6,D0
        SUB.W   NB_COLONNES,D0
        MOVE.W  D0,(A2)+
        SUB.W   #1,D0
        MOVE.W  D0,(A2)+
        ADD.W   #2,D0
        MOVE.W  D0,(A2)+
        RTS
```

```
CASES_INF:                          * i dans D6.W, retourne les cases INF dans (A2)
          MOVE.L  #0,D0
          MOVE.W  D6,D0
          ADD.W   NB_COLONNES,D0
          MOVE.W  D0,(A2)+
          SUB.W   #1,D0
          MOVE.W  D0,(A2)+
          ADD.W   #2,D0
          MOVE.W  D0,(A2)+
          RTS

CASES_GD:
          ADD.W   #1,D6
          MOVE.W  D6,(A2)+
          SUB.W   #2,D6
          MOVE.W  D6,(A2)+
          ADD.W   #1,D6
          RTS

CASES_HB:
          ADD.W   NB_COLONNES,D6
          MOVE.W  D6,(A2)+
          SUB.W   NB_COLONNES,D6
          SUB.W   NB_COLONNES,D6
          MOVE.W  D6,(A2)+
          ADD.W   NB_COLONNES,D6
          RTS
```

*** Fichier : BIB_AFFICHEUR**

```
*-------------------------------------------------------

AFFICHE_CHRONO:
          MOVE.L  COULEUR_SEG,D1
          JSR SET_PEN_COLOR
          MOVE.B  #3,D1
          JSR WIDTH_PEN
          MOVE.W  CHRONO,D7
          MOVE.W  OX_AFFICHEUR_2,OX_AFFICHEUR_ACTUEL
BOUCLE_CHRONO:
          JSR CMP_AFFICHEUR
          RTS
```

```
AFFICHE_SCORE:
        MOVE.L  COULEUR_SEG,D1
        JSR SET_PEN_COLOR
        MOVE.B  #3,D1
        JSR WIDTH_PEN
        MOVE.W  NB_MINES,D7
        MOVE.W  OX_AFFICHEUR_1,OX_AFFICHEUR_ACTUEL
        JSR CMP_AFFICHEUR
        RTS

CMP_AFFICHEUR:
        AND.L   #$0000FFFF,D7
        DIVU    #100,D7
        JSR EFFACER_CENTAINE
CMP_AFFICHEUR_CENTAINE:
        CMP.B   #0,D7
        BEQ CENTAINE_ZERO
        CMP.B   #1,D7
        BEQ CENTAINE_UN
        CMP.B   #2,D7
        BEQ CENTAINE_DEUX
        CMP.B   #3,D7
        BEQ CENTAINE_TROIS
        CMP.B   #4,D7
        BEQ CENTAINE_QUATRE
        CMP.B   #5,D7
        BEQ CENTAINE_CINQ
        CMP.B   #6,D7
        BEQ CENTAINE_SIX
        CMP.B   #7,D7
        BEQ CENTAINE_SEPT
        CMP.B   #8,D7
        BEQ CENTAINE_HUIT
        CMP.B   #9,D7
        BEQ CENTAINE_NEUF
FIN_CMP_AFFICHEUR_CENTAINE:
        SWAP    D7
        AND.L   #$0000FFFF,D7
        DIVU    #10,D7
        JSR EFFACER_DIZAINE
CMP_AFFICHEUR_DIZAINE:
        CMP.B   #0,D7
        BEQ DIZAINE_ZERO
        CMP.B   #1,D7
```

```
        BEQ DIZAINE_UN
        CMP.B   #2,D7
        BEQ DIZAINE_DEUX
        CMP.B   #3,D7
        BEQ DIZAINE_TROIS
        CMP.B   #4,D7
        BEQ DIZAINE_QUATRE
        CMP.B   #5,D7
        BEQ DIZAINE_CINQ
        CMP.B   #6,D7
        BEQ DIZAINE_SIX
        CMP.B   #7,D7
        BEQ DIZAINE_SEPT
        CMP.B   #8,D7
        BEQ DIZAINE_HUIT
        CMP.B   #9,D7
        BEQ DIZAINE_NEUF
FIN_CMP_AFFICHEUR_DIZAINE:
        SWAP    D7
        AND.L   #$0000FFFF,D7
        JSR EFFACER_UNITE
CMP_AFFICHEUR_UNITE:
        CMP.B   #0,D7
        BEQ UNITE_ZERO
        CMP.B   #1,D7
        BEQ UNITE_UN
        CMP.B   #2,D7
        BEQ UNITE_DEUX
        CMP.B   #3,D7
        BEQ UNITE_TROIS
        CMP.B   #4,D7
        BEQ UNITE_QUATRE
        CMP.B   #5,D7
        BEQ UNITE_CINQ
        CMP.B   #6,D7
        BEQ UNITE_SIX
        CMP.B   #7,D7
        BEQ UNITE_SEPT
        CMP.B   #8,D7
        BEQ UNITE_HUIT
        CMP.B   #9,D7
        BEQ UNITE_NEUF
FIN_CMP_AFFICHEUR_UNITE:
        RTS
```

```
EFFACER_CENTAINE:      * définit la position de l'afficheur sur centaine et efface
            MOVE.L  #$00000000,D1
            JSR SET_PEN_COLOR
            MOVE.W  OX_AFFICHEUR_ACTUEL,D0
            ADD.W   DX_NUM_AFFICHEUR,D0
            MOVE.W  D0,OX_AFFICHEUR
            JSR AFF_HUIT
            MOVE.L COULEUR_SEG,D1
            JSR SET_PEN_COLOR
            RTS
EFFACER_DIZAINE:         * définit la position de l'afficheur sur dizaine et efface
            MOVE.L  #$00000000,D1
            JSR SET_PEN_COLOR
            MOVE.W  OX_AFFICHEUR,D0
            ADD.W   LON_SEG,D0
            ADD.W   DX_NUM_AFFICHEUR,D0
            MOVE.W  D0,OX_AFFICHEUR
            JSR AFF_HUIT
            MOVE.L COULEUR_SEG,D1
            JSR SET_PEN_COLOR
            RTS
EFFACER_UNITE:           * définit la position de l'afficheur sur unité et efface
            MOVE.L  #$00000000,D1
            JSR SET_PEN_COLOR
            MOVE.W  OX_AFFICHEUR,D0
            ADD.W   LON_SEG,D0
            ADD.W   DX_NUM_AFFICHEUR,D0
            MOVE.W  D0,OX_AFFICHEUR
            JSR AFF_HUIT
            MOVE.L COULEUR_SEG,D1
            JSR SET_PEN_COLOR
            RTS

UNITE_ZERO:
            JSR AFF_ZERO
            BRA FIN_CMP_AFFICHEUR_UNITE
UNITE_UN:
            JSR AFF_UN
            BRA FIN_CMP_AFFICHEUR_UNITE
UNITE_DEUX:
            JSR AFF_DEUX
            BRA FIN_CMP_AFFICHEUR_UNITE
```

```
UNITE_TROIS:
        JSR AFF_TROIS
        BRA FIN_CMP_AFFICHEUR_UNITE
UNITE_QUATRE:
        JSR AFF_QUATRE
        BRA FIN_CMP_AFFICHEUR_UNITE
UNITE_CINQ:
        JSR AFF_CINQ
        BRA FIN_CMP_AFFICHEUR_UNITE
UNITE_SIX:
        JSR AFF_SIX
        BRA FIN_CMP_AFFICHEUR_UNITE
UNITE_SEPT:
        JSR AFF_SEPT
        BRA FIN_CMP_AFFICHEUR_UNITE
UNITE_HUIT:
        JSR AFF_HUIT
        BRA FIN_CMP_AFFICHEUR_UNITE
UNITE_NEUF:
        JSR AFF_NEUF
        BRA FIN_CMP_AFFICHEUR_UNITE
DIZAINE_ZERO:
        JSR AFF_ZERO
        BRA FIN_CMP_AFFICHEUR_DIZAINE
DIZAINE_UN:
        JSR AFF_UN
        BRA FIN_CMP_AFFICHEUR_DIZAINE
DIZAINE_DEUX:
        JSR AFF_DEUX
        BRA FIN_CMP_AFFICHEUR_DIZAINE
DIZAINE_TROIS:
        JSR AFF_TROIS
        BRA FIN_CMP_AFFICHEUR_DIZAINE
DIZAINE_QUATRE:
        JSR AFF_QUATRE
        BRA FIN_CMP_AFFICHEUR_DIZAINE
DIZAINE_CINQ:
        JSR AFF_CINQ
        BRA FIN_CMP_AFFICHEUR_DIZAINE
DIZAINE_SIX:
        JSR AFF_SIX
        BRA FIN_CMP_AFFICHEUR_DIZAINE
```

```
DIZAINE_SEPT:
        JSR AFF_SEPT
        BRA FIN_CMP_AFFICHEUR_DIZAINE
DIZAINE_HUIT:
        JSR AFF_HUIT
        BRA FIN_CMP_AFFICHEUR_DIZAINE
DIZAINE_NEUF:
        JSR AFF_NEUF
        BRA FIN_CMP_AFFICHEUR_DIZAINE
CENTAINE_ZERO:
        JSR AFF_ZERO
        BRA FIN_CMP_AFFICHEUR_CENTAINE
CENTAINE_UN:
        JSR AFF_UN
        BRA FIN_CMP_AFFICHEUR_CENTAINE
CENTAINE_DEUX:
        JSR AFF_DEUX
        BRA FIN_CMP_AFFICHEUR_CENTAINE
CENTAINE_TROIS:
        JSR AFF_TROIS
        BRA FIN_CMP_AFFICHEUR_CENTAINE
CENTAINE_QUATRE:
        JSR AFF_QUATRE
        BRA FIN_CMP_AFFICHEUR_CENTAINE
CENTAINE_CINQ:
        JSR AFF_CINQ
        BRA FIN_CMP_AFFICHEUR_CENTAINE
CENTAINE_SIX:
        JSR AFF_SIX
        BRA FIN_CMP_AFFICHEUR_CENTAINE
CENTAINE_SEPT:
        JSR AFF_SEPT
        BRA FIN_CMP_AFFICHEUR_CENTAINE
CENTAINE_HUIT:
        JSR AFF_HUIT
        BRA FIN_CMP_AFFICHEUR_CENTAINE
CENTAINE_NEUF:
        JSR AFF_NEUF
        BRA FIN_CMP_AFFICHEUR_CENTAINE

SET_ZERO_AFFICHEUR:          * place l'origine du repère de l'afficheur
        MOVE.W  OX_AFFICHEUR,D1
        MOVE.W  OY_AFFICHEUR,D2
        RTS
```

```
SEG_A:
        JSR SET_ZERO_AFFICHEUR
        MOVE.W  D1,D3
        ADD.W   LON_SEG,D3
        MOVE.W  D2,D4
        JSR DRAW_LINE
        RTS

SEG_B:
        JSR SET_ZERO_AFFICHEUR
        ADD.W   LON_SEG,D1
        MOVE.W  D1,D3
        MOVE.W  D2,D4
        ADD.W   LON_SEG,D4
        JSR DRAW_LINE
        RTS

SEG_C:
        JSR SET_ZERO_AFFICHEUR
        ADD.W   LON_SEG,D1
        MOVE.W  D1,D3
        ADD.W   LON_SEG,D2
        MOVE.W  D2,D4
        ADD.W   LON_SEG,D4
        JSR DRAW_LINE
        RTS
SEG_D:
        JSR SET_ZERO_AFFICHEUR
        MOVE.W  D1,D3
        ADD.W   LON_SEG,D3
        ADD.W   LON_SEG,D2
        ADD.W   LON_SEG,D2
        MOVE.W  D2,D4
        JSR DRAW_LINE
        RTS
SEG_E:
        JSR SET_ZERO_AFFICHEUR
        MOVE.W  D1,D3
        ADD.W   LON_SEG,D2
        MOVE.W  D2,D4
        ADD.W   LON_SEG,D4
        JSR DRAW_LINE
        RTS
```

```
SEG_F:
        JSR SET_ZERO_AFFICHEUR
        MOVE.W  D1,D3
        MOVE.W  D2,D4
        ADD.W   LON_SEG,D4
        JSR DRAW_LINE
        RTS
SEG_G:
        JSR SET_ZERO_AFFICHEUR
        MOVE.W  D1,D3
        ADD.W   LON_SEG,D3
        ADD.W   LON_SEG,D2
        MOVE.W  D2,D4
        JSR DRAW_LINE
        RTS

AFF_ZERO:
        JSR SEG_A
        JSR SEG_B
        JSR SEG_C
        JSR SEG_D
        JSR SEG_E
        JSR SEG_F
        RTS
AFF_UN:
        JSR SEG_B
        JSR SEG_C
        RTS
AFF_DEUX:
        JSR SEG_A
        JSR SEG_B
        JSR SEG_G
        JSR SEG_E
        JSR SEG_D
        RTS
AFF_TROIS:
        JSR SEG_A
        JSR SEG_B
        JSR SEG_C
        JSR SEG_D
        JSR SEG_G
        RTS
```

```
AFF_QUATRE:
        JSR SEG_F
        JSR SEG_G
        JSR SEG_B
        JSR SEG_C
        RTS
AFF_CINQ:
        JSR SEG_A
        JSR SEG_F
        JSR SEG_G
        JSR SEG_C
        JSR SEG_D
        RTS
AFF_SIX:
        JSR SEG_A
        JSR SEG_C
        JSR SEG_D
        JSR SEG_E
        JSR SEG_F
        JSR SEG_G
        RTS
AFF_SEPT:
        JSR SEG_A
        JSR SEG_B
        JSR SEG_C
        RTS
AFF_HUIT:
        JSR SEG_A
        JSR SEG_B
        JSR SEG_C
        JSR SEG_D
        JSR SEG_E
        JSR SEG_F
        JSR SEG_G
        RTS

AFF_NEUF:
        JSR SEG_A
        JSR SEG_B
        JSR SEG_C
        JSR SEG_D
        JSR SEG_F
        JSR SEG_G
        RTS
```