BAUD Thierry
ESIL, Dépt. informatique
Luminy – Case 925
13288 Marseille Cedex 9
Tél. : 04.91.82.85.10
Fax : 04.91.82.85.11

Marseille, Le 11 septembre 2001

Rapport de stage de 2ième année

Lunar Cheese Technologie

66 rue Villon, 69008 Lyon
Tél. : 04.37.90.53.53
Fax : 04.37.90.53.54

Table des matières

Introduction

L'objet de mon stage était :

Le développement en C, C++ et l'utilisation des bibliothèques graphiques 3D pour le développement d'un jeu.

Il s'est déroulé sur 2 mois. J'ai travaillé pour une entreprise spécialisée dans le jeu vidéo.

Mon stage a été axé principalement sur l'élaboration d'un code source. Je disposais d'une grande autonomie vis-à-vis du code et des fonctionnalités implantées à partir du moment ou le chef du projet approuvait les choix.

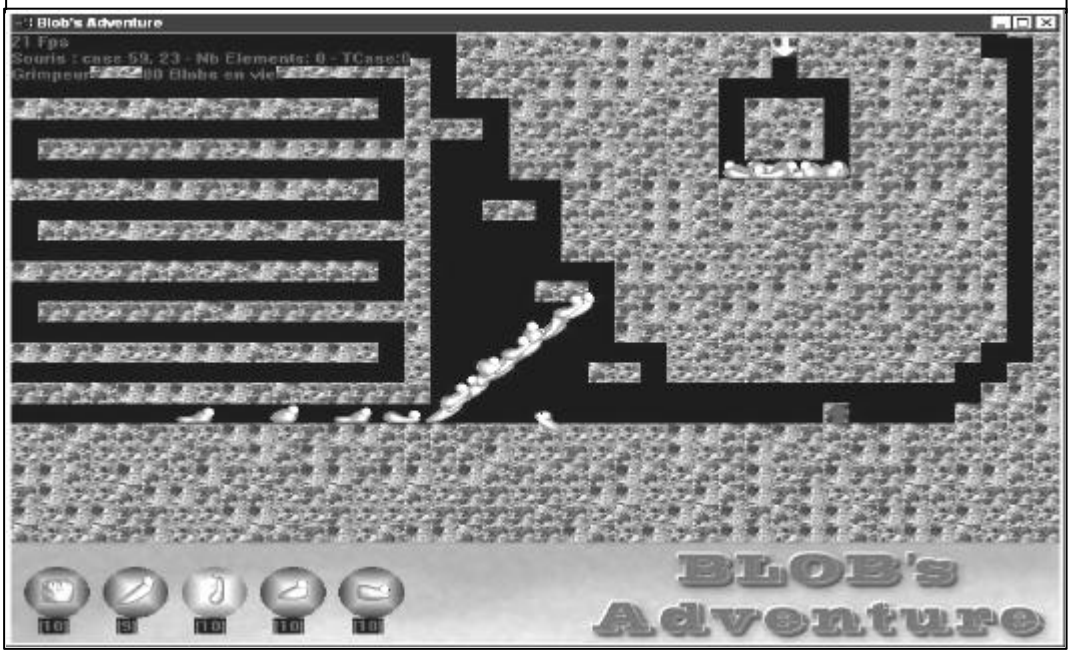
Présentation de l'entreprise

Lunar Cheese Technologies est une entreprise qui ne possède qu'un seul employé : M. Nicolas Cazagou. Cette entreprise est spécialisée dans le développement de page web et dans le développement de jeux.

Pour chaque nouveau projet le chef d'entreprise regroupe une équipe de développement soit par CDD soit par stage.

A. Présentation du projet

Figure 0



A.1. Un seul héros pour plusieurs jeux : Un blob

L'idée du projet était de développer un certain nombre de jeux autour d'un même héros, rapide à développer. Pour ne pas perdre trop de temps sur le scénario, les jeux devaient être basés sur des petits casse-tête (puzzle-games). Nous avions comme seul impératif de développement que les jeux fonctionnent sous Windows et soit rapide et léger.

Avantages du point de vue de la conception d'un jeu :

- Beaucoup de graphismes sont réutilisables.
- De plus nous avons essayé de rendre le développement beaucoup plus rapide en créant des classes réutilisables et le plus général possible.
- Ainsi Le développement d'un jeu se libère des initialisations fastidieuses pour la gestion des fenêtres, du DirectX et des périphériques.

Nous avons créer des classes qui nous ont permis d'initialiser l'affichage : avec le DirectX d'ouvrir une fenêtre (en mode fenêtré ou plein écran avec tous les choix de résolution).

Mon rôle dans ce projet était de développer l'un de ces jeux avec un infographiste.

A.2. Présentation du jeu : une copie de «lemmings »

Dans ce jeu, l'utilisateur est au contrôle d'une troupe de blob. Il doit en leur donnant des ordres les faire traverser un labyrinthe pour les faire parvenir à la sortie.

B. Du point de vue de la programmation : Les bases du jeu

Il s 'agit ici d'un jeu de plate-forme en 2 dimensions ou chaque élément graphique occupe une case. Pour des raisons de rapidité de développement et surtout pour pouvoir faire des testes de collisions efficaces, nous avons opté pour une représentation du jeu sous forme de tableau.

B.1. Les classes Jeu, Souris et Clavier :

La gestion de la fenêtre et des périphériques (clavier, souris). Ces classes sont en fait une interface avec la bibliothèque DirectX (DirectDraw et DirectInput) de Microsoft.

B.2. Classe Niveau :

Dans cette classe nous avons fait toute la gestion des variables nécessaires pour un niveau du jeu. Les fonctions implémentées dans cette classe sont les suivantes :

- Elles initialisent le chargement du niveau.
- Elles gèrent un écran virtuel et permettent de déplacer la zone affichée.
- Elles gèrent le tableau de jeu où tous les éléments graphiques sont stockés.

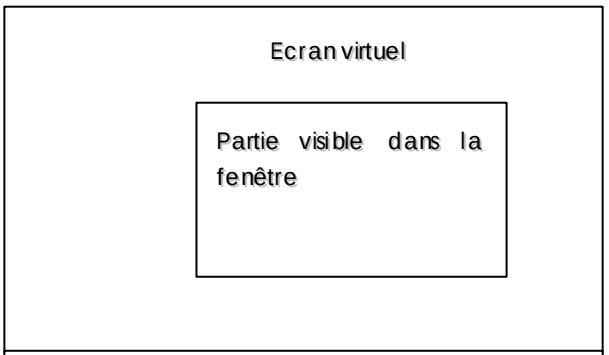


Figure 1

B.3. La gestion d'un tableau :

Dans la classe niveau nous avons rajouté un gros tableau «d'élément du jeu ».

Mais ceci pose un problème du point de vue des déplacements des objets animés : Savoir dans quelle case se trouve un blob qui est en train de se déplacer. Nous avons choisi de stocker le blob dans toutes les cases touchées par celui-ci.

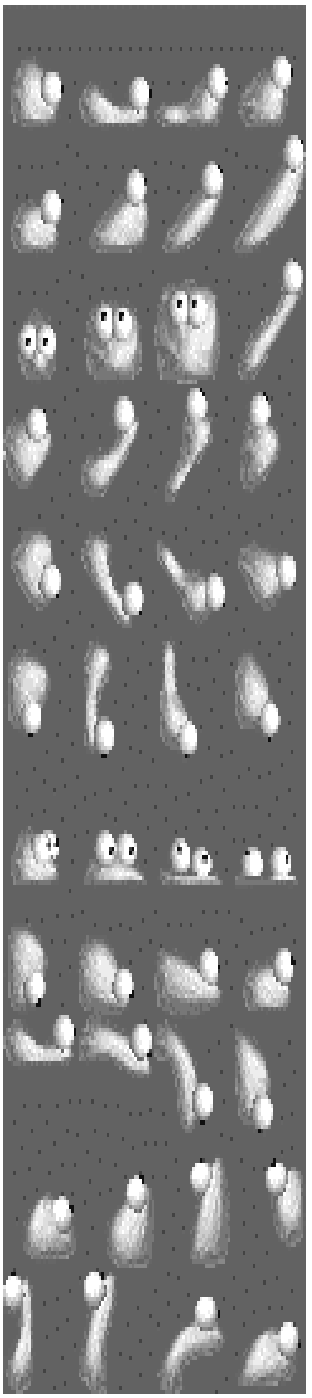
Ainsi dans chaque case de ce tableau il y a une liste d'éléments graphiques qui se trouvent à cette position ou qui chevauche cette case.

B.4. Classe ElementGraphique :

Chaque élément du jeu est représenté dans le code par une classe :

- La classe Terrain
- La classe EntréeBlob.
- La classe Blob
- La classe Bonus : Quand un blob rentre en collision avec un bonus il l'active.
- Toutes ces classes possèdent leur données et **leurs méthodes de sauvegarde et de chargement propre**. Mais elles découlent toutes de la classe Elément Graphique.
- Cette classe très générique permet de charger une image ou une animation (Cf. Figure) et de positionner cet élément dans l'écran virtuel (i.e. : c'est dans cette classe que les cordonnées de l'objet sont stockées).

Figure 2



C. Du point de vue de la programmation : Le jeu en lui même

La partie de programmation spécifique de ce jeu était de pouvoir faire «vivre » une grande quantité de vers (blob) dans un labyrinthe. Environ une centaine. Pour cela j'ai du réécrire plusieurs fois les parties critiques du code. Pour m'aider dans le développement j'ai utiliser les outils de Visual Studio tel que le profiler et Visual Source Safe.

Le Profiler permet de connaître dans quelle partie du programme l'exécution se trouve le plus souvent. Je me suis surtout servi du profiler pour écrire la classe blob. Comme je le disais plus haut cette classe devait pouvoir être instantiée une centaine de fois par niveau et nécessite une bonne quantité de calcule. Elle doit entre autre gérer l'animation d'un blob selon sont état, et gérer la mise à jour d'un blob (i.e. il doit savoir ce qu'il doit faire en fonction des cases environnantes)

Visual Source Safe est un outil d'aide au développement de projet multi-utilisateurs qui permet de tenir un historique des changements effectués sur le code. Ceci nous permettait de pouvoir tester des nouveaux algorithmes ou simplement quelques changements sur le code tout en gardant les anciennes versions qui fonctionnaient. Ce logiciel facilite grandement la gestion des versions d'un logiciel.

C.1. La classe Blob :

La classe blob dérive de la classe Element Graphique et rajoute la gestion des états d'un blob.

La position d'un blob est repérée de deux façon :

- Par ses coordonnées en pixel
- Par les coordonnées de la case qu'il occupe (ou de la case supérieur gauche s'il en occupe plusieurs cf figure 3)
- Ceci permet d'effectuer un **déplacement fluide** tout en gardant des repères pour faciliter les collisions.

La classe blob est en fait l'implémentation d'un automate d'états finis. Ceci nous a permis de simuler chaque états de la vie d'un blob. Les états retenus sont :

- Avancer horizontalement dans 2 directions (gauche ou droite).
- Avancer Diagonalement.
- Grimper : Le blob reste coller au terrain (il grimpe au mur et au plafond).
- Sauter. Pour cela il faut un blob « tremplin » qui le face sauter.
- Construire une diagonale fixe. Les blobs se montent les uns sur les autres pour former un pont en diagonal.
- Bloquer. Le blob se transforme en un gros bloc qui interdit le passage.
- Creuser horizontalement.
- Creuser vers le bas.

Pour écrire le code nous avons eu recours a des variables qui stockaient les pointeurs des fonctions d'états. Par exemple selon l'état la fonction update est différente ce qui veut dire que l'on a mis une variable Fonct_Update dans laquelle on stocke le pointeur de la fonction correspondant à l'état. Exemple :Fonct_Update = Grimpe_Update.

C.2.La gestion des collisions :

Après avoir développé un système de stockage adapté au jeu (le tableau) nous nous sommes attaqués à la gestion des déplacements des blobs et en particuliers aux collisions entre un Elément du jeu et le décor. Pour cela plusieurs choix se sont présenter à nous :

Vu que chaque Elément du jeu est stocké dans un tableau nous devions juste tester si les cases occupées par les blobs n'étaient pas aussi occupées par du terrain.

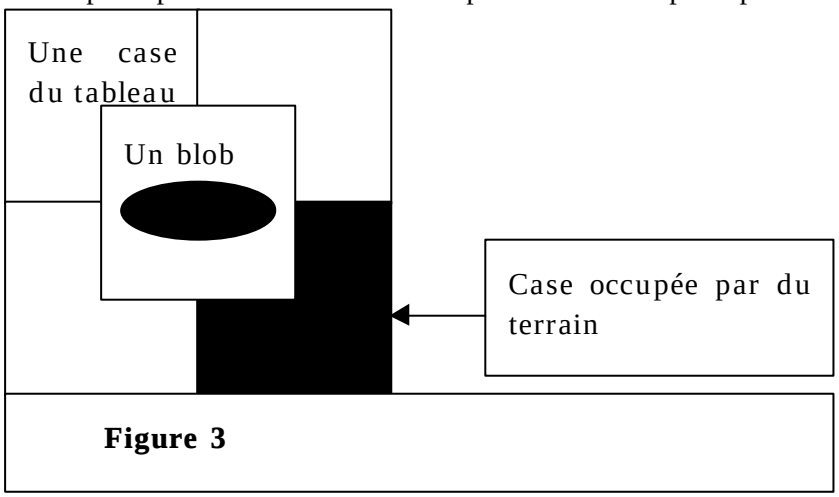


Figure 3

C.2.1. Tester dans la direction du blob :

Pour ce test nous avons besoin de savoir dans quelle direction le blob se déplace. Ce qui permet de faire le test dans la case en avant.

Algorithme lors de la mise à jour d'un blob :

- Avant d'avancer, on teste s'il y a une collision dans la direction ou l'on souhaite aller
- S'il y a collision on ne peut pas avancer.
- On traite le fait que l'on ne puisse plus avancer (Exemple : changement d'état du blob, demi-tour)

Défauts :

- Ce test limite le déplacement des blobs à des directions simples. Dés que le blob doit aller en diagonale en chevauchant 4 cases il faut tester au moins trois cases dans la direction du blob pour savoir s'il est en collision.
- Il faut réécrire les testes pour chaques directions testées.

Avantages :

- Ce test permet de tester la case en dessous du blob pour savoir s'il est en contact avec le sol.

C.2.2. Tester toutes les cases occupées par le blob :

Algorithme lors de la mise à jour d'un blob :

- On avance le blob dans sa direction
- On teste les éventuelles collisions et s'il y a une collision on revient ou on était
- On traite le fait que l'on ne puisse plus avancer (Exemple : changement d'état du blob)
-

Défauts :

- Ce test possède un défaut c'est que l'on doit faire les tests du sol dans la fonction de mis à jour. (Si le blob est aligné sur 2 cases horizontales on ne peut pas voir les case en dessous)
- Cette fonction teste au maximum 4 cases au environs du blob.

Avantages :

- Pour ce test nous avons besoin de connaître la position. Ce test permet de ne pas se soucier dans quel sens le blob se déplace. Il peut avoir des mouvements complexes.
- La gestion des différents états d'un blob :

D. Ce qu'il reste à faire

Le plus gros du travail qu'il reste à faire ce situe au niveau du gameplay et du design des niveaux. Il faudrait, maintenant que les compétences des blobs sont terminées, les utilisées pour faire des niveaux avec une difficulté croissante.

Le développement du code est en grande partie terminée. Le code qui existe déjà fonctionne sans bugs connus et offre la possibilité de jouer sur un niveau du jeu.

Malgré cela, la programmation n'est pas terminée :

- Il faudrait ajouter un écran de démarrage avec quelques options et la possibilité de choisir le niveau.
- Rendre les classes niveau et élément graphique beaucoup plus générales pour pouvoir les réutiliser dans un autre jeu à base de tableau. (Permettre d'utiliser des éléments graphiques d'une taille supérieure à une case).

Conclusion

Je suis content d'avoir fait un premier pas dans l'univers du développement de jeux, j'ai rencontré beaucoup de gens qui y sont depuis longtemps et ils m'ont tous dit que maintenant cet univers tend à devenir comme l'univers du cinéma ou seul compte la présentation du produit le temps mis pour le développer et l'argent investi dedans.

Annexe A : Le journal de bord du développement

```
********************************************************************************
** Blob's Adventure
** Journal de bord
********************************************************************************

********************************************************************************
** 04.08.01 :

********************************************************************************
Profile: Function timing, sorted by time
Date:    Sat Aug 04 14:16:36 2001


Program Statistics
------------------
    Command line at 2001 Aug 04 14:15: "D:\User\Lunar Cheese\Blob's
Adventure\Blobs"
    Total time: 43617,051 millisecond
    Time outside of functions: 3071,765 millisecond
    Call depth: 6
    Total functions: 41
    Total hits: 170567
    Function coverage: 70,7%
    Overhead Calculated 4
    Overhead Average 4

Module Statistics for blobs.exe
-------------------------------
    Time in module: 40545,286 millisecond
    Percent of time in module: 100,0%
    Functions in module: 41
    Hits in module: 170567
    Module function coverage: 70,7%

       Func          Func+Child           Hit
       Time   %         Time      %      Count  Function
-----------------------------------------------------------
   22363,750  55,2    22363,750  55,2     1025 CJeu::Draw(void) (jeu.obj)
    5068,011  12,5     8632,484  21,3        1 CJeu::Initialisation(void) (jeu.obj)
    3253,151   8,0     3274,837   8,1        1 CJeu::~CJeu(void) (jeu.obj)
    2706,701   6,7     2706,701   6,7     2705 CElementJeu::CElementJeu(class CNiveau *) (elementjeu.obj)
    2148,656   5,3     2148,689   5,3        1 CJeu::InitD3DX(void) (jeu.obj)
    1940,918   4,8     4114,191  10,1     1025 CJeu::Update(int) (jeu.obj)
    1332,851   3,3     1332,851   3,3    50311 CElementJeu::SetPosPixel(float,float) (elementjeu.obj)
     957,236   2,4      957,236   2,4     5706 CElementJeu::SetPosCase(int,int) (elementjeu.obj)
     432,869   1,1     2002,933   4,9    52618 CEJBlob::Update(int) (ejblob.obj)
     226,844   0,6      247,338   0,6    49325 CElementJeu::Collision(int) (elementjeu.obj)
      58,600   0,1       58,600   0,1     2607 CElementJeu::Chargement(struct _iobuf * *) (elementjeu.obj)
      14,916   0,0      129,271   0,3       94 CEJBlob::CEJBlob(class CNiveau *) (ejblob.obj)
      10,774   0,0       11,227   0,0       94 CEJBlob::`scalar deleting destructor'(unsigned int) (ejblob.obj)
      10,774   0,0       11,227   0,0       94 CEJBlob::`vector deleting destructor'(unsigned int) (ejblob.obj)
       7,452   0,0        7,452   0,0     2705 CElementJeu::~CElementJeu(void) (elementjeu.obj)
       4,644   0,0       13,813   0,0     1013 CEJBlob::ChangeEtat(int,int) (ejblob.obj)
       2,357   0,0        3,055   0,0        1 CJeu::ReleaseD3DX(void) (jeu.obj)
       2,288   0,0       20,494   0,1      606 CEJBlob::ConsequenceCollision(int,int,int,int) (ejblob.obj)
       0,597   0,0        0,597   0,0        1 CSouris::`scalar deleting destructor'(unsigned int) (jeu.obj)
       0,407   0,0        0,407   0,0      373 CEJBlob::DemiTour(void) (ejblob.obj)
       0,341   0,0        0,341   0,0      154 CEJBlob::FaireUneAnimDeTransition(int,int) (ejblob.obj)
       0,292   0,0        0,292   0,0        1 CJeu::CJeu(int,int,bool) (jeu.obj)
       0,247   0,0        0,676   0,0        4 CElementJeu::`scalar deleting destructor'(unsigned int)
(elementjeu.obj)
       0,247   0,0        0,676   0,0        4 CElementJeu::`vector deleting destructor'(unsigned int)
(elementjeu.obj)
       0,207   0,0        0,453   0,0       94 CEJBlob::~CEJBlob(void) (ejblob.obj)
       0,101   0,0        0,101   0,0        1 CClavier::`scalar deleting destructor'(unsigned int) (jeu.obj)
       0,034   0,0        0,034   0,0        1 CJeu::InitRenderer(void) (jeu.obj)
       0,011   0,0     3274,848   8,1        1 CJeu::`scalar deleting destructor'(unsigned int) (jeu.obj)
       0,011   0,0     3274,848   8,1        1 CJeu::`vector deleting destructor'(unsigned int) (jeu.obj)
```

```
********************************************************************************
** 01.08.01 :
      - Quelques reflexions sur la structure du programme
            - Enlever les variables globales Tableau et tcase et le mettre
                  dans niveau
            - quand on cree un niveau on le charge et quand on change de niveau
                  on le detruis
            - on ne devrait pas avoir acces a la classe concept en dehors de
niveau
                  niveau devrait gerer:
                  - la mise a jour du tableau
                  - la creation et la suppression d'element du jeu
                  - si on veux avoir la liste des elements du jeux present
                        sur la case (x,y)

                        en gros on devrait faire une grosse classe niveau qui
permettrait
                        de gerer tous les jeux a base de tableau et que ce soit
invisible
                        de l'exterieur: on aurait juste des fonctions qui
permettrait
                        de recuperer les elements sur une case, de creer un
elements et de
                        supprimer un element.

            - jeux devrait comporter l'initialisation de la fenetre
                  de l'interface du jeux avant de demarrer et le choix du niveau
                  puis c'est lui qui lancerait en fonction de ca la creation
                  d'un niveau
            - jeux devrait aussi comporter les pointeurs sur souris et clavier

********************************************************************************
** 28.07.01 : Conclusion du profilage:
      -     il ya trop de setpos il faut en enlever pour en garder au maximum 1
(ou 2)
      par update. (en gros il faut reduire de moitier le nbre de setpos).
      ( en faire un seul par changement d'etat qui arrive
      assez frequement dans update)

********************************************************************************
** 28.07.01
Profile: Function timing, sorted by time
Date:    Sat Jul 28 14:44:54 2001


Program Statistics
------------------
    Command line at 2001 Jul 28 14:42: "D:\User\Lunar Cheese\Blob's
Adventure\Blobs"
    Total time: 131805,364 millisecond
    Time outside of functions: 125923,158 millisecond
    Call depth: 6
    Total functions: 33
    Total hits: 313917
    Function coverage: 57,6%
    Overhead Calculated 5
    Overhead Average 5

Module Statistics for blobs.exe
-------------------------------
    Time in module: 5882,206 millisecond
```

```
    Percent of time in module: 100,0%
    Functions in module: 33
    Hits in module: 313917
    Module function coverage: 57,6%

       Func          Func+Child           Hit
       Time   %         Time      %      Count  Function
-----------------------------------------------------
    2474,472  42,1     2474,472  42,1    77200 CElementJeu::MetAJourTableau(float,float) (elementjeu.obj)
    1253,051  21,3     4468,673  76,0    77701 CEJBlob::Update(int) (ejblob.obj)
    1164,543  19,8     1164,543  19,8     1127 CElementJeu::CElementJeu(void) (elementjeu.obj)
     522,022   8,9      533,117   9,1    74934 CElementJeu::Collision(int) (elementjeu.obj)
     429,099   7,3     2686,969  45,7    75724 CElementJeu::SetPosPixel(float,float) (elementjeu.obj)
      11,188   0,2       30,901   0,5     1085 CElementJeu::Chargement(struct _iobuf * *) (elementjeu.obj)
       7,505   0,1        7,505   0,1     1127 CElementJeu::~CElementJeu(void) (elementjeu.obj)
       5,629   0,1      202,518   3,4     2832 CElementJeu::SetPosCase(int,int) (elementjeu.obj)
       4,275   0,1        4,698   0,1       40 CEJBlob::`scalar deleting destructor'(unsigned int) (ejblob.obj)
       4,275   0,1        4,698   0,1       40 CEJBlob::`vector deleting destructor'(unsigned int) (ejblob.obj)
       3,189   0,1        9,041   0,2     1008 CEJBlob::ChangeEtat(int,int) (ejblob.obj)
       2,044   0,0       42,310   0,7       40 CEJBlob::CEJBlob(void) (ejblob.obj)
       0,339   0,0        0,339   0,0      264 CEJBlob::DemiTour(void) (ejblob.obj)
       0,164   0,0       11,096   0,2      524 CEJBlob::ConsequenceCollision(int,int,int,int) (ejblob.obj)
       0,139   0,0        0,139   0,0      220 CEJBlob::FaireUneAnimDeTransition(int,int) (ejblob.obj)
       0,103   0,0        0,109   0,0        2 CElementJeu::`scalar deleting destructor'(unsigned int)
(elementjeu.obj)
       0,103   0,0        0,109   0,0        2 CElementJeu::`vector deleting destructor'(unsigned int)
(elementjeu.obj)
       0,050   0,0        0,422   0,0       40 CEJBlob::~CEJBlob(void) (ejblob.obj)
       0,015   0,0        0,191   0,0        7 CEJBlob::ClickSouris(void) (ejblob.obj)

*****************************************************************************
** 27.07.01
- Le pb avec le passage en mode editeur a ete resolu:
      ce n'était pas un problème avec la souris malgré les apparence
      on ne réallouait pas la souris. C'était du a une variable temporaire
      de char (dans draw de jeu.cpp) qui était de type static char[64]
      et ou je stockait un truc plus grand que 64 => le pointeur souris
      était changer chez moi.....?????? !!!!!! WWWWWWWW
      je l'ai mis a 1024 et tout marche

- il y avait aussi un pb avec le type des elements du jeu
      le terain etait de type ELEMENTJEU_BASE maintenant il est
      de type ELEMENTJEU_DECORS (ca avait créé un pb lors des sauvegarde des
niveaux
      et du chargement: C maintenant resolue)

*****************************************************************************
** 20.07.01
Version 2

DONE:
- nvlle gestion de souris (Bouton droit + glisser = deplacement)
- debuggage des procedures draw (on a multiplie par 3 les fps!!!!!)
      cf draw dans niveau.cpp
- ajout de l'etat creuse horizontal mais pas encore fini
- debuggage de bcp de petites procedures du jeu notamment celle de l'init du
niveau

TODO:
- Il reste des pb lors du passage au mode editeur les changements dans l'init
font planter le jeux .... ????
      (pb avec les pointeurs de la classe de la souris ???)

*****************************************************************************
** 15.07.01
un autre profilage  a montrer que la fonction draw était trop appelée
je vais chercher pkoi ......

*****************************************************************************
```

```
** 10.07.01
Les commentaires des nouvelles seront stockée a l'aide de source safe dans les
commentaire de la version.....

********************************************************************************
** 04.07.01
Profile: Function timing, sorted by time
Date:    Wed Jul 04 14:27:19 2001

Program Statistics
------------------
    Command line at 2001 Jul 04 14:25: "D:\User\Lunar Cheese\Blob's
Adventure\Blobs"
    Total time: 109861,731 millisecond
    Time outside of functions: 105138,644 millisecond
    Call depth: 6
    Total functions: 33
    Total hits: 177463
    Function coverage: 57,6%
    Overhead Calculated 4
    Overhead Average 4

Module Statistics for blobs.exe
-------------------------------
    Time in module: 4723,087 millisecond
    Percent of time in module: 100,0%
    Functions in module: 33
    Hits in module: 177463
    Module function coverage: 57,6%

       Func          Func+Child           Hit
       Time   %         Time      %      Count  Function
-----------------------------------------------------------
    2352,649  49,8     3769,305  79,8    42030 CEJBlob::Update(int) (ejblob.obj)
    1218,503  25,8     1218,503  25,8    43638 CElementJeu::MetAJourTableau(float,float) (elementjeu.obj)
     784,918  16,6      784,918  16,6      885 CElementJeu::CElementJeu(void) (elementjeu.obj)
     208,832   4,4      254,543   5,4    40770 CElementJeu::Collision(int) (elementjeu.obj)
     107,790   2,3     1175,483  24,9    42342 CElementJeu::SetPosPixel(float,float) (elementjeu.obj)
      18,729   0,4       45,711   1,0     1640 CEJBlob::ConsequenceCollision(int,int,int,int) (ejblob.obj)
      10,976   0,2       20,230   0,4      843 CElementJeu::Chargement(struct _iobuf * *) (elementjeu.obj)
       5,696   0,1       13,384   0,3     1322 CEJBlob::ChangeEtat(int,int) (ejblob.obj)
       4,211   0,1      145,768   3,1     1977 CElementJeu::SetPosCase(int,int) (elementjeu.obj)
       2,924   0,1        3,031   0,1       40 CEJBlob::`scalar deleting destructor'(unsigned int) (ejblob.obj)
       2,924   0,1        3,031   0,1       40 CEJBlob::`vector deleting destructor'(unsigned int) (ejblob.obj)
       2,467   0,1        2,467   0,1      885 CElementJeu::~CElementJeu(void) (elementjeu.obj)
       0,992   0,0       38,656   0,8       40 CEJBlob::CEJBlob(void) (ejblob.obj)
       0,822   0,0        0,822   0,0      461 CEJBlob::Proche(int,int) (ejblob.obj)
       0,449   0,0        0,449   0,0      503 CEJBlob::DemiTour(void) (ejblob.obj)
       0,084   0,0        0,289   0,0        2 CElementJeu::`scalar deleting destructor'(unsigned int)
(elementjeu.obj)
       0,084   0,0        0,289   0,0        2 CElementJeu::`vector deleting destructor'(unsigned int)
(elementjeu.obj)
       0,026   0,0        0,106   0,0       40 CEJBlob::~CEJBlob(void) (ejblob.obj)
       0,009   0,0        0,257   0,0        3 CEJBlob::ClickSouris(void) (ejblob.obj)

-> Il faut regarder si la gestion des listes d'elements ne peut pas etre
      reduite.(ou appelee moins souvent => faire moins de SetPos...)
-> La fonction update est obligatoirement lourde: il faut chercher a
      faire le moins de teste possible.

********************************************************************************
** 03-07-01
- Ajout de la gestion d'un ecran virtuel
      (Deplacement avec le bouton droit de la souris)
      Fixe quelque bug dans la gestion du tableau d'element
- Ajout de l'etat blob saute
      Pb: la hauteur du saut depend de "milli" => desactive pour l'instant.
      on verra plustard. (Le probleme doit surement etre dans les
      restrictions sur les vitesses (cf CEJBlob::Update)
```

```
- Ajout de l'etat blob tremplin (en fait on devra utiliser
      les blob diagonal fixe pour faire le tremplin).


*****************************************************************************
** 29-06-01
- Abandon de la gestion de la souris par un thread
      (Probleme de l'ordonnanceur de windows ou pb de priorite)
      impossible de faire rafraichir la souris a 20 fps le thread
      ne prend pas la main (meme avec un timer)
      Il faudrait en plus de ca changer la facon de dessiner la souris.
      (Utilisation du GDI plutot que directX)

*****************************************************************************
** 26-06-01
** TODO:

1- Faire des blocs de terrain en biais pour que l'on puisse creuser en diagonal
et marcher sur les pentes.
(il seront stockes dans le tableau comme des TDG ou TDD)

2- Faire des anims pour chaque transition d'etats
ex:   * grimpeur a besoin de rotation a 90°
      * quand un blob passe de GlisseHorizontal->tombe
            on devrait faire une anim comme celle du grimpeur (rotation a 90°)
      * quand passe de Glisse Diagonal->GlisseHorizontal idem

3- faire des blocs de terrain arrondis sur les bords pour faire moins
impression de cases.A rajouter dans la fonction de draw de l'element_Terrain
      si pas de bloc a cote dessiner un bloc arrondi sinon faire un bloc droit

4- Faire des bloc d'eau (pour le nageur)
Animation de l'eau? (facile? setanimation et commenceanimation)

5- Pourquoi ne pas faire bouger le terrain (comme si on etait dans un arbre
avec du vent?
But du jeu on est des pucerons et on doit ramasser le poellene (ou la seve)
dans les fleurs et le ramener au nid
-> Branche qui oscille un peu (animation du terrain).
-> Eau sur les feuilles.
-> differents graphismes branches, feuilles, fleurs.

6-    Etat Saute pour le faire sauter une marche
      Etat pont pour que les blobs fassent un pont (idem que glisse diagonal)

7- Faire Dessiner la souris 20 fois par seconde (timer)
pour que meme si ca rame la souris soit fluide.


*****************************************************************************
** 25-06-01
- EJGoutte->EJBlob
- EntreeGouttes->EntreeBlob
- Reecriture de la classe EJBlob
      (fonctionne maintenant avec des fonctions d'etats)
- Ajout de l'effacement d'une case en mode editeur
- Etats creer :
      * GlisseHorizontale
      * GlisseDiagonale
      * Tombe
      * Meurt
```

```
      * Grimpe
      * BlocFixe
      * DiagonaleFixe

*******************************************************************************
** 10-06-01
-ajout du flip
-ajout de freeimage de ressource manager
-sauvegarde

*******************************************************************************
** 06-06-01
-optimisation de cconcept.

*******************************************************************************
** 04-06-01
-gestion simple des collisions,
-etat de base de la goutte

*******************************************************************************
** 03.06.01
-environnement de developpement
-classes ElementJeu, Goutte et EntreeGouttes.
-systeme d'animations
```