Tutoriel	**AmigaCracking : TOKI**
Protection	**MFM (multifile) + CRC**
Auteur Original	**Heavy aka VrS!-Cybfree**
Source original	**http://cyberpingui.free.fr**
Version	**04/06/2018 Gi@nts**
Vérification/Correction	**V1.1, Testé et fonctionnel de A à Z**

* TOKI CRACK TUTORIEL *

Table des matières

Matériels nécessaires 3
Général Info 3
Agencement des disquettes Amiga v1.2 5
WinUAE 8
Part 1 X-copy 9
Part 2 Analyse de l'image IPF 10
Part 3 Comportement des chargements du jeu et cheat mode/trainer 11
Part 4 Analyse et modification du bootblock 12
Part 5 Analyse du TrackLoader du bootblock 13
Part 6 Conclusion analyse du TrackLoader du bootblock 17
Part 7 Sauvegarde des pistes custom part #1 – Pre Game 21
Part 8 Analyse du TrackLoader principal 29
Part 9 Sauvegarde des pistes custom part #2 – In Game 36
Part 10 Patch du fichier 16-25, credits/logo Ocean 38
Part 11 Code - Bootsecteur 39
Part 12 Code - Décompacteur 41
Part 13 Code - Trackloader 44
Part 14 Code - TokiStarter 49
Part 15 Construction de notre disquette 52
Part 16 Test – part #1 54
Part 17 Première protection - Patch Level2 ne fonctionne pas 56
Part 18 Seconde protection - Patch Level1 bug graphique sur TIMER<1 minute 62
Part 19 Troisième protection - Patch général bug graphique au bout de 8 minutes 67
Part 20 Modification du code tokistarter et de notre disk hacké 72

Matériels nécessaires

1) Un AMIGA avec une extension mémoire de 512K ou l'émulateur WINUAE
2) Un lecteur externe en plus est fortement recommandé.
3) Une Carte ACTION REPLAY (ou ça ROM Image) selon configuration utilisé.
4) Le jeu Original Toki ou son image CAPS française (SPS 0040)
5) * Outils de décompression/compression pour les formats **ByteKiller 1.3** et **Pack Ice 2.11** (ex : Suite XFD)
6) * Outils 'sector' : **Mega-mon 3.2**, SectorTrasher, etc
7) Le logiciel Xcopy-pro en disquette ou son image disk.
8) L'assembleur ASM-One
9) Les connaissances pour utiliser un l'assembleur ci-dessus !

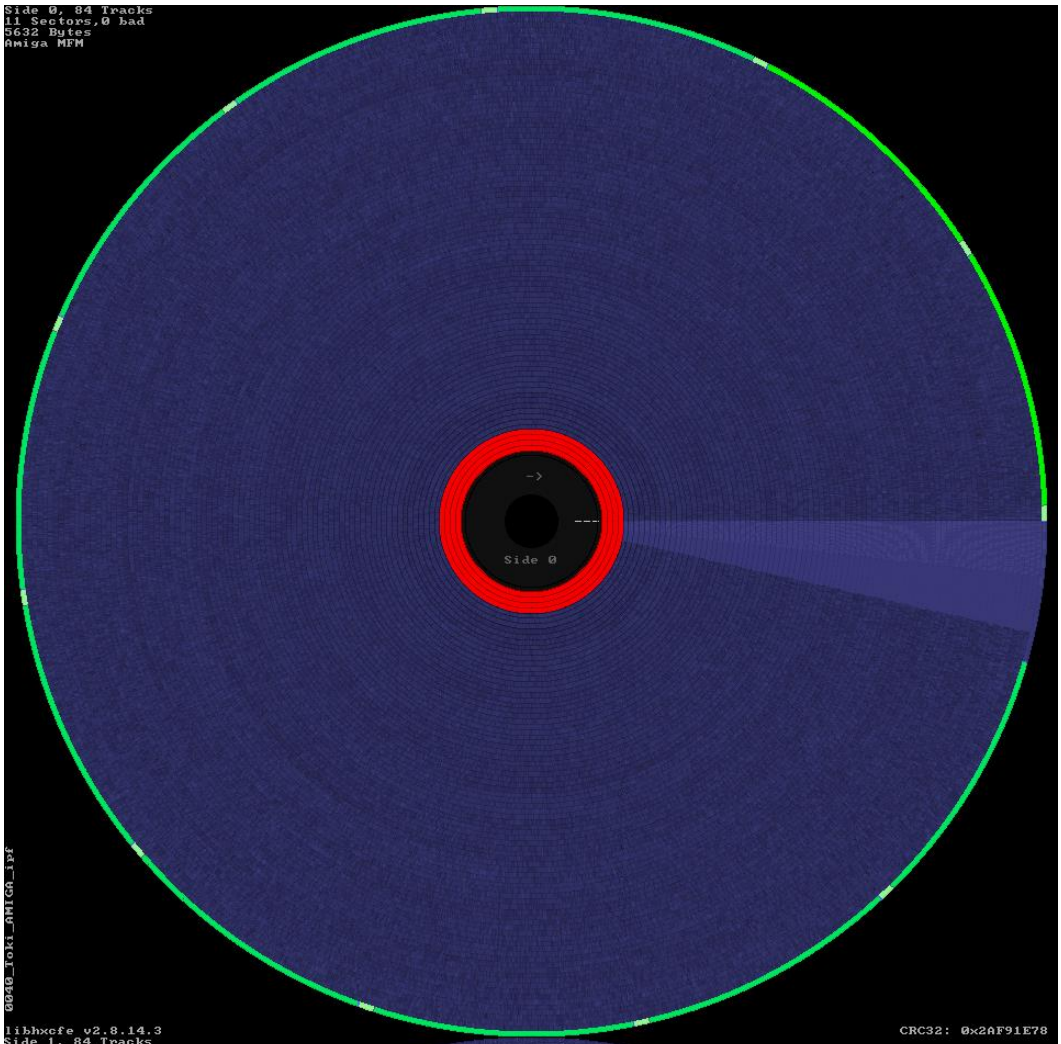
- Dans ce tuto sera utilisé **mega-mon 3.**2 disponible sur la compilation PD LSD 009
- Dans ce tuto sera utilisé la suite **XFD** disponible sur aminet mais il existe beaucoup de compilation contenant des décompacteur capable de décompresser les formats bytekiller et Ice 2.11

Général Info

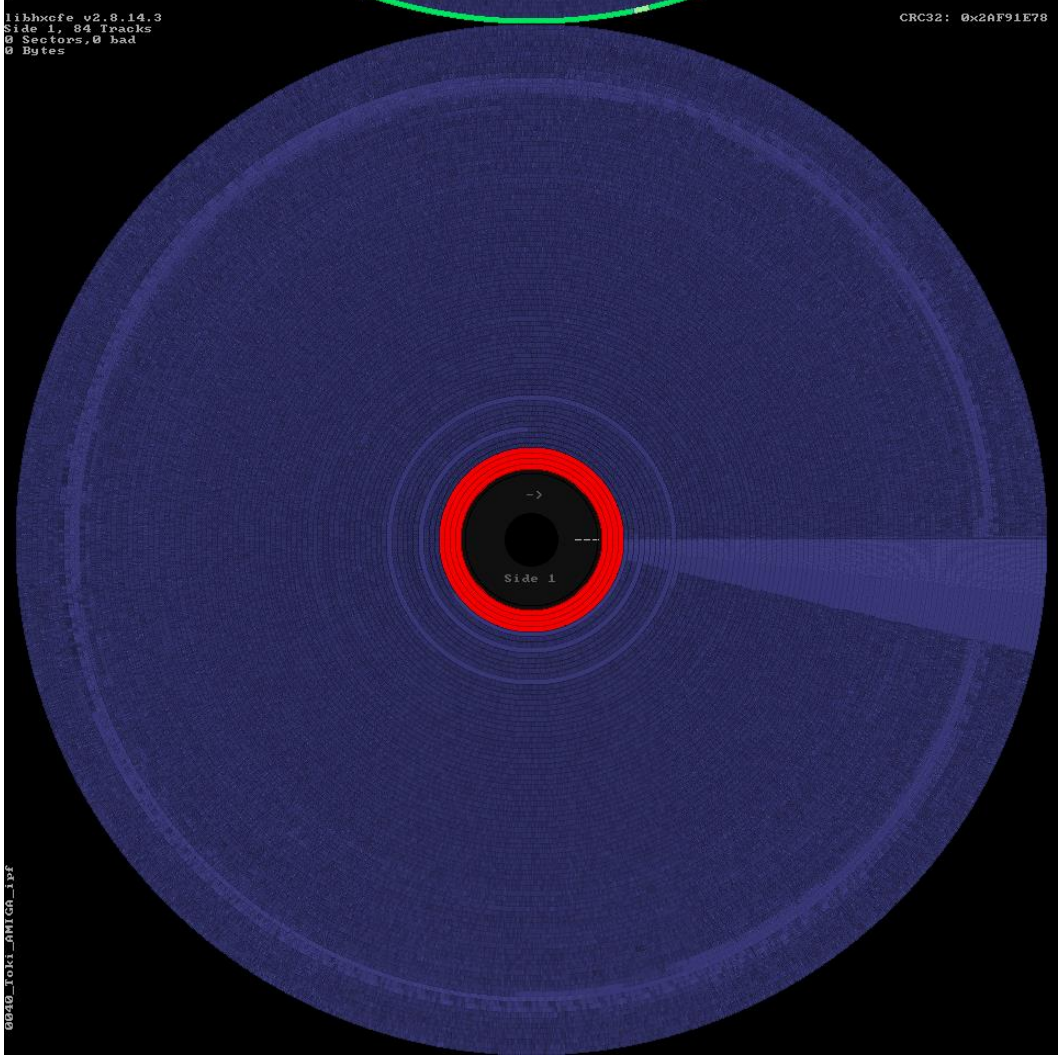
Ce tutoriel Français est basé sur le tutoriel original mis en ligne sur le site **The Legacy** *http://cyberpingui.free.fr/tuto_crk4.htm*
Ce document n'est pas une traduction mot par mot de celui-ci mais plus une nouvelle version, plus détaillé et avec parfois une nouvelle approche de la problématique mais toujours basé sur le travail effectuée par **Heavy.**
Un grand merci à son auteur **Heavy aka VrS!-Cybfree** qui m'a apporté son aide afin de bien comprendre la problématique.

Bon Tuto.

Gi@nts


Side 0, 84 Tracks
11 Sectors,0 bad
5632 Bytes
Amiga MFM
->

Side 0
0040_Toki_AMIGA_ipf


libhxcfe v2.8.14.3
Side 1, 84 Tracks
0 Sectors,0 bad
0 Bytes
CRC32: 0x2AF91E78
->

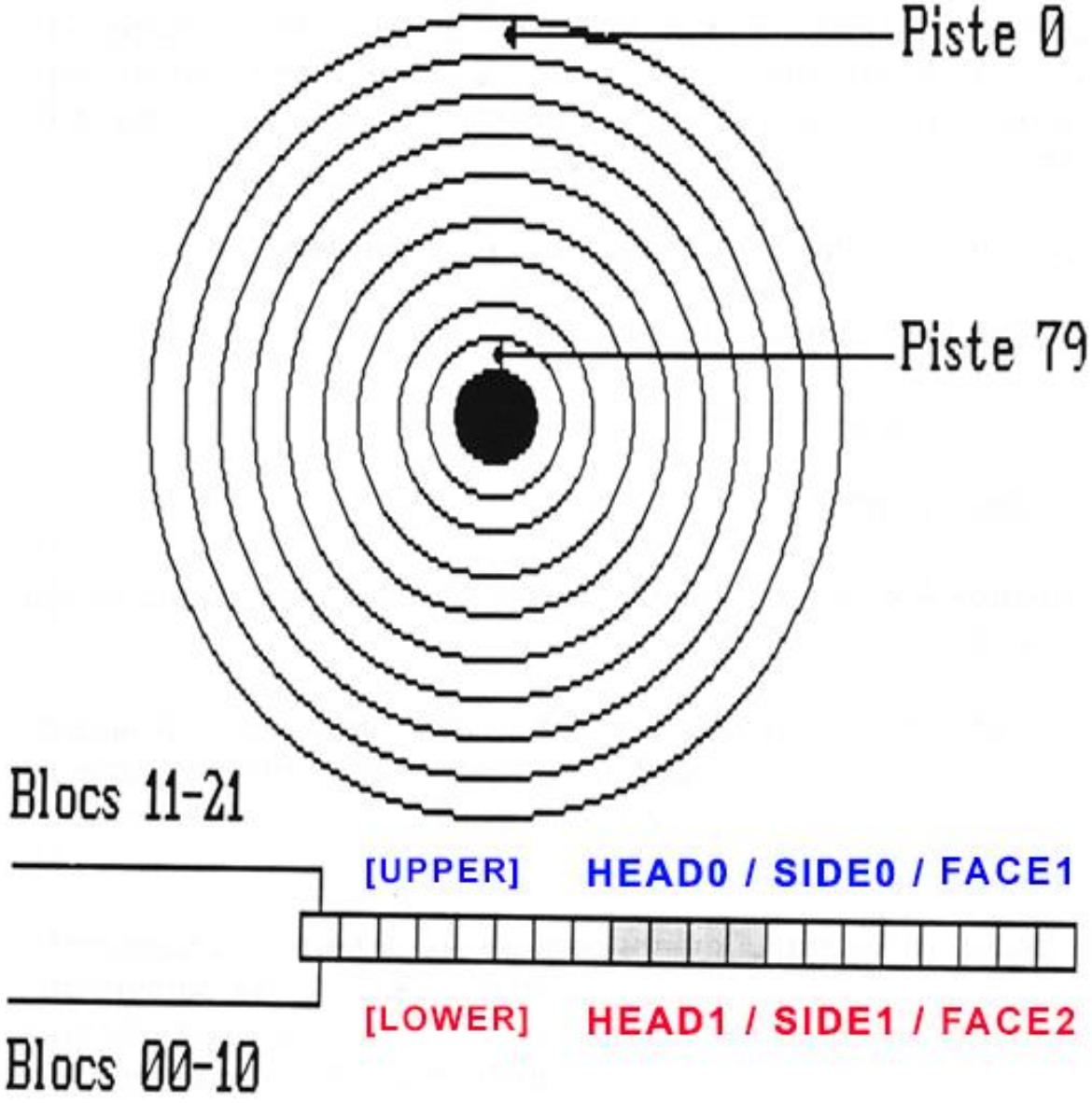
Side 1
0040_Toki_AMIGA_ipf

Agencement des disquettes Amiga v1.2

En France :

On utilise des termes comme : *piste, bloc, secteur, face*…
Piste : 0 à 79 Certaine disquette pousse jusqu'à 81 voire 82 pistes mais le standard reste quand même 80 pistes (de 0 à 79)
Face : 0/1 ou **1/2** ou **A/B**, Dessus ou dessous tout simplement. Sur Amiga nous avons deux faces utilisées sur 99% des jeux.

Chaque piste, pour un format standard 'AmigaDOS' est composée de plusieurs *bloc ou secteur*, en général 11 **par face**.
Le terme piste peut désigner l'ensemble d'une piste (les deux 'side' du disque), ou uniquement une 'side' d'une piste.
Une piste standard *amigados* est découpée en plusieurs partie appelé **bloc**, **secteur**, **sector**.



Dans d'autre pays :

On utilisera d'autre terme, comme **sector, keys, tracks, cylindre**, **head**…

Le terme **track** par exemple que l'on aurait vite fait de traduire 'piste' ne colle pas forcément à notre description française.
En général, le terme **tracks** désigne toujours une position sur la disquette mais **elle va de 0 à 159** (soit 160 **tracks**)
Le maximum étant 160 et non 80 car on a deux faces bien sûres, en fait, elle correspond à une piste sur une face.

Il peut néanmoins arriver que l'on utilise dans des tuto anglo-saxon le terme *tracks* dans le sens 'piste' en français
(donc de 0 à 79 et non de 0 à 159). Mais en règle générale, il a plutôt une plage de 0 à 160.

C'est le terme **cylindre** qui 'colle' plus à notre définition française de **piste.**
En effet, il est courant d'utiliser le terme **cylindre** pour désigner une position sur la disquette de 0 à 79.

Le terme **sector** ou **key** quant à lui corresponds au terme français **bloc** ou **secteur.**

Sur une disquette au format ***Amigados***, nous avons 880ko et nous avons 11 secteurs par face, par piste.

La taille d'une piste ayant une valeur physiquement maximum.
Le nombre maximum de **sector** sur une piste dépends assez logiquement de la taille de ses **sectors.**

Pref…beaucoup de terme qui ne sont pas forcément utilisé dans leur sens propre, le mieux est de lire un tuto et de comprendre quel sens l'auteur a voulu leurs donner.

Il existe aussi un autre type d'appellation utilisé par exemple par le logiciel **MFM-Warp** de Ferox*
**C'est un programme qui scan le disque en bas niveau et essaye d'en réaliser une copie.*

Track	Calcul	Résultat	Format utilisé sous MFMWarp
0	0/2	0 et pair	0.0
1	1/2	0 et impair	0.1
2	2/2	1 et pair	1.0
3	3/2	1 et impair	1.1
156	156/2	78 et pair	78.0
157	157/2	78 et impair	78.1
158	158/2	79 et pair	79.0
159	159/2	79 et impair	79.1

On notera que :

Le premier secteur (secteur 0) appelé aussi *bootbloc* commence sur la *lowerSide* en piste 00
et se fini en piste 79 sur le *upperside*

En *tracks* c'est le même système sauf que l'on terminera en Track 179 et non 79.

La piste Zero est celle situé le plus à l'extérieure du disque.

Le 1er secteur logique, donc le premier bloc sur la disquette, se trouve **piste 0 secteur 0**
Les *bloc* se suivent physiquement mais ne sont pas forcément ordonnée, on parle aussi d'entrelacement.

Le bloc 11 (si on part de 0 bien sur) n'est pas le 1er secteur de la seconde piste mais le 1er secteur *de la face suivante.*
(voir image ci-dessus)

En format **Amigados**, **la taille d'un secteur est de 512 octets**
Ce qui nous donnes comme taille disponible : 512*11 secteurs*80 pistes*2 faces = 901 120 octets soit 880Ko
Une 'track' AmigaDos a une taille de 512 * 11 = **5632** en décimal soit ***$1600*** *octets*

Mise en application sous l'AR :

Il existe deux commandes sous l'AR qui permettent de charger sauver des pistes, à savoir : **RT** et **WT**
Elles fonctionnent pareil.
L'une permet la lecture, l'autre l'écriture.

#**RT** alias Read Track. Permet le chargement de donnée située sur la disquette vers la mémoire.
#la première valeur sera la track de **départ** [0 à 159] à indiquer en hexa. **/!\ ne pas confondre avec piste**

#La seconde valeur sera le nbr de demi track à copié à partir de là.

#**WT** alias Write Track. Permet la sauvegarde de donnée située en mémoire vers la disquette.

Exemples :

RT 20 1 50000
Start Track = $20 et taille à lire = 1
On copiera donc la piste !16 (en décimal) side 0 en mémoire $50000

Oui car **20** est donné en hexa, ce qui nous donnes !32 en décimal **mais** il indique une track (de 0 à 159) **PAS en piste.**
Pour avoir l'équivalent en piste on divisera donc par 2 (car deux faces).

$20/2=$10 = !16 (en décimal donc) et comme il n'y a pas de retenu on est sur la face0.

RT 21 2
Start Track = $21 et taille à lire = 2
On copiera la piste !16 side 1 et la piste !17 side 0 en mémoire 50000

21 est donné en hexa **donc $21 = !33** en décimal.
33/2 = 16.5 ,donc piste 16 side 1 et comme on continue à lire/copier les donnes (**taille à lire =2**), on continue la copie.
On change donc de track car on est déjà sur la face 1 (il existe que 2 faces sur une disquette)
On arrive donc sur la prochaine track à savoir, piste 17 en side 0 puisque que c'est la première face au niveau structure la side 0.

Les secteurs

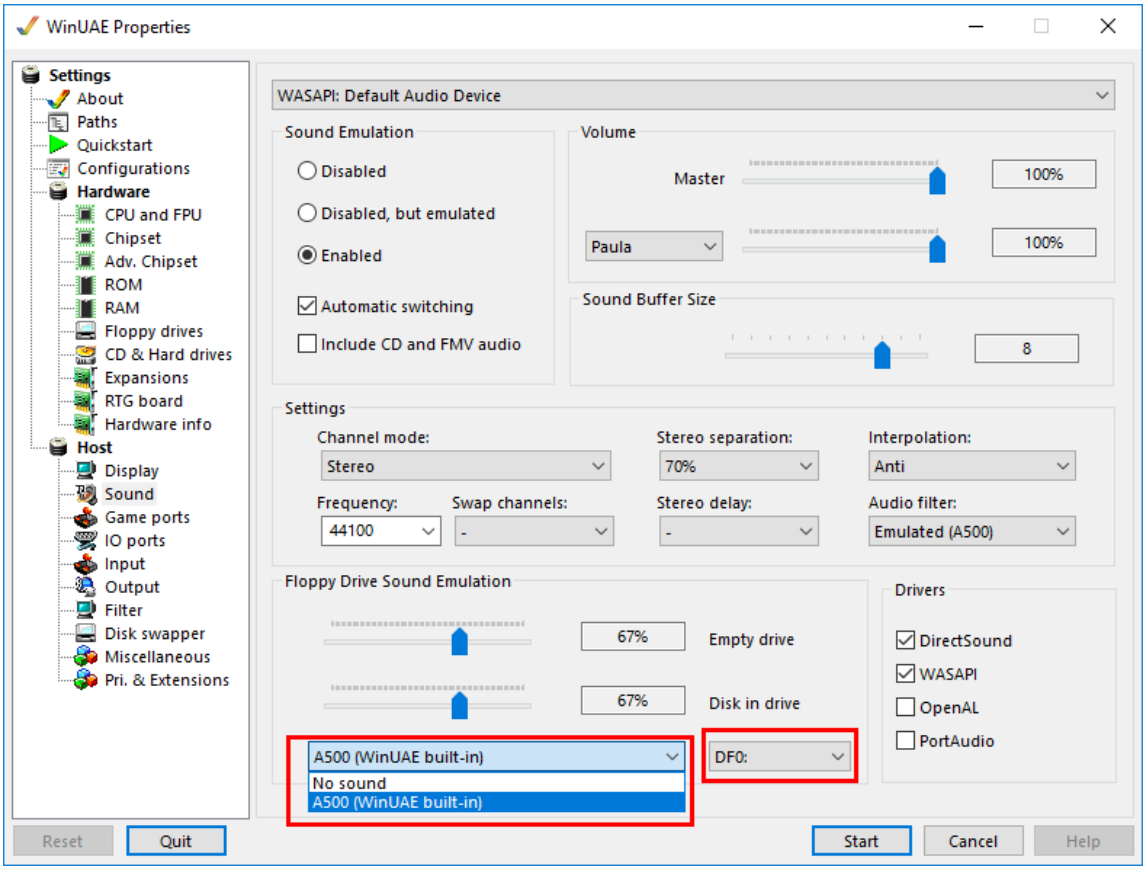
On peut aussi adresser un disque avec la notion de secteur.
Comme on l'a vue au-dessus, un disque AmigaDos fait 80 Piste (0-79), 2 faces et 11 secteurs par Piste
Chaque secteur fait 512 octets.
Si on fait le calcul cella nous donnes : 80*2*11=1760 Secteurs
Ainsi on peut avoir une position exacte en secteur sur une disque amiga avec une valeur entre 0 et 1759

Exemple DiskBlock Position 520 correspond au Secteur 03 de la piste 23 sur la face 1

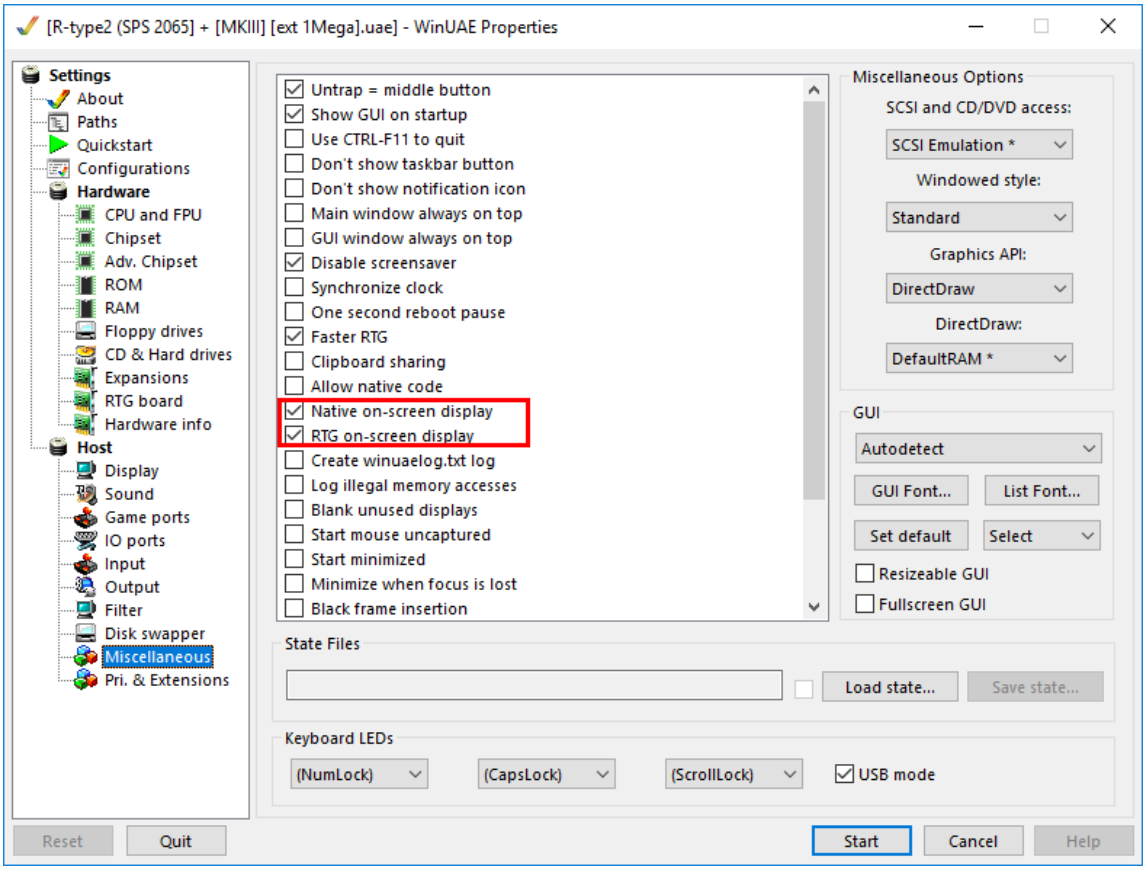
On peut aussi fonctionner en mode **RAW**, directement en adressant en octet, cela dépends du 'trackloader' en question.

WinUAE

Pour ceux qui utilisent ***winUAE*** pour ces tutoriels (j'imagine, la plupart des personnes), Je vous conseille fortement d'activer le son des lecteurs de disquette histoire d'entendre ce que le lecteur effectue comme accès.
HOST -> SOUND -> FLOPPY DRIVE SOUND EMULATION - > DF0 Built-In

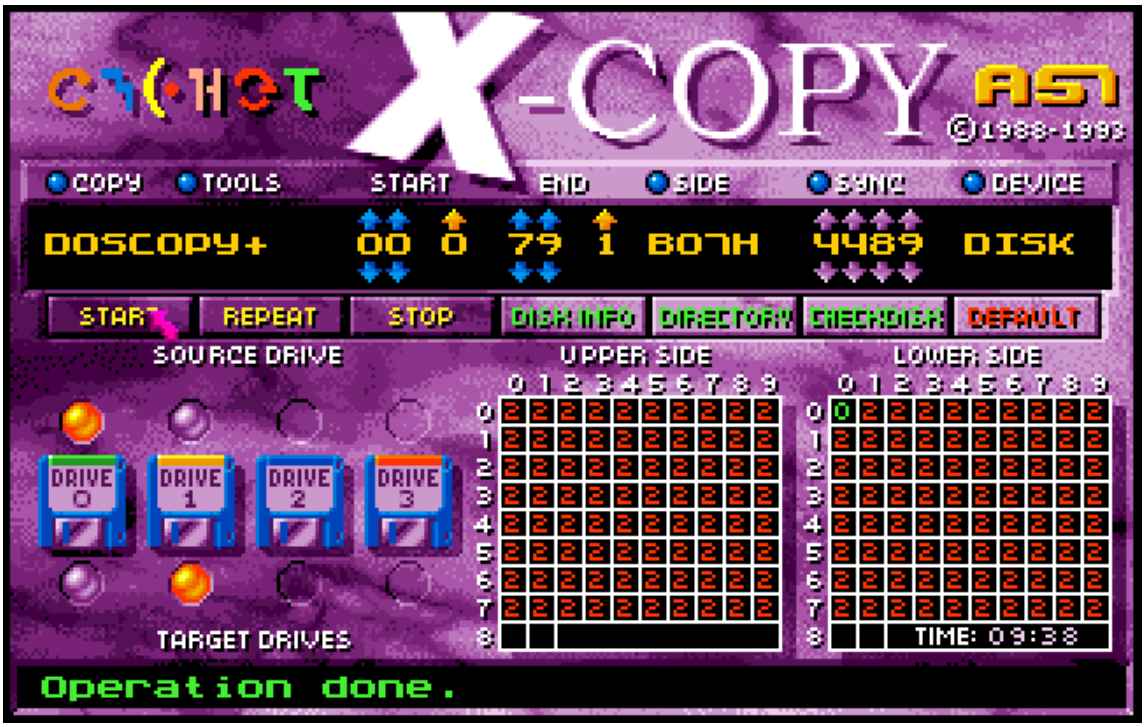


Voir même, pour plus d'information. Par exemple afficher sur qu'elle face l'on se trouve, d'activer :
Host -> Miscellaneous -> Native on-screen display AND RTG on-screen display



Part 1 X-copy

Time to check the protection
On va dans un premier temps essayer de copier les disquettes originales à l'aide de Xcopy Pro.



On passe de l'erreur 5 à 2 sur l'ensemble de la disquette, ce qui indique un format propriétaire
Hormis bien sur la 1er piste au format AmigaDOS.
On va quand même garder cette copie qui, bien sûr, ne fonctionnera pas.

Rappel des codes d'erreur de Xcopy :

1. *Less or more than 11 sectors*
2. ***No sync found***
3. *No sync after gap found*
4. *Header checksum error*
5. ***Error in header/format long***
6. *Data block checksum error*
7. *Long track*
8. *Verify error*

Part 2 Analyse de l'image IPF

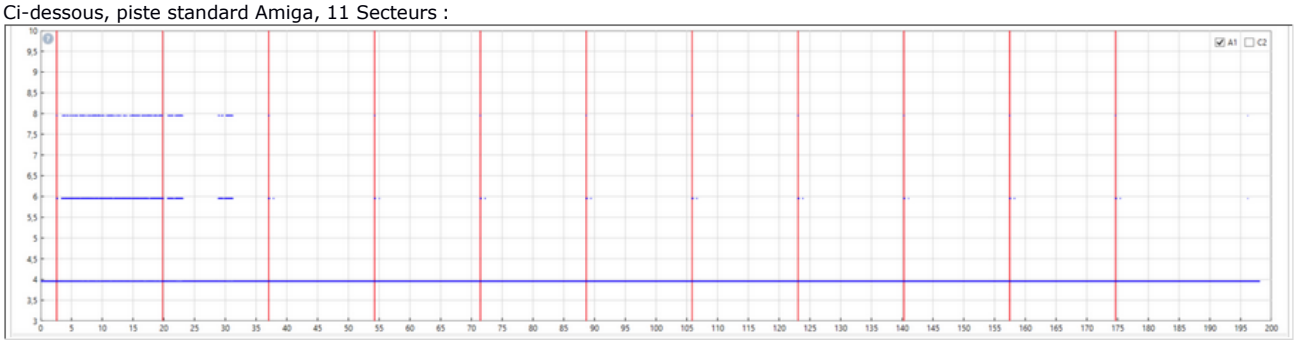
FILENAME	0040_Toki.ipf
TYPE	Floppy_Disk
ENCODER	CAPS(V1)
FILE	0040(V1)
DISK	2
TRACK	00-83
SIDE	0-1
PLATFORM	Amiga
REVOLUTION	5

Comme prévue, un format Standard *AmigaDOS* sur la 1er piste, 11 blocks standard.
Mais ensuite, toutes les pistes sont dans un format non standard avec un '*Data Length*' de 6380 bytes

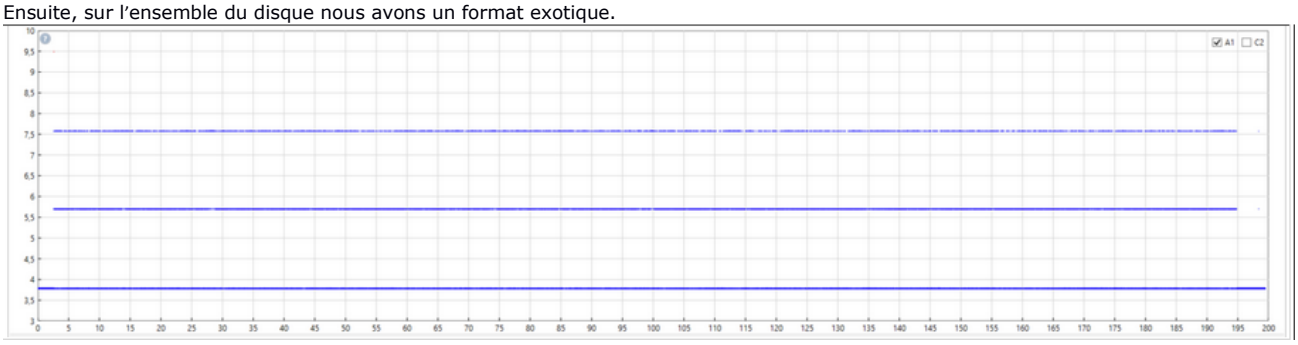
TrackNumber	Size Record (bytes)	Crc	Status	Track Size	Detail Tr. Size	Start Byte	Bit	DataKey	Block	Density	Signal	Encoder	Flag
T00.0	80	492E965E	Good	12523	100184 bits = Data=95744 + Gap=4440	303	2428	001	11	Auto	cell_2us	0	None
T00.1	80	EDBD99E1	Good	13151	105208 bits = Data=101424 + Gap=3784	4	38	002	1	Auto	cell_2us	0	None
T01.0	80	D8D003D5	Good	13141	105128 bits = Data=101424 + Gap=3704	7	62	003	1	Auto	cell_2us	0	None

TRACK		Data Length (bytes)		Data (bits)				CRC32 of the complete Extra Data Block			Adress
Data block Description	Sector ID	Data		bytes/sector	GAP		Codage	GapDef	DataOff		Adresse
		MFM bits	bytes		MFM bits	bytes			MFM bits	bytes	
[T00.0]		6446		51568				99E4B260			13576-20021
#0	0	8704	545	512	0	1	MFM	0352	0352	15	13576-13607
#1	1	8704	545	512	0	1	MFM	0906	0906	57	13608-13639
#2	2	8704	545	512	0	1	MFM	1460	1460	92	13640-13671
#3	3	8704	545	512	0	1	MFM	2014	2014	126	13672-13703
#4	4	8704	545	512	0	1	MFM	2568	2568	161	13704-13735
#5	5	8704	545	512	0	1	MFM	3122	3122	196	13736-13767
#6	6	8704	545	512	0	1	MFM	3676	3676	230	13768-13799
#7	7	8704	545	512	0	1	MFM	4230	4230	265	13800-13831
#8	8	8704	545	512	0	1	MFM	4784	4784	300	13832-13863
#9	9	8704	545	512	0	1	MFM	5338	5338	334	13864-13895
#10	10	8704	545	512	4440	278	MFM	5892	5892	369	13896-13927
[T00.1]		6380		51040				0D55FF26			20050-26429
#0	N/A	101424	6340	N/A	3784	237	MFM	0032	0032	2	20050-20081
[T01.0]		6380		51040				04423071			26458-32837
#0	N/A	101424	6340	N/A	3704	232	MFM	0032	0032	2	26458-26489
[T01.1]		6380		51040				851F2EBE			32866-39245
#0	N/A	101424	6340	N/A	3736	234	MFM	0032	0032	2	32866-32897
[T02.0]		6380		51040				61DAA787			39274-45653
#0	N/A	101424	6340	N/A	3712	233	MFM	0032	0032	2	39274-39305
[T02.1]		6380		51040				33BA9BC2			45682-52061
#0	N/A	101424	6340	N/A	3736	234	MFM	0032	0032	2	45682-45713

Ci-dessous, piste standard Amiga, 11 Secteurs :



Ensuite, sur l'ensemble du disque nous avons un format exotique.



Part 3 Comportement des chargements du jeu et cheat mode/trainer

Avant d'entrer dans le vif du sujet on va s'attarder 5 minutes à voir comment le jeu se comporte au niveau des chargements.
Insérer la disquette originale du jeu dans le lecteur et booter dessus.

Voilà ce que l'on peut déduire par rapport au bruit de chargement des pistes du lecteur de disquette.

Nbr Piste Lu	Affichage Après chargement	Piste Lue WinUAE
02	**N/A - écran noir**	77-79
01	Ecran Blanc + logo Ocean/Credits	75-76
19	**N/A - écran noir**	55-74
09	Animation du début	16-25
06	Mini animation du level	07-13
14	Level1	01-15

Noter que ce n'est pas forcément 'exact' à une unité prête car il n'est pas évident à l'oreille d'être formel sur le chargement ou pas d'une piste.
Mais ça permet d'avoir une idée des accès disque.

TRAINER :
Bon, on ne va pas faire une section ici sur comment chercher des trainers avec l'**AR**, je pense que le manuel de celle-ci est assez clair.
Voilà ce que l'on peut trouver :

$D239C9	Level	(Attention, adresse impaire, forte chance que le code passe par une lecture en Word à l'adresse D239C8)
$23CD9	Nombre de vie	Sub trouvé en 2C9F0 avec la commande **TFD 23DC9**
$278EC	Time	Sub trouvé en 370CC avec la commande **FA 278EC**
$2A17E	Nbr vie départ	'**MOVE.B #6,00023CD9**', trouvé avec la commande **FA 23CD9**

CHEAT MODE :
Nous avons un grand avantage, il se trouve qu'il existe un *cheatMode* sur ce jeu qui permet entre autres de choisir son Level.
Pour cela il suffit, pendant une partie de taper sur le clavier le mot **killer** et d'appuyer sur **F1** à **F8**
Un scrolling de couleur dans le bord de l'écran vous indiquera que le *cheatMode* est activé.

Part 4 Analyse et modification du bootblock

La seule piste lisible est la piste de *bootblock* au format *AmigaDos*, nous allons donc la charger et regarder de plus près tout ça.
Insérer notre **disquette de backup** préalablement crée avec *Xcopy* dans le lecteur

#RT alias Read Track, permet le chargement de la track 0 à 1 (1ère piste de la face 0)
#D, alias Désassemble
Taper : `RT 0 1 10000` puis `D 10000+c`

```
rt 0 1 10000
Disk ok

d 10000+c
~01000C MOVE.L  #7FFF7FFF,D0
~010012 MOVE.L  D0,00DFF09A
~010018 MOVE.W  D0,00DFF096
~01001E LEA     10038(PC),A0
~010022 LEA     00060000,A1
~010028 MOVE.L  A1,00000010.S
~01002C MOVE.W  #18F,D0
~010030 MOVE.L  (A0)+,(A1)+
~010032 DBF     D0,00010030
~010036 ILLEGAL
```

+c car le code du bootblock commence à cette adresse, avant c'est la signature disque.

Regardons ça en détail :

```
01000C MOVE.L   #$7FFF7FFF,D0          ; Arrêt de l'OS
010012 MOVE.L   D0,$DFF09A             ; On désactive toutes les interruptions
010018 MOVE.W   D0,$DFF096             ; On désactive toutes les DMA
01001E LEA      10038(PC),A0           ; Adresse Source en A0, à savoir $10038
010022 LEA      00060000,A1            ; Adresse de Destination en A1, à savoir $60000
010028 MOVE.L   A1,$00000010.S         ; On met l'adresse de A1 dans $10 (pour le futur ILLEGAL)
01002C MOVE.L   #18F,D0                ; D0=18F
010030 MOVE.L   (A0)+,(A1)+            ; ➔
010032 DBF      D0,_00010030           ; ← Boucle de recopie du code 10038 vers 60000
010036 ILLEGAL                         ; Et on exécute le code pointé en adr. $10
                                       ; A savoir $60000
```

Comme il est préférable de travailler avec les adresses réelles utilisées dans le jeu et pour un meilleur facilité d'analyse, nous allons faire quelques modifications.

Taper :
#TRANS, alias transfert. Permet le transfert d'une zone mémoire vers une autre.
#A, alias Assemble, Instruction qui va permettre de taper du code assembleur.
#BOOTCHK Alias Boot Check. Permet de calculer un nouveau checksum pour un bootblock
#WT, alias Write Track. Permet d'écrire une zone mémoire sur la disquette à l'adresse indiqué en cylindre.

On déplace le code principale vers *60000*
`TRANS 10038 10038+400 60000`

On va ajouter une temporisation à savoir : Un écran rouge en attente d'appuie sur le bouton de la souris, puis un écran vert pour dire que le code continue.

```
A 10038
$010038  move.w #$f00,dff180          ; Fond d'écran en rouge
$010040  btst #6,$bfe001              ; Le bouton de la souris est appuyé ?
$010048  bne 10038                    ; Non, on retourne en 10038
$01004A  move.w #$0f0,dff180          ; Oui, on continue avec un fond d'écran en vert
$010052
```

Taille de notre petit code : $10052-$10038=**$1A**

On replace le code préalablement déplacé à la suite de notre petit bout de code.
`TRANS 60000 60000+400 10052`

On recalcul le checksum de notre code final
`BOOTCHK 10000`

Et on écrit le nouveau *bootblock*
`WT 0 1 10000`

Rebooter ensuite votre Amiga normalement.

Part 5 Analyse du TrackLoader du bootblock

Le *bootblock* se charge et notre fond d'écran rouge apparait.
Entrer dans l'**AR, taper :** **D 60000+1A**

```
d 60000+1A
~06001A LEA      00000400.S,A7
~06001E MOVE.L   60222(PC),-(A7)
~060022 MOVE.W   #2000,SR
~060026 LEA      60222(PC),A0
~06002A LEA      60206(PC),A5
~06002E LEA      00BFD100,A4
~060034 LEA      00DFF024,A6
~06003A MOVE.L   #55555555,D6
~060040 MOVE.L   (A0)+,D5
~060042 MOVE.L   (A0)+,16(A5)
~060046 MOVE.L   (A0)+,D4
~060048 MOVE.L   #18B8,D3
~06004E MOVE.B   #7F,(A4)
~060052 JSR      (A5)
```

Comme on va le voir un peu plus loin, on a ici un *trackloader*.
Le but n'est pas de comprendre et maitriser toutes les subtilités du code en question mais de comprendre son fonctionnement.
On va suivre le code en mode Trace (commande **TR** et **ST** sous **l'AR),** attention… ça va être long **O_o'**

```
06001A LEA 00000400.S,A7          ; Pile Supervisor en $400
06001E MOVE.L 60222(PC),-(A7)     ;
060022 MOVE.W #2000,SR            ;
060026 LEA 60222(PC),A0           ; A0 = 60222
06002A LEA 60206(PC),A5           ; A5 = 60206
06002E LEA 00BFD100,A4            ; A4 = BFD100, CIA-A PRB (Disk subsystem)
060034 LEA 00DFF024,A6            ; A6 = DFF024, registre DSKLEN
06003A MOVE.L #55555555,D6        ; D6 = 55555555 (utilisé pour décodage MFM)
060040 MOVE.L (A0)+,D5            ; Copie du LongWord pointé part A0 vers D5, ensuite A0=A0+4
                                  :60222 00 04 00 00 00 0F 28 D8 00 00 4A 28 2D 2D 20 54

060042 MOVE.L (A0)+,16(A5)        ; Copie du LongWord pointé part A0 vers adr. Pointé par A5+16, ensuite A0=A0+4
                                  :60222 00 04 00 00 00 0F 28 D8 00 00 4A 28 2D 2D 20 54

060046 MOVE.L (A0)+,D4            ; Copie du LongWord pointé part A0 vers D4, ensuite A0=A0+4
                                  :60222 00 04 00 00 00 0F 28 D8 00 00 4A 28 2D 2D 20 54

060048 MOVE.L #18B8,D3            ; D3=$18B8
```

D0				000018B8	00004A28	00040000	55555555	
A0		00060640			00BFD100	00060206	00DFF024	000003FC

```
06004E MOVE.B #7F,(A4)            ; Moteur floppy On
060052 JSR (A5)                   ; Saut à la sous routine en A5 qui est actuellement à 60206
```

CPU_DELAY

```
060206 MOVE.W D7,-(A7)            ; On sauve D7
060208 MOVE.W #2710,D7            ; On met la valeur 2710 dans D7
06020C DBF D7,0006020C            ; On décrémente D7, jusqu'à qu'il soit égale à -1
060210 MOVE.W (A7)+,D7            ; On restaure D7
060212 RTS                        ; Une fois fini, on revient au code.
```

```
060054 BCLR #3,(A4)               ; Bit 3 de BFD100 = /SEL0 (DF0)
060058 JSR (A5)                   ; go sub ➔ CPU_DELAY

06005A BSR 00060172               ; On branche en 60172
```

DISK READY ?

```
060172 BTST #5,F01(A4)            ; On test le bit 5 (/RDY) de BFE001, (BFD100+F01 = BFE001)
060178 BNE 00060172               ; Disque pas prêt ? On revient en 60172
06017A RTS                        ; Disque Prêt, on revient au code
```

```
06005E CLR.W 12(A5)                 ; On 'nettoie' d'une longueur Word l'adresse pointé par A5+12 (60206+12=60218)

060062 MOVE.L 16(A5),D7             ; Copie du LongWord pointé par A5+16 dans D7, (60206+16=6021C)
                                    :6021C 00 0F 28 D8 FF 00 00 04 00 00 00 0F 28 D8 00 00

060066 BSR 0006017C                 ; On branche en 6017C

06017C MOVEM.L D0-D3,-(A7)          ; On sauvegarde les registres D0 à D3 dans la pile
060180 MOVE.L D7,D0                 ; Copie du LongWord D7 dans D0
060182 MOVE.L D3,D2                 ; Copie du LongWord D3 dans D2
```

D0	000F28D8		000018B8	000018B8	00004A28	00040000	55555555	000F28D8
A0		00060640				00060206		000003E8

```
060184 DIVU.W D2,D0                 ; Divise D0/D2 et met le résultat en Word dans D0, (D0 devient 0000009D)
                                    ; F28D8/18B8=9D

060186 MOVE.W D0,D1                 ; Copie du word D0 dans D1
060188 SWAP D0                      ; Swap de D0, (D0=0000009D devient 009D0000)
06018A MOVE.W D0,14(A5)             ; Copie du word D0 (0000) à l'adresse pointé par A5+14 (60206+14= 6021A)
                                     :6021A 00 00 28 D8 FF 00 00 04 00 00 00 0F 28 D8 00 00

06018E EXT.L D0                     ; On étend D0 vers un LongWord en réplicant le bit de signe vers la gauche,
                                    (D0=0000009D devient 00000000)

060190 SUB.L D0,16(A5)              ; On soustrait D0 de 6021C, (60206+16= 6021C), 60206-(D0)=60206
060194 CMPI.W   #50,D1              ; Compare D1 avec #50
060198 BGE 000601A0                 ; Si D1 résultat plus grand ou égal alors on branche en 601A0
```

D0	00000000	8000009D	000018B8	000018B8	00004A28	00040000	55555555	000F28D8
A0	0006022E	00060640				00060206		000003E8

```
0601A0 SF 1B(A5)                    ; Nettoie 1B+A5 (40000+1B=4001B)
0601A4 SUBI.W #50,D1                ; Soustrait 50 à D1, D1 devient 8000004D
0601A8 MOVE.B D1,D7                 ; Copie du bytes D1 ($4D) en D7, donc D7 devient F284D
0601AA MOVE.B 1B(A5),D0             ; copie en byte du contenu de 1B+A5 dans D0
0601AE BEQ 000601B6                 ; Branche en 601B6 si résultat est égal (Z=1)
```

D0	00000000	8000001D	000018B8	000018B8	00004A28	00040000	55555555	000F284D
A0	0006022E	00060640			00BFD100	00060206	00DFF024	000003E8

```
0601B6 BCLR #2,(A4)                 ; Met le Bit2 (SEL) à 0 du CIA-B
0601BA JSR (A5)                     ; go sub → CPU_DELAY
0601BC TST.B 1A(A5)                 ; Comparaison du byte contenu en 60220 avec zéro
0601C0 BGE 000601C4                 ; On branche en 601C4 si résultat est plus grand ou égal
0601C2 BSR 000601DE                 ; Ce n'est pas le cas, on continue avec un branchement en 601DE
0601DE BTST #4,F01(A4)              ; Test le bit 4 (/LED) de BFE001, (BFD100+F01 = BFE001)
0601E6 SF 1A(A5)                    ; Efface le byte à l'adresse (A5)+1A
0601EA RTS                          ; On retourne au code.
```

Un Pas en Avant

```
0601C4 CMP.B 1A(A5),D7              ; On compare en byte D7 avec le contenue de (A5)+1A, D7=Position actuel
0601C8 BEQ 000601D6                 ; On branche en 601D6 si égal alors go sub → Position Atteinte
0601CA BLT 000601D0                 ; On branche en 601D0 si plus petit que
0601CC BSR 000601EC                 ; On branche en 601EC alors goto → Recherche en Avant
0601CE BRA 000601D2                 ; On branche en 601D2
```

Recherche en Avant (Avance d'une piste)

```
0601EC BCLR #1,(A4)                 ; Met le Bit1 (/DIR) à 0 du CIA-B (On avance)
0601F0 ADDQ.B #1,1A(A5)             ; + 1 à (A5)+1A        1A(A5) /!\ va donc contenir la Position actuel
0601F4 BRA 000601FE                 ; On branche en 601FE
```

```
0601FE BCLR #0,(A4)                 ; Met le Bit0 (/STEP) à 0 du CIA-B
060202 BSET #0,(A4)                 ; Met le Bit0 (/STEP) à 1 du CIA-B

060206 MOVE.W D7,-(A7)              ; → On est dans la routine CPU_DELAY
060208 MOVE.W #2710,D7              ; …
06020C DBF D7,0006020C              ; … …
060210 MOVE.W (A7)+,D7              ; … … …
060212 RTS                          ; ←

0601CE BRA 000601D2                 ; On branche en 601D2
0601D2 BSR 00060172                 ; go sub → DISK READY
0601D4 BRA 000601C4                 ; goto → Un Pas en Avant
```

→ Reste dans cette boucle de **Un Pas en Avant** et **Recherche en Avant (Avance d'une piste)**
jusqu'à que la condition effectuée en **601C4** donne une résultat 'égal', on aura alors atteint la piste désirée
et on branche alors en **601D6**

Position atteinte

```
0601D6 MOVEM.L (A7)+,D0-D3      ; On restore les valeurs D0-D3 préalablement sauvé dans la pile
0601DA RTS                      ; On revient au code
```

```
06006A MOVE.W #8210,72(A6)      ; 72+(A6)=DFF096=DMACON, activation DMA DISK
060070 MOVE.W #7F00,7A(A6)      ; 7A+(A6)=DFF09E=DMACON, adkcon clear du bit 8 à 14
060076 MOVE.W #9500,7A(A6)      ; 72+(A6)=DFF096=DMACON,
                                ; bit  8=1 → FAST=2us/ bit d'entrée, standard MFM
                                ; bit  9=0 → MBSYNC désactivée
                                ; bit 10=1 → WORDSYNC, Sync Read Word défini avec DSKSYNC 7E
                                ; bit 12=1 → MFMPREC, MFM Precompensation
                                ; bit 13=0 et bit 14=→0 Aucune PRECOMP0

06007C MOVE.W #4124,5A(A6)      ; Définit DSKSYNC (7E) à 4124, Word Standard=$4489
060082 MOVE.W #2,78(A6)         ; 78+(A6)=DFF09C=INTREQ, DSKBLK
060088 BSR 00060172             ; go sub → DISK READY
06008C MOVE.L #70000,-4(A6)     ; (A6)-4=DFF020=DSKPTH
060094 MOVE.W #4000,(A6)        ; (A6)=DFF024=DSKLEN initialisation
060098 MOVE.W #98C4,(A6)        ; (A6)=DFF024=DSKLEN Raw Read len (en word)
06009C MOVE.W #98C4,(A6)        ; Encore une fois pour démarrer le transfert DMA
0600A0 MOVE.L #A0000,D7         ; Copie de #A0000 dans D7

0600A6 BTST #1,-5(A6)           ; On Attends que le Lecteur soir prêt (transfert DMA fini)
0600AC BNE 000600B4             ; On branche en 600B4 si résultat différent (Z=0)
0600B4 BSR 0006011c             ; On branche en 6011C
06011C MOVEM.L D1-D7/A0-A6,-(A7) ; On sauve tous les registres dans la Pile
060120 MOVEA.L E(A5),A0         ; Copie en LongWord du contenu de 602014 (A5+E) dans A0 (A0 passe à 70000)
060124 BSR 00060164             ; On branche en 60164
```

Decodage1

```
060164 MOVE.L (A0)+,D0          ; copie en LongWord pointé par A0 dans D0 ensuite A0=A0+4
060166 MOVE.L (A0)+,D1          ; copie en LongWord pointé par A0 dans D1 ensuite A0=A0+4
060168 AND.L D6,D0              ; …
06016A AND.L D6,D1              ; … …
06016C ADD.L D1,D1              ; … … …
06016E OR.L D1,D0               ; Humm… Ressemble à un décodage tout ça.
060170 RTS                      ; On revient au code
```

```
060126 MOVE.L D0,D1             ; copie en LongWord de D0 dans D1
060128 SWAP D1                  ; On swap D1 (D1=50114D00 devient 4D005041)
06012A CMPI.W #5041,D1          ; Compare D1 avec #5041
06012E BNE 0006015C             ; Si résultat pas identique alors on branche 6015C (Z=0)
060130 CMP.W 1A(A5),D0          ; Compare D0 avec #5041
060134 BNE 0006015C             ; Branche en 6015C si résultat pas égal(Z=0)
060136 SUBQ.L #8,A0             ; On soustrait 8 à la valeur contenue dans A0
060138 MOVE.W #62E,D7           ; copie en Word de 62E dans D7
06013C MOVEQ #0,D3              ; copie en byte de 0 dans D3
```

Decodage2

```
06013E MOVE.L (A0)+,D0          ; → Début boucle sur A0
068140 MOVE.L (A0)+,D2          ;
060142 AND.L D6,D0              ;
060144 AND.L D6,D2              ; Encore une section de décodage.
060146 ADD.L D0,D3              ;
060148 ADD.L D2,D3              ;
06014A DBF D7,0006013E          ; ← On décrémente la valeur de D7
```

→ Reste dans cette boucle de **Decodage2**
jusqu'à que la condition demandée en **6014A** aboutisse, à savoir que **D7** (codé sur 2 bytes) arrive à 0000

```
06014E BSR 00060164             ; go sub → Decodage1
060150 CMP.L D0,D3              ; Compare D3 avec D0
060152 BNE 0006015C             ; Branche en 6015C si résultat pas égal (Z=0)
060154 MOVEM.L (A7)+,D1-D7/A0-A6 ; On restaure les registres sauvé dans la pile
060158 MOVEQ #0,D7              ; copie en Word de 0 dans D7
06015A RTS                      ; On revient au code

0600B6 BEO 000600CE             ; Branche en 600CE si résultat égal (Z=1)
0600CE MOVEA.L E(A5),A0         ; Copie en LongWord du contenu de 60214 (A5+E) dans A0 (A0 passe à 70000)
0600D2 ADDQ.W #8,A0             ; + 8 à A0 (A0 passe à 70008)
0600D4 MOVEA.L D5,A1            ; Copie en LongWord de D5 dans A1 (A1 passe à 40000)
0600D6 MOVE.W 14(A5),D0         ; Copie en Word du contenu de 6021A (A5+14) dans D0 (A0 passe à 12650000)
0600DA MOVE.L D3,D7             ; Copie en LongWord de D3 en D7
0600DC SUB.W D0,D7              ; Soustrait en Word D0 de D7
0600DE LSR.W #2,D7              ; Décale de 2 bit sur le Word vers la gauche de D7
                                ; D7=$18B8   = 0001100010111000
                                ; D7 Devient = 0000011000101110
```

```
0600E0 SUBQ.W #1,D7                    ; Soustrait en Word 1 de D7
0600E2 ADD.W D0,D0                     ; Ajout en Word de D0 dans D0
0600E4 ADDA.W D0,A0                    ; Ajout en Word de D0 dans A0
```

Decodage3

```
0600E6 MOVE.L (A0)+,D0                 ; ➔ Début boucle sur A0 et A1
0600E8 MOVE.L (A0)+,D1                 ; …
0600EA AND.L D6,D0                     ; … …
0600EC AND.L D6,D1                     ; … … …
0600EE ADD.L D1,D1                     ; … … …
0600F0 OR.L D1,D0                      ; Encore une section de décodage.
0600F2 MOVE.L D0,(A1)+                 ; … … … …
0600F4 SUBQ.L #4,D4                    ; … … … … …
0600F6 BEO 00060106                    ; Branche en 600106 si résultat égal (Z=1)
0600F8 DBF D7,000600E6                 ; ➔ On décrémente la valeur de D7
```

➔ Reste dans cette boucle de **Decodage3**
jusqu'à que la condition demandée en **600F8** aboutisse, à savoir que **D7** (codé sur 2 bytes) arrive à 0000

```
0600FC ADD.L D3,16(A5)                 ;
060100 MOVE.L A1,D5                    ;
060102 BRA 00060062                    ; On repart pour un tour complet de chargement de donnée ==➔
```

Quand les données désirées auront été toutes chargées, on sortira de **Decodage3** vers **600106** **Stop Floppy**

Stop Floppy

```
060106 BSET #7,(A4)                    ; Met le Bi7 (/MTR) à 1 du CIA-B BFD100 = /MTR (arrêt MOTEUR DF0)
06010A BSET #3,(A4)                    ; Met le Bit3 (/SEL0) à 1 du CIA-B BFD100 = /SEL0 (arrêt DF0)
06010E JSR (A5)                        ; go sub ➔ CPU_DELAY
060110 BCLR #3,(A4)                    ; Met le Bit3 (/STEP) à 0 du CIA-B BFD100 = /SEL0 (active DF0)
060114 JSR (A5)                        ; go sub ➔ CPU_DELAY
060116 BSET #3,(A4)                    ; Met le Bit3 (/SEL0) à 1 du CIA-B BFD100 = /SEL0 (arrêt DF0)
06011A RTS                             ; Retour au code
```

Puis une exécution du code à l'adresse chargé suivra.

```
040000 XXXXXXX                         ; Exécution du code chargée en 4000
```

Bon, pas besoin d'aller plus loin dans notre analyse, on a toutes les informations pour ripper les données.
Le code du ***trackloader*** contient surement d'autre routine que nous n'avons pas vue en œuvre comme le **RETRY** par exemple mais…
on en a pas besoin d'une analyse complète pour sauver nos données :=)

Part 6 Conclusion analyse du TrackLoader du bootblock

```
06003A MOVE.L #55555555,D6          ; D6 = 55555555 (utilisé pour décodage MFM)
```

➔ **Pistes enregistrées au format MFM**

```
060040 MOVE.L (A0)+,D5              ; Copie du LongWord pointé part A0 vers D5, ensuite A0=A0+4
                                    :60222 00 04 00 00 00 0F 28 D8 00 00 4A 28 2D 2D 20 54
```

➔ Une **'table'** est disponible en mémoire contenant les informations des données à charger.
➔ Le **'pointeur'** de cette table semble être **A0**
➔ Le **'1er champ'** que l'on retrouve copier en **D5** semble être la **destination mémoire**

```
060042 MOVE.L (A0)+,16(A5)          ; Copie du LongWord pointé part A0 vers adr. Pointé par A5+16, ensuite A0=A0+4
                                    :60222 00 04 00 00 00 0F 28 D8 00 00 4A 28 2D 2D 20 54
```

➔ Chaque **'champ'** semble avoir une longueur d'un **LongWord**

```
060046 MOVE.L (A0)+,D4              ; Copie du LongWord pointé part A0 vers D4, ensuite A0=A0+4
                                    :60222 00 04 00 00 00 0F 28 D8 00 00 4A 28 2D 2D 20 54
```

➔ Le **'3eme champ'** que l'on retrouve copier en **D4** semble être la **longueur de donnée à copier**

```
060048 MOVE.L #18B8,D3              ; D3=$18B8
060182 MOVE.L D3,D2                 ; Copie du LongWord D0 dans D2
060184 DIVU.W D2,D0                 ; F28D8/18B8=9D
… … …
```

➔ La longueur custom d'une track semble être de $18B8

```
06007C MOVE.W #4124,5A(A6)          ; Définit DSKSYNC (7E) à 4124, Word Standard=$4489
```

➔ DSK SYNC à une taille custom de $4124 au lieu de $4489

```
06005E CLR.W 12(A5)                 ; On 'nettoie' d'une longueur Word l'adresse pointé par A5+12 (60206+12=60218)
060062 MOVE.L 16(A5),D7             ; Copie …
060102 BRA 00060062                 ; On repart pour un tour complet de chargement de donnée ==
```

➔ L'adresse 60062 semble être un bon **'BreakPoint'** pour récupérer les infos de destination et longueur avant lecture

```
060106 BSET #7,(A4)                 ; Met le Bi7 (/MTR) à 1 du CIA-B BFD100 = /MTR (arrêt MOTEUR DF0)
06010A BSET #3,(A4)                 ; Met le Bit3 (/SEL0) à 1 du CIA-B BFD100 = /SEL0 (arrêt DF0)
…
```

➔ L'adresse 60106, routine **'Stop Floppy'** semble être un bon **'BreakPoint'** pour sauver les données traitées en mémoire

```
0601D4 BRA 000601C4                 ; goto ➔ sous routine Un pas en Avant
```

#Reste dans cette boucle de Un Pas en Avant et Recherche en Avant (Avance d'une piste)
jusqu'à que la condition effectuée en 601C4 donne une résultat 'égal',
on aura alors atteint la piste désirée et on branche alors en 601D6

```
0601D6 MOVEM.L (A7)+,D0-D3          ; début Sous Routine Position Atteinte
```

➔ L'adresse 601D6, routine 'Position Atteinte' semble aussi être un bon **'BreakPoint'** pour connaitre la piste désirée

On va vérifier tout ça.

Notre disquette de backup préalablement crée avec **Xcopy** devrait être toujours dans le lecteur.
Rebooter votre Amiga normalement.

Le ***bootblock*** se charge et notre fond d'écran rouge apparait, **entrer** dans l'**AR**.
Retirer notre disquette de **backup** du lecteur et **insérer** la disquette **originale** dans le lecteur **DF0**

#BS, alias BreakPoint. Permet, dès que l'adresse mémoire indiquée est atteinte, d'effectuer un arrêt du code.
Taper : `BS 60062` puis `BS 60106` puis `BS 601D6`

60062 ➔ va nous permettre de connaitre l'adresse mémoire de destination
601D6 ➔ va nous permettre de connaitre la piste désirée à lire
60106 ➔ va nous permettre de sauver les DATA lu du disque

Taper : `X` , on revient à notre écran rouge, **appuyer sur le bouton de la souris**
#X permet le retour au code Amiga en court.
Assez rapidement, notre *BreakPoint* 60062 est atteint et on entre automatiquement dans notre **AR**

#R permet d'afficher tous les registres du 68000 et/ou de changer des valeurs des registres.
Taper : `R`

On va donc lire des **DATA** d'une longueur de **$4A28** présent sur le disque et les copier vers la zone mémoire **$40000**

```
Breakpoint raised at address: 00060062

r
D0=7FFFFFFF 8000FFFF 00000001 000018B8  00004A28 00040000 55555555 0000004F
A0=0006022E 00060640 FFFFFFFF 00FE86EE  00BFD100 00060206 00DFF024 000003FC
PC = 00060062 USP = 00C014B2 SR = 2004 T=0 S=1 I=000 X=0 N=0 Z=1 V=0 C=0
```

#M alias memory read. Permet de voir les données en mémoire au format HEXA/ASCII.
Jetons un petit coup d'œil en **$40000, taper : `M 40000`** puis `M 40000+4A28-10`

On va remplir la zone mémoire de destination avec '**AA**', on pourra après chargement savoir quelle quantité de donnée a été chargé.
#O, alias Fill mémory. Permet de remplir une zone mémoire avec une valeur hexa.
Taper : `O "AA", 40000 40000+5000`

```
m 40000
:040000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .................
:040010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .................

m 40000+4A28-10
:044A18 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .................
:044A28 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .................

o "AA", 40000 40000+5000
Ready.

m 40000
:040000 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
:040010 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
```

Taper : `X` pour retourner au code.
Le lecteur s'empresse d'aller jusqu'à la piste désirée et notre *BreakPoint* 601D6 est atteint, on entre automatiquement dans notre **AR**

Taper : `R` et Regarder la valeur en bytes de **D7** (encadré en VERT dans l'image ci-dessous), **$4D** = **77** en décimal

```
x
No known virus in memory!
Ready.
Breakpoint raised at address: 000601D6
r
D0=00000000 8000004D 000018B8 000018B8  00004A28 00040000 55555555 000F284D
A0=0006022E 00060640 FFFFFFFF 00FE86EE  00BFD100 00060206 00DFF024 000003E8
PC = 000601D6 USP = 000018B2 SR = 2004 T=0 S=1 I=000 X=0 N=0 Z=1 V=0 C=0

CPU: 9%  FPS: 50.0  Power  HD  CD  77  40  00  00
```

Si vous êtes sous WinUAE, vous pouvez voir que cela concorde avec l'affichage de la position actuel de la tête du lecteur.
Encadré en rouge dans l'image ci-dessus.

Taper : `X` pour retourner de nouveau au code.
Très rapidement, notre *BreakPoint* 60062 est encore atteint et on entre automatiquement dans notre **AR.**

On va vérifier la zone mémoire écrite.
Taper : `F "AA", 40000` et appuyer rapidement sur la touche **ESC** des apparitions des résultats.

```
f "AA", 40000
Search from: 040000 to: 080000
0418B8  0418B9  0418BA  0418BB  0418BC  0418BD  0418BE  0418BF  0418C0  0418C1
0418C2  0418C3
```

Etrange, on a bien chargée des données mais pas jusqu'à **$44A28** mais jusqu'à **$418B7** (418B8-1).
Regardons quelles infos ont a dans les registres pour le prochain chargement.

Taper : `R`
Effectivement, **D4** et **D5** ont changé de valeur.
On va maintenant lire une longueur **$3170** bytes du disque et la copier en **$418B8**
Il semblerait donc que l'on ait ici plusieurs 'appels' pour le chargement complet de la zone initial.

Taper : `M 418B8-10`
On peut voir qu'effectivement on a bien chargée en mémoire des données jusqu'à **$418A7** (418B8-1)
Et que le prochain chargement va continuer à remplir cette zone mémoire.
Il semblerait donc que le *trackloader* utilise plusieurs chargements pour cette section, étrange mais bon, pourquoi pas.

```
Breakpoint raised at address: 00060062

r
D0=553A7E14 002A2A00 00000001 000018B8  00003170 000418B8 55555555 0000FFFF
A0=00073178 000418B8 FFFFFFFF 00FE86EE  00BFD100 00060206 00DFF024 000003FC
PC = 00060062 USP = 000018B2 SR = 2000 T=0 S=1 I=000 X=0 N=0 Z=0 V=0 C=0

m 418B8-10
:0418A8 15 04 A2 18 EF F5 0E DE CE F5 5E 88 55 3A 7E 14 ..........^.U:~.
:0418B8 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
:0418C8 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
:0418D8 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
```

Taper : `X` pour retourner de nouveau au code.
Notre *BreakPoint* 601D6 est atteint et on entre automatiquement dans notre **AR**

Taper : `X` de nouveau, notre ***BreakPoint* 60062** est une fois de plus atteint et on entre automatiquement dans notre **AR**
Taper : `R`
Encore une fois, **D4** et **D5** ont changé de valeur.
On va maintenant lire une des données d'une longueur de **$18B8** bytes du disque et la copier en **$43170**

```
Breakpoint raised at address: 00060062

r
D0=84E17EA7 80A02AA2 00000001 000018B8  000018B8 00043170 55555555 0000FFFF
A0=00073178 00043170 FFFFFFFF 00FE86EE  00BFD100 00060206 00DFF024 000003FC
PC = 00060062 USP = 000018B2 SR = 2000 T=0 S=1 I=000 X=0 N=0 Z=0 V=0 C=0

m 418B8-10
:0418A8 15 04 A2 18 EF F5 0E DE CE F5 5E 88 55 3A 7E 14 ..........^.U:~.
:0418B8 65 93 81 C7 E3 E4 F8 B7 5F 52 91 C0 F2 CF BC F4 e....ä.._R......

m 43170-10
:043160 85 72 68 FD 49 ED 3F 29 6F D2 14 60 84 E1 7E A7 .rh.I.?)o..`..~§
:043170 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
```

Comme affiché dans l'image ci-dessus, vous pouvez vérifier le chargement des données.
Taper : `M 418B8-10` On peut voir qu'il y a eu chargement de donnée, avant ont avait une suite de **00**

Taper : `M 43170-10` On peut voir qu'effectivement, pour l'instant, il n'y a rien en **$43170**

Taper : `X` pour retourner de nouveau au code.
Notre *BreakPoint* 601D6 est atteint et on entre automatiquement dans notre **AR**
Taper : `R` et Regarder la valeur en bytes de **D7** (encadré en ROUGE dans l'image ci-dessous), **$4F** = **79** en décimal

```
Breakpoint raised at address: 000601D6
r
D0=00000000 80A0004F 000018B8 000018B8  000018B8 00043170 55555555 000F5A4F
A0=00073178 00043170 FFFFFFFF 00FE86EE  00BFD100 00060206 00DFF024 000003E8
PC = 000601D6 USP = 000018B2 SR = 2004 T=0 S=1 I=000 X=0 N=0 Z=1 V=0 C=0
```

Taper : `X` de nouveau
Notre dernier *BreakPoint* 60106 est finalement atteint et on entre automatiquement dans notre **AR**
On vérifie que des données ont bien été lue et ajouter, **Taper** : `M 43170-10`

```
M 43170-10
:043160 85 72 68 FD 49 ED 3F 29 6F D2 14 60 84 E1 7E A7 .rh.I.?)o..`..~§
:043170 88 02 2D EC A5 10 3E 21 A1 B1 14 C1 6B 83 7A 88 ..-...>!....k.z.
```

On peut aussi vérifier avec la recherche de notre pattern 'AA'

```
f "AA", 40000
Search from: 040000 to: 080000
044A28  044A29  044A2A  044A2B

M 44A28-10
:044A18 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
:044A28 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
:044A38 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
:044A48 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
```

Si on regarde bien, on a effectué 3 chargements :

D5 Adresse de Destination	**D4 Longueur de data à lire**	**Data réel écrite en mémoire**
40000	**4A28**	**40000 → 418B7**
418B8	**3170**	**418B8 → 43169**
43170	**18B8**	**43170 → 44A28**

Part 7 Sauvegarde des pistes custom part #1 – Pre Game

Bon, on va profiter de ce test pour sauver ce 1er chargement**.**

Insérer une nouvelle disquette dans le lecteur
#FORMAT, Permet de formater une disquette au format AmigaDOS.
#SM, alias SaveMemory. Permet donc de sauver une zone mémoire vers un fichier.

Taper `FORMAT TOKI_FILE`

```
FORMAT TOKI_FILE
Ready to format disk in drive DF0: (y/n)?
y
Formatting track !75, head 0
```

Et on sauve tout ça sur disque, **taper** `SM 77-79, 40000 40000+4A28`

Rebooter votre amiga sur votre disquette contenant vos outils de décompression.
Ici l'outils **Black Decrunch V1.1** de la compilation : **CruncherDisk v7.0 (1993-04-09)(Effect)**
Charger notre fichier **77-79**, comme l'indique l'image ci-dessous, il est compressé.
Sauver le fichier décompressé sur notre disk `TOKI_FILE` en tant que **77-79_decomp**

```
AmigaDOS
Copyright ©1987 Commodore-Amiga, Inc.
All rights reserved.
Release 1.3

 THE BLACK DECRUNCHER V1.11
 all coded in pure assembler by BLACK SUN in 1992!
 WARNING: This utility is not for lamers... sorry...

 Now Requirements: - 512k Fast or more
                   - many softwares to swap with me...

 My Address: BLACK SUN
             P.O. Box 8
             CH-1096 CULLY
             SWITZERLAND

 Source File is now in memory at: $00C171B8
 Analysing...  Cruncher recognized: Black&Decker 2.0
 Decrunch to: $00C1BBE0
 File decrunched, length: $00004048
```

Ou passer par un autre outil, par exemple **xfdDecrunch** (ici, sous Kickstart 1.3 et workbench 1.3.3)

```
Workbench release.      692168 free memory
                                          NEWCON
                                          TOKI_FILE
New Cli Window
1> cd DF1:
1> xfddecrunch 77-79 77-79_decomp
xfdDecrunch 1.9 (09.09.1999) © 1994-99 Georg Hörmann, Dirk Stöcker

Reading "77-79" (18984 bytes)...
Decrunching "ByteKiller 1.3" file...
Writing "TOKI_FILE:77-79_decomp" (16456 bytes)...
Saving relocated image (Addr: $00060000, Jmp: $00060000)...

1>
```

Noter au passage, que la signature trouvée n'est pas la même…étrange
J'aurais tendance à faire plus confiance à **xfdDecrunch** car il est plus récent mais bon, les deux fichiers décompactés sont identiques.

Disk `TOKI_FILE` dans le lecteur, **Rebooter votre amiga** et **entrer dans l'AR**
Nous allons charger le fichier préalablement décompressé à l'adresse d'origine.

#LM alias loadMemory. Permet de charger un fichier vers une zone mémoire spécifique.
Taper : `LM 77-79_decomp, 40000` puis `D40000`

```
lm 77-79_decomp, 40000
Loading from 040000 to 044048
Disk ok
d 40000
~040000 BRA     00043EB6
;=====================================
~040004 ASL.?   D7,D5
~040006 MOVEA.W (A6)+,A1
~040008 ROXR.W  #7,D3
~04000A ADDQ.L  #7,-(A1)
~04000C LINEF
```

On voir clairement que le 'code' à partir de **$40004** est complétement incohérent, nous avons de grande chance qu'il soit crypté.
D'ailleurs, on **$40000** on branche directement vers **$43EB6** qui semble faire du décodage.

```
d 43EB6
~043EB6 MOVE.L  #7FFF7FFF,D0
~043EBC MOVE.L  D0,00DFF09A
~043EC2 MOVE.L  D0,00DFF096
~043EC8 LEA     43ED2(PC),A0
~043ECC MOVE.L  A0,00000010.S
~043ED0 TAS.B   SR
~043ED2 LEA     00000024.S,A7
~043ED6 LEA     43EE0(PC),A0
~043EDA MOVE.L  A0,00000080.S
~043EDE TRAP    #0
~043EE0 MOVE.W  #2700,SR
~043EE4 MOVEQ   #7F,D6
~043EE6 ADDQ.W  #8,D6
~043EE8 ADDQ.W  #1,D6
~043EEA LEA     43F40(PC),A0
~043EEE MOVEA.L D6,A1
~043EF0 MOVE.L  (A0)+,(A1)+
~043EF2 MOVE.L  (A0)+,(A1)+
~043EF4 MOVE.L  (A0)+,(A1)+
~043EF6 MOVE.L  (A0)+,(A1)+
~043EF8 MOVE.L  (A0)+,(A1)+
~043EFA MOVE.L  (A0)+,(A1)+
~043EFC MOVE.L  (A0)+,(A1)+
~043EFE MOVE.L  (A0)+,(A1)+
~043F00 MOVE.L  (A0)+,(A1)+
~043F02 MOVE.L  (A0)+,(A1)+
~043F04 MOVE.L  (A0)+,(A1)+
~043F06 MOVE.L  (A0)+,(A1)+
~043F08 MOVE.L  (A0)+,(A1)+
~043F0A MOVE.L  (A0)+,(A1)+
~043F0C MOVE.L  (A0)+,(A1)+
~043F0E MOVE.L  (A0)+,(A1)+
~043F10 LEA     40010(PC),A0
~043F14 LEA     00000488.S,A1
~043F18 MOVEA.L A1,A2
~043F1A MOVE.W  #1387,D7
~043F1E MOVE.L  (A0)+,(A1)+
~043F20 DBF     D7,00043F1E
~043F24 MOVE.L  D6,00000024.S
~043F28 MOVE.L  #EFDA43AB,D0
~043F2E MOVEA.L #12345678,A0
~043F34 MOVE.L  A0,USP
~043F36 MOVE.W  #7FFF,D3
~043F3A MOVEQ   #FFFFFFFF,D2
~043F3C MOVE.W  D2,SR
~043F3E JMP     (A2)
;=====================================
```

Peu importe, on va laisser le code original décrypter ses propres données.

Réinsérer la disquette **originale** dans le lecteur **DF0** puis **Taper :** `G 40000`
On entend la tête du lecteur se déplace assez rapidement, **entrer dans l'AR** avant que l'on atteigne la piste désirée.

Taper : `R`
Si le *trackloader* en action fonctionne pareil que celui que l'on a pu voir dans notre 1ère section de ce tuto,
alors **D4** et **D5** devrait nous intéresser.

```
r
D0=00000000 A828004B 000018B8 000018B8  00002710 00060000 55555555 000E2121
A0=0000055A 0000008E 00000178 1B601C44  00BFD100 00000532 00DFF024 00000106
PC = 00000538 USP = 42E2338B SR = 2000 T=0 S=1 I=000 X=0 N=0 Z=0 V=0 C=0
```

Ce qui nous donnes :
D5 = **60000** adr de destination.
D4 = **2710** longueur data à copier.

Même 'technique' qu'utiliser précédemment, on remplit la zone en question
Taper : `O "AA", 60000 60000+3000`

Si ce *trackloader* fonctionne comme celui que l'on a déjà vue, on va retrouver nos données en **D5 ($60000)**
Mais vue que le code vu après **$40000** semble crypté, il est toujours possible que l'on ait ici une autre version de *trackloader* ou une variante de celui-ci.

Quoi qu'il en soit, fort probable que la sous routine '`Stop Floppy`' soit toujours la même.
A savoir : `BSET #7,(A4)` ➔ **08 D4 00 07**
On va donc chercher en mémoire celle-ci, **Taper :** `F 08 D4 00 07`

De même, toujours dans l'hypothèse que le *trackloader* fonctionne comme celui que l'on a déjà vue, on devrait aussi retrouver la sous routine : '`Position Atteinte`'

A savoir : `MOVEM.L (A7)+,D0-D3` ➔ **4C DF 00 0F**
Cherchons aussi celle-ci en mémoire, **Taper :** `F 4C DF 00 0F`

```
F 08 D4 00 07
Search from: 000000 to: 080000
000228
Ready.

F 4C DF 00 0F
Search from: 000000 to: 080000
000502
Ready.
```

On pose nos *BreakPoints*, **Taper :** `BS 228` puis `BS 502`

```
bs 228
Breakpoint inserted
Ready.
bs 502
Breakpoint inserted
Ready.
```

On retourne au code, **Taper :** `X`
Le lecteur de disque continue son petit job et atteint la piste désirée.
Notre *BreakPoint 502* `'Position Atteinte'`, est alors atteint et on entre automatiquement dans l'**AR**
Taper : `R`

On regarde en Byte **D7** qui nous donnes la valeur **$4B** qui est décimal nous donnes **75**
Nous sommes donc à la **piste 75** (ceci peut être confirmé visuellement si vous utiliser WinUAE)

```
r
D0=00000000 A828004B 000018B8 000018B8  00002710 00060000 55555555 000EF74B
A0=0000055A 0000008E 00000178 1B601C44  00BFD100 00000532 00DFF024 0000010C
PC = 00000502 USP = 42E2338B SR = 2004 T=0 S=1 I=000 X=0 N=0 Z=1 V=0 C=0
```

CPU: 9% | FPS: 50.1 | Power | HD | CD | 75 | 00 | 00 | 00

On retourne à notre code, **Taper :** `X`

Après un bref chargement, notre *BreakPoint 228* `'Stop Floppy'`, est atteint et on entre automatiquement dans l'**AR**
Allons vérifier si on a maintenant quelque chose en **$60000**
Taper : `M 60000`

```
Breakpoint raised at address: 00000228

m 60000
:060000 49 63 65 21 00 00 1F 48 00 00 84 18 A8 9A B6 0D Ice!...H........
:060010 D9 74 96 E7 5C B3 E7 5E 4E AC FC 00 26 4A FC 46 .t..\..^N.ü.&JüF
```

Non seulement des données sont présentes mais on peut voir clairement qu'elles sont au format **ICE**

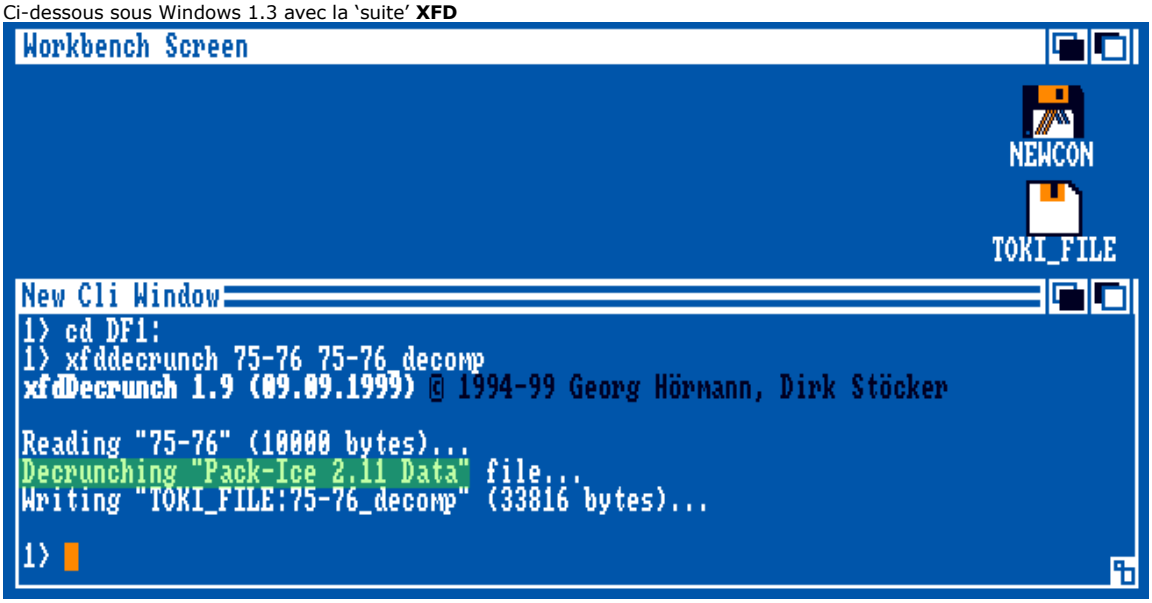
On va chercher la fin de ces données.

On cherche la fin des données, **Taper :** `F "AA", 60000` et appuyer rapidement sur la touche **ESC** des apparitions des résultats.

```
f "AA", 60000
Search from: 060000 to: 080000
062710  062711  062712  062713  062714  062715  062716  062717  062718  062719
06271A  06271B  06271C  06271D
```

62710, exactement la valeur que l'on attendait !
On va s'empresser de sauver tout ça.
Réinsérer notre Disk `TOKI_FILE` dans le lecteur et **Taper :** `SM 75-76, 60000 60000+2710`

Ne pas oublier de décompresser ce fichier avec votre compil de cruncher/Decruncher
Ci-dessous sous Windows 1.3 avec la 'suite' **XFD**



On va reprendre le même scénario pour le prochain 'chargement'.
Disk `TOKI_FILE` dans le lecteur et fichier **75-76_decomp** présent sur celui-ci, **rebooter votre Amiga**, **Entrer dans l'AR**

Taper : `L 75-76_decomp, 60000`

```
No known virus in memory!
Ready.
lm 1:75-76_decomp, 60000
Loading from 060000 to 068418
Disk ok
```

Alors, crypté ou pas crypté cette fois ci ☺
Si le *trackloader* en action fonctionne encore pareil, on va rechercher nos deux sous routine : `Position Atteinte` et `Stop Floppy`

Taper : `F 4C DF 00 0F` puis `F 08 D4 00 07`

```
F 08 D4 00 07
Search from: 000000 to: C80000
0605B0
Ready.

F 4C DF 00 0F
Search from: 000000 to: C80000
060680
Ready.
```

On retrouve nos deux sous routines en mémoire, ce qui est une bonne nouvelle.
On pose donc nos deux *BreakPoints*, **Taper :** `BS 605B0` puis `BS 60680`

Réinsérer ensuite la disquette **originale** dans le lecteur **DF0** puis **Taper :** `G 60000`
On entend la tête du lecteur se déplace assez rapidement et notre *BreakPoint 60680* `'Position Atteinte'`, est atteint, on entre automatiquement dans l'**AR**

Taper : `R` et regarder **D4** et **D5** ainsi que la valeur de **D7** en byte

```
r
D0=00000000 00000037 000018B8 000018B8  0001EE60 00023800 55555555 000D0937
A0=000606D8 000604AC 00FC04FC 00C0153E  00BFD100 000606B0 00DFF024 000003E8
PC = 00060680 USP = 00C01532 SR = 2004 T=0 S=1 I=000 X=0 N=0 Z=1 V=0 C=0
```

CPU: 21% FPS: 50.0 Power HD CD 55 00 00 00

On regarde en 2300 si c'est bien vide, **Taper :** `M 23800`

```
m 23800
:023800 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

Même 'technique' qu'utiliser précédemment, on remplit la zone en question avec un petit supplément.
Taper : `O "AA", 23800 23800+1EE60+2000`

On retourne à notre code, **Taper :** `X`

Le lecteur de disque continue son petit job et effectue un long chargement (profiter en pour compter à l'oreille le nbr d'accès)
Notre *BreakPoint 605B0* `'stop Floppy'` est finalement atteint au bout de 19 accès.

On entre automatiquement dans l'**AR, Taper :** `R`

```
r
D0=15792DBC 002828A8 FFFF4AFC 000018B8  00000000 00040DA8 55555555 00000000
A0=00073178 00042660 00FC04FC 00C0153E  00BFD100 000606B0 00DFF024 000003FC
PC = 000605B0 USP = 00C01532 SR = 2004 T=0 S=1 I=000 X=0 N=0 Z=1 V=0 C=0
```

00000000 en en **D3** = Pas de taille = Fin de copie des données en mémoire.
On peut d'ailleurs vérifier, **Taper :** `F "AAAAAAAA", 23800` et appuyer rapidement sur la touche **ESC** des apparitions des résultats

```
F "AAAAAAAA", 23800
Search from: 023800 to: C80000
042660  042661  042662  042663  042664  042665  042666  042667  042668  042669
04266A  04266B  04266C  04266D
```

Comme adresse de base on avait : **D5**=**$23800** et **D4**=**$1EE60**, **23800+$1EE60=$42660**
Et l'on retrouve bien une fin de donnée en **$42660**.

Nous avons compté 19 accès et l'on a commencé à la piste 55 ce qui nous donnes : 55+19=**74** comme piste de fin

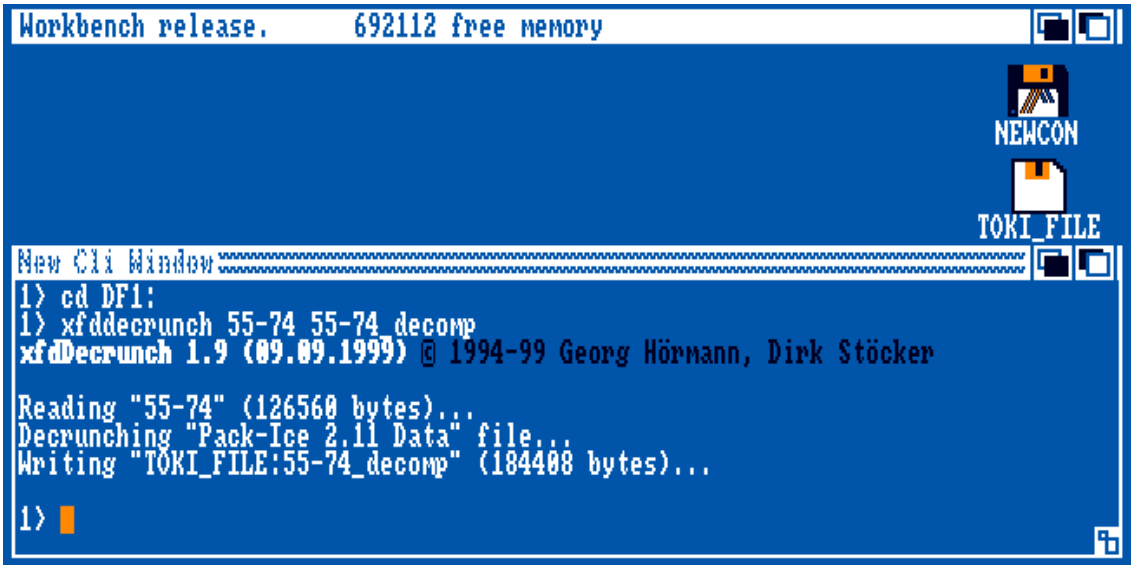
On s'empresse de sauver tout ça mais avant on jette un coup d'œil en mémoire.
Taper : `M 23800` puis `23800+1EE60-20`

```
m 23800
:023800 49 63 65 21 00 01 71 08 00 02 D0 58 08 F9 BF E0 Ice!..q....X....
:023810 01 23 FC E7 96 26 38 A0 80 4E 40 01 20 F3 30 1C .#ü..&8..N@. .0.

m 23800+1EE60-20
:042640 A4 1A C6 63 2A EF 1B 9F 0A C6 FD A8 6B 1B F5 07 ...c*.......k...
:042650 58 D8 58 DF 81 CA 29 9A 47 C3 D5 B8 15 79 2D BC X.XÞ..).G....y-.
:042660 AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA ................
```

Encore un format compressé ☺

Réinsérer notre Disk `TOKI_FILE` dans le lecteur et **Taper :** `SM 55-74, 23800 23800+1EE60`

Ne pas oublier de décompresser ce fichier avec votre compil de cruncher/Decruncher.
Ci-dessous sous Windows 1.3 avec la 'suite' **XFD**



Let's go pour le 4eme fichier à sauver

Disk `TOKI_FILE` dans le lecteur et fichier **55-74_decomp** présent sur celui-ci, **rebooter votre Amiga**, **Entrer dans l'AR**

Taper : `L 55-74_decomp, 23800`

```
lm 55-74_decomp, 23800
Loading from 023800 to 050858
Disk ok
```

On recherche les ***Upcodes*** de nos deux sous routines.

Taper : `F 4C DF 00 0F` puis `F 08 D4 00 07`

```
f 4C DF 00 0F
Search from: 000000 to: C80000
02B888
Ready.

f 08 D4 00 07
Search from: 000000 to: C80000
02B7FC
Ready.
```

On va poser nos deux *BreakPoints*
Taper : `BS 2B888` puis `BS 08 2B7FC`

Réinsérer ensuite la disquette **originale** dans le lecteur **DF0** puis **Taper :** `G 23800`
On entend la tête du lecteur se déplace assez rapidement et notre *BreakPoint 2B888* `'Position Atteinte'`, est atteint, on entre automatiquement dans l'**AR**

Taper : `R`

```
Breakpoint raised at address: 0002B888
r
D0=00000300 FFD00010 000018B8 000018B8  0000DB8C 00060000 55555555 00094810
A0=00029F5E 00200000 00000000 00080000  00BFD100 0002E116 00DFF024 000002E4
PC = 0002B888 USP = 00000000 SR = 2704 T=0 S=1 I=111 X=0 N=0 Z=1 V=0 C=0
```

Noter au passage la valeur en byte de **D7** à savoir **$10** (16 en décimal)

On utilise toujours la même 'technique' à savoir remplir la zone de destination.
Avant, on va quand même jeter un coup d'œil à la zone en question.
Taper : `M 60000` puis `M 60000+DB8C-20`

```
M 60000
:060000 F2 D4 B6 98 00 00 00 00 00 00 00 00 00 00 00 00 ................
:060010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
:060020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
:060030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
:060040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................

M 60000+DB8C-20
:06DB6C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
:06DB7C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
:06DB8C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
:06DB9C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

On a quand même un 'début' de chargement de donnée jusqu'en **$60003**, on va donc modifier le début de l'adresse à remplir dans ce sens.
De toute façon, c'est surtout la futur adresse de fin qui nous intéresse.

Taper : `O "AA", 60000+4 60000+DB8C+2000`

Et on retourne à notre code, **Taper :** `X`

Le lecteur de disque continue son petit job et effectue un long chargement (profiter en pour compter à l'oreille le nbr d'accès)
Notre *BreakPoint 2D7FC* `'Stop Floppy'` est finalement atteint au bout de 9 accès.

On entre automatiquement dans l'**AR, Taper :** `R`

```
Breakpoint raised at address: 0002B7FC
r
D0=61004241 20000200 00000000 000018B8  00000000 0006DB54 55555555 00000620
A0=00001078 0006DB8C 00000000 00080000  00BFD100 0002E116 00DFF024 000002F8
PC = 0002B7FC USP = 00000000 SR = 2704 T=0 S=1 I=111 X=0 N=0 Z=1 V=0 C=0
```

Tout semble chargé, on va quand même vérifier.
Taper : `F "AAAAAAAA", 60000` et appuyer rapidement sur la touche **ESC** des apparitions des résultats

```
F "AAAAAAAA",60000
Search from: 060000 to: C80000
06DB8B  06DB8C  06DB8D  06DB8E  06DB8F  06DB90  06DB91  06DB92  06DB93  06DB94
06DB95  06DB96  06DB97
```

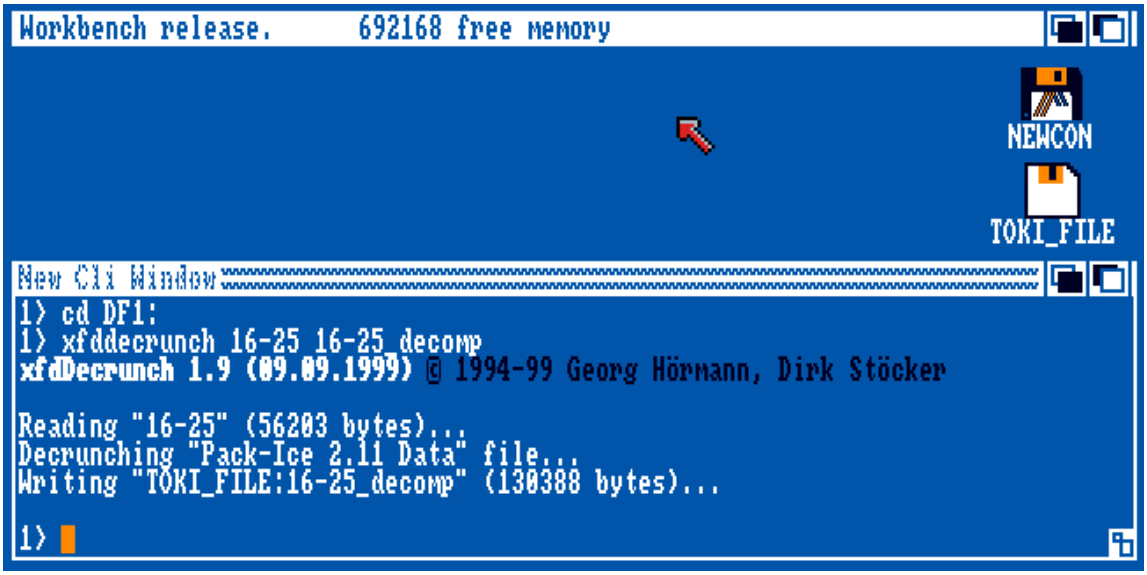
6DB8C, exactement la valeur que l'on attendait !
On en profite pour vérifier ce qui a été chargé, **taper :** `M 60000` puis `M 60000+DB8C-20`

```
M 60000
:060000 49 63 65 21 00 00 DB 8A 00 01 FD 54 8D 5A CC 34 Ice!.......T.Z.4
:060010 9A 57 39 C0 45 7F FF 14 11 D6 12 03 8B E6 08 AB .W9.E▒...Ö......

M 60000+DB8B-20
:06DB6B 07 51 37 01 F4 54 0F 4D C0 7A 81 07 89 10 1E 17 .Q7..T.M.z......
:06DB7B 01 E0 FC 1E 00 40 78 00 00 00 00 EA 03 61 00 42 ..ü..@x......a.B
:06DB8B 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
:06DB9B 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
```

Encore des données compressées au format **ICE**

On sauver tout ça.
Nous avons compté 9 accès en partant de la piste 16, cela nous donne donc une piste de fin en **25** (16+9)

Réinsérer notre Disk **TOKI_FILE** dans le lecteur et **taper :** **SM 16-25, 60000 60000+DB8C**
Puis, décompresser ce fichier avec votre compil de cruncher/Decruncher
Ci-dessous sous Windows 1.3 avec la 'suite' **XFD**



Part 8 Analyse du TrackLoader principal

On va essayer de mieux comprendre le mécanisme de chargement.

Réinsérer la disquette originale dans le lecteur DF0 et **Rebooter** ensuite votre Amiga.
Laisser le jeu se charger normalement jusqu'au Level1 puis, **ENTRER** dans l'**AR**

Comme dis plus haut, dans une routine de *trackloader* on retrouve la plupart du temps l'instruction ***MOVE.L #55555555,Dx***
Il arrive aussi que l'on utilise l'instruction **lea.**

On va regarder si on trouve quelque chose en mémoire dans ce sens, à savoir **l'UPCODE** du *move.l #55555555*
#F, alias FIND. Permet de rechercher une suite ascii ou hexa en mémoire.
Taper : `F 3C 55 55 55 55`

```
f 3C 55 55 55 55
Search from: 000000 to: 080000
00013F  02B741
Ready.
```

Comme on adresse vers **D0-D7**, le move devrait être quelque chose : *move.l #55555555,D0* ou **D1** ou **D2**, etc…
Et *l'upcode* complet de cette instruction serait **XX 3C 55 55 55 55**
Donc à l'adresse-1 de l'adresse trouvée, on jette donc un coup d'œil à ce que l'**AR** nous a donné comme adresse.

Taper : `D 13F-1` et `D 2B741-1`

```
d 00013F-1
~00013E MOVE.L  #55555555,D6

d 02B741-1
~02B740 MOVE.L  #55555555,D6
```

Remonter le code de $13F-1

```
!=========================================
~000122 MOVEA.L -(A3),A0
~000124 BLE     00000195
~000126 BGE     00000148
~000128 MOVE.W  -(A0),-(A7)
~00012A LEA     54E(PC),A0
~00012E LEA     532(PC),A5
~000132 LEA     00BFD100,A4
~000138 LEA     00DFF024,A6
~00013E MOVE.L  #55555555,D6
~000144 MOVE.L  (A0)+,D5
~000146 MOVE.L  (A0)+,16(A5)
~00014A MOVE.L  (A0)+,D4
~00014C MOVE.L  #18B8,D3
~000152 LEA     8A(PC),A1
~000156 LEA     15A(PC),A2
~00015A CLR.L   (A1)+
~00015C CMPA.L  A0,A2
~00015E BCC     0000015A
~000160 MOVE.B  #7F,(A4)
~000164 JSR     (A5)
~000166 BCLR    #3,(A4)
~00016A JSR     (A5)
~00016C BSR     0000049E
~000170 LEA     8A(PC),A1
~000174 LEA     178(PC),A2
```

Bon, on est clairement dans le même style de code trouvée au début de ce tuto.
Noté que l'on retrouve encore la valeur **$18B8** et **A0** est souvent utilisé avec incrémentation, surement indique-t-il une adresse mémoire.

Même manipulation sur $2B740-1

```
;=====================================
~02B72E MOVE.L  A6,-(A7)
~02B730 LEA     2E116(PC),A5
~02B734 LEA     00BFD100,A4
~02B73A LEA     00DFF024,A6
~02B740 MOVE.L  #55555555,D6
~02B746 MOVE.L  (A0)+,D5
~02B748 MOVE.L  (A0)+,-4236(A5)
~02B74C MOVE.L  (A0)+,D4
~02B74E MOVE.L  #18B8,D3
~02B754 MOVE.B  #7F,(A4)
~02B758 PEA     2C8F6(PC)
~02B75C JMP     (A5)
;=====================================
```

Le code n'est pas aussi complet mais on retrouve aussi nos 'marques' du *trackloader.*
Et l'on retrouve toujours notre valeur **$18B8** ainsi que le 'travaille' **avec A0**
On notera aussi le 'début' de nos deux 'routines' : **$122** et **$2B72E**

On en profite pour rechercher nos sous routines que l'on a déjà vue dans la 1er phase d'Analyse du code.
'**Stop Floppy**' et '**Position Atteinte**'

Taper : `F 08 D4 00 07` puis `F 4C DF 00 0F`

```
f 08 D4 00 07
Search from: 000000 to: C80000
000228  02B7FC
Ready.

f 4C DF 00 0F
Search from: 000000 to: C80000
02B888
Ready.
```

On retrouve aussi nos deux sous routines, c'est une bonne chose.
On pose nos *BreakPoints* : **Taper** : `BS 122` puis `BS 2B72E` et `BS 2B7FC` sans oublier `BS 2B888`

Perdez vos vies et laisser le jeu revenir à son menu, il devrait logiquement utiliser un *trackloader*

BINGO.
Nous voilà entré automatiquement dans l'**AR**, l'adresse *$2B72E 'routine trackloader'* a été atteinte, **taper :** `X`
Après un très bref accès disque, c'est le *BreakPoint $2B788 'Position atteinte'* qui est atteint, **taper :** `X`
Un plus long chargement s'effectue et c'est maintenant le *BreakPoint $2B7FC 'Stop Floppy'* qui est atteint, **taper :** `X`

La table des **HighScores** apparait.

NO	NAME	SCORE	AREA
1	MIC	00018000	5
2	JCM	00015000	5
3	PEL	00011000	2
4	PAD	00010000	2
5	ROG	00009000	2
6	ALF	00008500	1
7	IVA	00008000	1
8	ERM	00007500	1
9	STE	00007000	1
10	GUI	00006500	1
11	TAR	00006000	1
12	BOU	00005500	1
13	JEA	00005000	1
14	FAB	00004500	1
15	TEO	00004000	1
16	ARM	00003500	1
17	ARI	00003000	1
18	JAC	00002500	1
19	PAT	00002000	1
20	JUL	00001500	1

Appuyer sur FIRE ou attendez le timeout de l'affichage de la table des **HighScores.**

Nous voilà de nouveau entré automatiquement dans l'**AR**, l'adresse *$2B72E 'routine trackloader'* a été atteinte, **taper : x**
Très rapidement, c'est maintenant le *BreakPoint $2B788 'Position atteinte'* qui est atteint, **taper : x**
Un plus long chargement s'effectue et c'est maintenant le *BreakPoint $2B7FC 'Stop Floppy'* qui est atteint, **taper : x**
L'animation principale du jeu apparait, **appuyer sur FIRE**

Le BreakPoint $2B72E 'routine trackloader' a été atteinte, **taper : x**
Après un très bref accès disque, c'est le *BreakPoint $2B788 'Position atteinte'* qui est atteint, **taper : x**
Encore une fois, c'est le *BreakPoint $2B7FC 'Stop Floppy'* qui est atteint, **taper : x**
Après un bref affichage de ce que l'on a appelé '**mini animation du level'** notre *BreakPoint $2B72E 'routine trackloader'* a été atteinte.
Taper : x
C'est maintenant le tour du *BreakPoint $2B788 'Position atteinte'* à être atteint, **taper : x**
Un plus long chargement s'effectue et c'est maintenant le *BreakPoint $2B7FC 'Stop Floppy'* qui est atteint, **taper : x**
Nous sommes revenus à la '**mini animation du level'** avec la musique. **Appuyer sur FIRE.**
Nous voilà de retour dans le jeu, **Level1**

Noté au passage qu'il semblerait bien que l'on soit tombé sur notre *trackloader principal* puisqu'il est appelé à chaque chargement de piste. Ce qui est une très bonne nouvelle.

On a déjà rippé les quatre 1er chargements de notre tableau.
On sait que **pre-loader** et le **Level1** utilise le *trackloader* en *$2B72E*

Donc on en est là :

Nbr Piste Lu	Affichage Après chargement	Information	Adr. Du Tracker	Piste Lue WinUAE
06	Mini animation du level	Pre-Loader	2B72E	07-13
14	Level1	Level1	2B72E	01-15

On va se servir du '*cheatMode'* vue au début du tuto pour surfer dans les niveaux et ainsi analyser les accès au Levels du jeu, ça va être beaucoup plus rapide.

Rebooter votre amiga, une fois l'anim. Principale chargée (9 pistes), **entrer** dans l'**AR**

Petit rappel de ce que l'on a déjà vue dans la **partie 5 'Analyse du Trackloader du bootblock'**

La sous routine *'Position atteinte'* est appelé par la sous routine *'Un Pas en Avant'* quand la condition en $601C4 est validée :

```
0601C4          CMP.B 1A(A5),D7
```

Donc, **au début** de la sous routine *'position atteinte'*, **D7** devrait contenir une la position désirée. (en Byte pour ce test)

Au début de la routine *'trackloader'* on peut voir qu'il utilise **A0** comme pointeur mémoire et recopie son contenue en **D5** :

```
060040          MOVE.L (A0)+,D5
```

On devrait retrouver donc en **A0** une adresse mémoire qui pointe vers des données qui seront recopiée dans les registres **Dx**

Dans la routine du *'trackloader',* on voir qu'il fonctionne en *longword*. Que le 1er champ est recopié en **D5** et que le 3eme 'champ' de cette zone mémoire est copié en **D4**

```
060040          MOVE.L (A0)+,D5
060042          MOVE.L (A0)+,16(A5)
060046          MOVE.L (A0)+,D4
```

Grace à nos *BreakPoints*, on va pouvoir mieux comprendre tout ça !
Entrer dans l'AR et taper : `BS 2B72E` puis `BS 2B7FC` sans oublier `BS 2B888` puis `X`
Appuyer sur Fire pour passer l'animation.

Le *BreakPoint* **2B72E** 'Routine TrackLoader' est alors atteint, on entre automatiquement dans notre **AR, taper : R**
Noter dans un tableau la valeur de **A0** puis **Taper : X**

Le *BreakPoint* **2B888** 'Position atteinte' est alors atteint, on entre automatiquement dans notre **AR, taper : R**
Dans la même colonne de notre tableau, noter la Valeur de **D4, D5** et **D7** puis **Taper : X**

Après quelque chargement de donnée, notre *BreakPoint* **2B7FC** 'Stop Floppy' est atteint, on entre alors dans l'**AR, taper : X**
Le code chargé est alors exécuté, début de la '**Mini animation du level'**
Vous pouvez donc noter dans votre tableau que les infos que l'on vient de collecter correspondent au chargement de celui-ci.

Notre *BreakPoint* **2B72E** 'Routine TrackLoader' est atteint et on entre automatiquement dans notre **AR, taper : R**
Noter dans une nouvelle colonne vue que c'est un nouveau chargement, la valeur de **A0** puis **Taper : X**

Le lecteur déplace ça tête de lecture jusqu'à la zone désirée et notre *BreakPoint* **2B888** 'Position atteinte' est atteint, **taper : R**
Dans la même colonne de notre tableau, noter la valeur de **D4, D5** et **D7** puis **taper : X**
Après un plus long chargement de donnée, notre *BreakPoint* **2B7FC** 'Stop Floppy' est atteint, on entre alors dans l'**AR, taper : X**
Appuyer sur FIRE, nous somme dans le **LEVEL1** et nous n'avons eu aucun nouveau accès disque, on vient donc de charger le **LEVEL1**
Noter **LEVEL1** dans la colonne de votre tableau.

C'est maintenant beaucoup plus clair dans le fonctionnement du code.

(1) Appel de la routine ***Trackloader 2B72E***
(2) ➔ déplacement de la tété du lecteur de disquette jusqu'à la zone demandée.
(3) Appel de la sous routine ***Position Atteinte 2B888***
(4) ➔ Chargement des données
(5) Appel de la routine ***Stop Floppy 2B7FC***
(6) ➔ Décompression si nécessaire et exécution du code préalablement chargé.

Entrez dans le *cheatMode* et continuer la construction de votre tableau grâce aux informations ci-dessus.
à savoir, effectuer un relevé des valeurs du registre comme préalablement demandé en utilisant les touches **[F1]** a **[F8]**

N'oublier pas l'étape **'standard'** à savoir, perdre normalement toute ses vies et laisser le code retourner à son menu,
cela permet de sauver les données du tableau des scores.

Avec les informations que l'on a eu sur la 1er phase de RIP, vous deviez avoir un tableau qui ressemble à ceci :

					F1	F2	F3	F4
A0 (Addr Tabl)					00029EE6	00029EF2	00029EFE	00029F0A
Chargement de	N/A	Logo Ocean	N/A	Animation Principale	Level1	Level2	Level3	Level4
PISTE lu	77-79	75-76	55-74	16-25	01-15	15-28	28-42	42-54
D7	F284D	EF74B	D0937	94810	1801	1870F	2CC1C	40F2A
Nbr Piste Lu	2	1	19	9	14	13	14	12
Format	ByteKiller 1.3	Ice 2.11	Ice 2.11	Ice 2.11	Ice 2.11	N/A	N/A	N/A
D5 (Addr Start)	40000	60000	23800	60000	50858	50858	50858	50858
D4 (Longueur)	4A28	2710	1EE60	DB8C	16EF8	14500	142F0	134EC
Nom de fichier	77-79	75-76	55-74	16-25	Leve1	Level2	Level3	Level4

	F5	F6	F7				F8
A0 (Addr Tabl)	00029F16	00029F22	00029F2E	00029F3A	00029F46	00029F52	00029F5E
Chargement de	Level5	Level6	Level6Bis	Mini Anim Level	Scores	N/A	Animation fin
PISTE lu	54-68	68-77	77-79 1-7	07-13	13-16	16-25	25-30
D7	54436	69144	7834D	87907	90D0D	94810	A2319
Nbr Piste Lu	14	9	9	6	3	3	5
Format	N/A	N/A	N/A	N/A	Ice 2.11	Ice 2.11	Ice 2.11
D5 (Addr Start)	50858	50858	50858	D300	60000	60000	60000
D4 (Longueur)	14D70	F170	F620	93D8	3AC0	DB8C	9100
Nom de fichier	Level5	Level6	LevelFinal	Pre-Loader	Scores	16-25bis	Anim-End

Vous n'aurez l'information des '**Piste Lu'** que si vous utiliser *WinUAE* bien sûr.

Si on regarde bien les valeurs de **A0** on peut voir qu'elle commence en ***$29EE6*** jusqu'à **$29F5E** et qu'on semble avoir une valeur de saut de **$C**

$29EE6 (Level1) +**$C** = **$29EF2** (Level2)
$29EF2 (Level2) +**$C** = **$29EFE** (Level3)
Etc..

Bon, si vous avez suivi ce tuto jusqu'ici, vous savez déjà que le *trackloader* que l'on a ici fonctionne avec un tableau en mémoire.
Le pointeur se situe en **A0** comme précédemment expliqué.

Le jeu lancer et démarrer sur le **LEVEL1, Entrer dans l'AR et taper :**
`M 29EE6` puis `M 29EF2` puis `M 29EFE` puis `M 29F0A` puis `M 29F16` puis `M 29F22` puis `M 29F2E` puis `M 29F3A` puis `M 29F46` puis `M 29F52` puis `M 29F5E`

```
M 29EE6
:029EE6 00 05 08 58 00 00 18 B8 00 01 6E F8 00 31 6E F8 ...X......n..1n.
M 29EF2
:029EF2 00 31 6E F8 00 01 87 B0 00 01 45 00 00 31 45 00 .1n.......E..1E.
M 29EFE
:029EFE 00 31 45 00 00 02 CC B0 00 01 42 F0 00 31 42 F0 .1E.......B..1B.
M 29F0A
:029F0A 00 31 42 F0 00 04 0F A0 00 01 34 EC 00 31 34 EC .1B.......4..14.
M 29F16
:029F16 00 31 34 EC 00 05 44 8C 00 01 4D 70 00 31 4D 70 .14...D...Mp.1Mp
M 29F22
:029F22 00 31 4D 70 00 06 91 FC 00 00 F1 70 00 30 F1 70 .1Mp...ü...p.0.p
M 29F2E
:029F2E 00 30 F1 70 00 07 83 6C 00 00 F6 20 00 00 D3 00 .0.p...l..ö ....
M 29F3A
:029F3A 00 00 D3 00 00 08 79 8C 00 00 93 D8 00 06 00 00 ......y.........
M 29F46
:029F46 00 06 00 00 00 09 0D 64 00 00 3A C0 00 06 00 00 .......d..:.....
M 29F52
:029F52 00 06 00 00 00 09 48 24 00 00 DB 8C 00 06 00 00 ......H$........
M 29F5E
:029F5E 00 06 00 00 00 0A 23 B0 00 00 91 00 48 E7 FF FE ......#.....H...
```

Nous allons travailler avec ce tableau et le chargement du **LEVEL1**

On a noté qu'il y a eu un chargement de 14 pistes concernant celui-ci, mais en *$29EE6*, on ne voie pas cette valeur que ce soit en hexa (*$E*) ou directement en décimal (*!14*). On va donc logiquement plus s'orienter vers une **adresse de fin de donnée** ou une quantité de donnée mais dans tous les cas, pas en 'piste'.

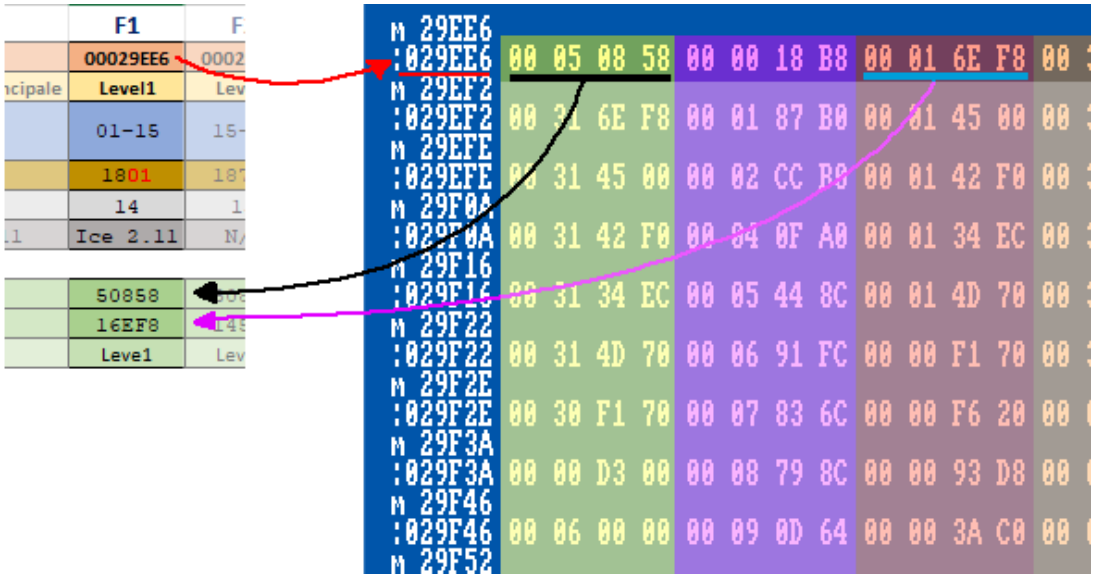
On sait déjà que ce tableau est accédé par 'palier' de **LongWord** (voir explication début de ce chapitre).
Et de plus, visiblement ça se voie , c'est aussi simple que ça ☺

```
M 29EE6
:029EE6 00 05 08 58 00 00 18 B8 00 01 6E F8 00 31 6E F8 ...X......n..1n.
M 29EF2
:029EF2 00 31 6E F8 00 01 87 B0 00 01 45 00 00 31 45 00 .1n.......E..1E.
M 29EFE
:029EFE 00 31 45 00 00 02 CC B0 00 01 42 F0 00 31 42 F0 .1E.......B..1B.
M 29F0A
:029F0A 00 31 42 F0 00 04 0F A0 00 01 34 EC 00 31 34 EC .1B.......4..14.
M 29F16
:029F16 00 31 34 EC 00 05 44 8C 00 01 4D 70 00 31 4D 70 .14...D...Mp.1Mp
M 29F22
:029F22 00 31 4D 70 00 06 91 FC 00 00 F1 70 00 30 F1 70 .1Mp...ü...p.0.p
M 29F2E
:029F2E 00 30 F1 70 00 07 83 6C 00 00 F6 20 00 00 D3 00 .0.p...l..ö ....
M 29F3A
:029F3A 00 00 D3 00 00 08 79 8C 00 00 93 D8 00 06 00 00 ......y.........
M 29F46
:029F46 00 06 00 00 00 09 0D 64 00 00 3A C0 00 06 00 00 .......d..:.....
M 29F52
:029F52 00 06 00 00 00 09 48 24 00 00 DB 8C 00 06 00 00 ......H$........
M 29F5E
:029F5E 00 06 00 00 00 0A 23 B0 00 00 91 00 48 E7 FF FE ......#.....H...
```

On peut, par exemple, s'imaginer un délimitateur virtuel de champ (valeur **$00**), C'est une idée comme une autre non ?
Vue que les tailles dans ce tableau ne dépassent pas 3 bytes.

```
M 29EE6
:029EE6 00 05 08 58 00 00 18 B8 00 01 6E F8 00 31 6E F8 ...X......n..1n.
M 29EF2
:029EF2 00 31 6E F8 00 01 87 B0 00 01 45 00 00 31 45 00 .1n.......E..1E.
M 29EFE
:029EFE 00 31 45 00 00 02 CC B0 00 01 42 F0 00 31 42 F0 .1E.......B..1B.
M 29F0A
:029F0A 00 31 42 F0 00 04 0F A0 00 01 34 EC 00 31 34 EC .1B.......4..14.
M 29F16
:029F16 00 31 34 EC 00 05 44 8C 00 01 4D 70 00 31 4D 70 .14...D...Mp.1Mp
M 29F22
:029F22 00 31 4D 70 00 06 91 FC 00 00 F1 70 00 30 F1 70 .1Mp...ü...p.0.p
M 29F2E
:029F2E 00 30 F1 70 00 07 83 6C 00 00 F6 20 00 00 D3 00 .0.p...l..ö ....
M 29F3A
:029F3A 00 00 D3 00 00 08 79 8C 00 00 93 D8 00 06 00 00 ......y.........
M 29F46
:029F46 00 06 00 00 00 09 0D 64 00 00 3A C0 00 06 00 00 .......d..:.....
M 29F52
:029F52 00 06 00 00 00 09 48 24 00 00 DB 8C 00 06 00 00 ......H$........
M 29F5E
:029F5E 00 06 00 00 00 0A 23 B0 00 00 91 00 48 E7 FF FE ......#.....H...
```

On reste sur l'analyse du **LEVEL1**, donc Adresse de **A0** = **$29EE6**
1er LongWord de cette adresse mémoire = **05 08 58**
2eme LongWord de cette adresse mémoire = **00 00 18 B8**
3eme LongWord de cette adresse mémoire = **00 01 6E F8**



On remarque que le 1er champ a la même valeur que **D5** du **level1** (adresse mémoire de départ)
Que le 3eme champ a la même valeur que **D4** du **LEVEL1** (Longueur de donnée à copier)

Le code utilise donc bien un tableau en mémoire pour récolter les informations qu'il a besoin pour charger les données.
Par contre, pour les plus attentifs, on voie sur l'image ci-dessus que le pour le **LEVEL2,** le 1er champ *LongWord*, à savoir :
00 31 6E F8 ne correspond pas à ce que l'on a noté pour ce Level dans notre tableau (attendu = 50858)

C'est tout à fait normal, en fait, si vous regarder ce tableau avant le chargement du **LEVEL1** ou de l'animation principale
Il aura une toute autre tête et si d'ailleurs vous avez bien compris la **partie 5 de ce tuto**, vous savez que c'est normal.
Ce tableau est mis à jour via la routine **trackloader** lors d'un appel de celle-ci.

Nous n'allons pas entrer dans les détails mais, ce qu'il faut retenir c'est que les informations pour charger les données se trouve dans un tableau en mémoire. Que **A0** contient, lors de l'appel de la routine *Trackloader*, l'adresse à lire de ce tableau.
Qu'ensuite le code va utiliser ce tableau (et le modifier) pour positionner les valeurs de Taille et de départ dans **D4** et **D5**

Comme le but n'est pas de maitriser parfaitement le fonctionnement de ce trackloader, on ne va pas entrer dans les détails.
Au final, ce qui nous intéresse c'est de pouvoir ripper nos données correctement et là, nous avons toutes les informations nécessaires.

Part 9 Sauvegarde des pistes custom part #2 – In Game

Rebooter de nouveau votre amiga, **sur l'anim. Principale à l'écran**, **entrer** dans l'**AR**
Taper : `BS 2B72E` puis `BS 2B7FC` sans oublier `BS 2B888` et pour finir `X`

De retour à l'animation, **appuyer sur FIRE**
Notre *BreakPoint* 2B72E 'routine trackloader' est atteint quasiment tout de suite, **taper**: `R`
Regarder la valeur de **A0**, elle est à **29F3A**

Taper : `M 29F3A`

```
m 29F3A
:029F3A 00 00 D3 00 00 08 79 8C 00 00 93 D8 00 06 00 00 ......y.........
```

Adr. Source
Nbr Donnée à copier

Taper : `X`
Le *BreakPoint* 2B888 'Position Atteinte' est rapidement atteint, **taper**: `R`

On retrouve en D4 et D5 nos valeurs **d'adresse Source** et de **donnée à copier**.

```
Breakpoint raised at address: 0002B888
r
D0=00001300 00000007 000018B8 000018B8  000093D8 0000D300 55555555 00087907
A0=00029F46 000610F4 0001119F 00073D88  00BFD100 0002E116 00DFF024 000002E4
PC = 0002B888 USP = 42E2338B SR = 2004 T=0 S=1 I=000 X=0 N=0 Z=1 V=0 C=0
```

Taper : `X`
Le jeu continu son chargement (de 6 pistes) et notre *BreakPoint en 2B7FC* **Stop Floppy** alors atteint.
Ce que nous avons appelé '**Pre-Loader'** dans notre tableau est complétement chargé.
Il nous reste plus qu'à le sauver

/!\ Enlever la disquette originale de toki dans le lecteur
Comme il est fort probable que tout cela ne tienne pas sur notre disquette de départ,
formater une nouvelle disquette que vous pouvez appeler **TOKI_FILE2 et insérer celle-ci dans votre Amiga**

On sauve, **taper :** `SM Pre-Loader, D300 D300+93D8`

```
SM Pre-Loader, D300 D300+93D8
Disk ok
```

On peut regarder ce que l'on a sauvé, **taper :** `M D300-10` puis `M 166D8-10`
Ps : D300+93D8=166D8

```
m D300-10
:00D2F0 FF FF FF FF FF C0 00 00 00 00 00 00 00 00 00 00 ................
:00D300 33 C0 00 00 DE BC 23 C8 00 00 D4 D0 23 C9 00 00 3.....#.....#...
:00D310 D4 D4 23 CA 00 00 D4 D8 4D F9 00 DF F0 00 3D 7C ..#.....M..β..=|

m 166D8-10
:0166C8 EA A0 B8 9F AF C6 E7 C0 AF CA D8 C8 00 00 00 00 ................
:0166D8 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

Taper : `X`
La mini animation de level apparait furtivement et notre *BreakPoint* 2B72E 'routine trackloader' est atteint.
Taper: `R` et regarder la valeur de **A0**, elle est à **29EE6**

Taper : `M 29EE6`

```
m 29EE6
:029EE6 00 05 08 58 00 00 18 B8 00 01 6E F8 00 31 6E F8 ...X......n..1n.
```

Taper : `X`

Après un déplacement rapide de la tête du floppy, le *BreakPoint* 2B888 'Position Atteinte' est rapidement atteint.
Taper: `R`

```
Breakpoint raised at address: 0002B888
r
D0=000000FF 00000001 000018B8 000018B8  00016EF8 00050858 55555555 00001801
A0=00029EF2 0002B72E 00013376 00013B58  00BFD100 0002E116 00DFF024 000002E0
PC = 0002B888 USP = 42E2338B SR = 2004 T=0 S=1 I=000 X=0 N=0 Z=1 V=0 C=0
```

Taper : `X`
Un plus long chargement intervient, en l'occurrence le **LEVEL1** (14 pistes) puis on atteint notre *BreakPoint* : *2B7FC* 'Stop Floppy'
Le **LEVEL1** est complétement chargé.

On peut regarder le contenu mémoire en question, **taper : `M 50858`**

```
M 50858
:050858 49 63 65 21 00 01 6E F7 00 02 DE E0 06 02 04 97 Ice!..n.........
:050868 7B C6 54 58 8C 39 CA C2 03 70 01 2F 01 22 6F 0E {.TX.9...p./."o.
:050878 27 80 12 14 A3 48 A2 EB 02 30 B3 9C 97 1C 85 5B '....H...0.....[
```

On sauve tout ça ☺

/!\ Enlever la disquette originale de Toki dans le lecteur et **insérer notre disquette TOKI_FILE**
Taper `SM LEVEL1, 50858 50858+16EF8`

/!\ Réinsérer la disquette originale de toki dans le lecteur
Taper `X` puis **appuyer sur FIRE** pour retourner au code du jeu, nous sommes toujours dans le **LEVEL1**

Maintenant c'est simple, nous allons utiliser passer en mode '*cheat*' pour ripper tout les niveaux du jeu.
Taper sur le clavier de votre amiga le mot ***killer.***

Je vous invite à faire de même pour les autres **LEVEL (touche F2 à F7)**
à savoir :

1. D'appuyer sur la touche de Fonction du **LEVEL** que vous voulez sauver (exemple **F5**=**LEVEL5**)
2. De bypasser la partie 'pre-loader' que l'on a déjà sauvé mais qui va se charger à chaque début de phase de chargement d'un niveau.
3. Lorsque le *BreakPoint 2B888* '***Position Atteinte'*** est atteint, Afficher les registres et noter **D4** et **D5**
   Ils doivent bien sur correspondre à ce que l'on a noté dans notre tableau (et ça sera le cas).
4. Retourner au code jusqu'au *BreakPoint 2B7FC* **Stop Floppy**
5. Sauver le tout grâce au valeur noté de **D4** et **D5** en oubliant pas de swapper les disquettes original<>sauvegarde

Si le jeu bug :
1. **Rebooter** votre Amiga
2. Charger le jeu jusqu'au **LEVEL1** et passer en mode '*cheat'.*
3. **Entrer dans l'AR** et placer nos *BreakPoints* : *2B888* '***Position Atteinte'*** et *2B7FC* '***Stop Floppy'***
   Ps : Le *BreakPoint 2B7FC* n'est pas utile pour le rip des données.
4. Retourner dans le code du jeu `X` et de continuer les étapes 1 à 5 au-dessus.

<u>Deux petites nuances :</u>

- La touche **F8** qui va charger directement l'animation de fin et ne passera pas par **'Pre-Loader'**
  F8 nous emmènera sur l'animation de fin du jeu, sauver la sous le nom de **Anim-End** comme prévue dans notre tableau.

- N'oubliez pas de sauver aussi la partie **HighScores**, pour y accéder c'est simple, perdez toutes vos vies et laisser le jeu revenir justement sur la table des HighScores, nos *BreakPoints* étant posés, vous pourrez sauver cette partie (*29F46 pour info*)

Nos deux disquettes de RIP devraient ressembler à ça :

```
dir
Directory of (TOKI_FILE)
      016456  77-79_decomp
      010000  75-76
      126560  55-74
      130388  16-25_decomp
      018984  77-79
      184408  55-74_decomp
      033816  75-76_decomp
      056203  16-25
0552 blocks free, 68.6 % of disk used
Disk ok
```

```
dir
Directory of (TOKI_FILE2)
      093944  Level1
      083200  Level2
      082672  Level3
      079084  Level4
      085360  Level5
      061808  Level6
      015040  Scores
      037120  Anim-End
      037848  Pre-Loader
      063008  LevelFinal
0417 blocks free, 76.3 % of disk used
Disk ok
```

Part 10 Patch du fichier 16-25, credits/logo Ocean

Après toutes les analyses/relevées que l'on a effectué, il est temps de mettre en place une stratégie pour le fonctionnement de notre futur crack.

On sait déjà que les fichiers ripé **77-79, 75-76, 56-74,** sont indépendant les uns des autres (voir tableau et début d'analyse).
Il suffit par exemple de charger le fichier **75-76_decomp** en mémoire et l'exécuter pour qu'il fonctionne et charge ensuite le 'prochain fichier'.

Ces fichiers d'ailleurs n'apparaissent PAS dans le tableau mémoire vue en *$29EE6* **'Part 8'** de ce tuto.

On sait aussi que le code principale se trouve dans le fichier **55-74**, on a bien sur besoin de ce fichier.
Il serait aussi appréciable d'avoir les credits/logo ocean affiché, fichier **75-76**
Sans oublier bien sûr l'animation principale, fichier **16-25**

Pour pouvoir charger ces fichiers sans qu'ils utilisent leur routine de *trackloader* original, il va donc falloir la trouver (sur chaque fichier) et la patché, bon… ca ne fait que 2 fichiers à examiner voir un seul car on a déjà analysé le fichier **16-25** en **'Part 8'** de ce tuto.

On s'attaque donc au fichier credits/logo Ocean, fichier **75-76,** bien sûr, dans sa version décompressé.
Disquette TOKI_FILE dans le lecteur, entrer dans l'**AR.**
Taper `LM 75-76_decomp, 60000`

On va chercher la routine *TrackLoader (voir détail plus haut dans le tuto)*
Taper `F 3C 55 55 55 55`

Adresse trouvé : **$604E5**
Taper `D 604E5-20`

Descendez un peu dans le code, il s'agit bien de notre *TrackLoader* et on peut voir qu'il commence en **$164D0**
On va chercher où le code fait appel à cette adresse, **Taper** `FA 604D0`
Adresse trouvé : **$6009E**

```
f 3C 55 55 55 55
Search from: 000000 to: C80000
0604E5
Ready.

d 604e5-20
~0604C4 ORI.?   #600EE,64D8(A4)
~0604CC LINEF
~0604CE LINEF
~0604D0 LEA     606CC(PC),A0
~0604D4 LEA     606B0(PC),A5
~0604D8 LEA     00BFD100,A4
~0604DE LEA     00DFF024,A6

fa 604D0
Search from: 000000 to: C80000
06009E BSR      000604D0
Searched up to adr: C80000
Ready.
```

Bon, comme on dit, il n'y a plus qu'à ☺

Part 11 Code - Bootsecteur

Afin de pouvoir utiliser nos fichiers rippé, il va nous falloir écrire un peu de code.
Nous allons avoir besoin :

- De recréer un bootsecteur qui va charger notre programme.
- De créer une programme de décompression du format Ice.
- D'écrire un version modifier du *trackloader* pour qu'il fonctionne avec notre futur disquette.
- De créer ce programme qui va charger tout ça en mémoire et l'utiliser (comprendre patcher le code original)

Tout cela va se passer sous **ASMone** dans notre cas.
Charger donc l'assembleur, réserver un peu de mémoire chip, 100ko suffira largement pour commencer.

```
ASM-One V1.20 By T.F.A. Source »

   ----  ASM-One V1.20 MC680x0/MC6888x Macro Assembler (KS 1.2/1.3) ----

      Original coding by Rune Gram-Madsen          (1990-1991)
      Additional coding by T.F.A.                  (1991-1992)
      Additional 680x0/6888x coding by T.F.A.      (1992-1993)
      Release date 19-09-1993 by T.F.A.

        Changed standard directory to » SOURCES:

ALLOCATE Fast/Chip/Publ/Abs>chip
WORKSPACE (Max.386) KB>100
>
```

Passer en mode édition **<ESC>**, **taper** ces quelques lignes de code :

```
ASM-One V1.20 By T.F.A. Source » newboot.s
; TOKI
; original boot block
start:

  dc.b 'DOS',0
  dc.l 0
  dc.l $370

;-----------------------------------------------
bootcode:
  move.w        #2,$1c(a1)
  move.l        #$7f000,$28(a1)
  move.l        #2*512,$2c(a1)  ; offset
  move.l        #4*512,$24(a1)
  movea.l       4.w,a6
  jsr   -$1c8(a6)
  lea   $7f000,sp
  jmp   $7f000
sizeb=*-bootcode

  dc.b '-- TOKI bootloader! --Disk routine and protection written '
  dc.b 'by Pierre ADANE and cracked by VrS! ',0

end:
<END>

Line:    25  Col:   1  Bytes:     426  Free:  537/  283  A*--- Time: 09:02:00
```

```
                                ; Utilisation du trackdisk device au boot via A1 et l'execbase en A6
move.w  #2,$1c(a1)              ; mode 2 = en position lecture
move.l  #$7f000,$28(a1)         ;  $28(a1) = IO_DATA, adresse de destination en chip
move.l  #2*512,$2c(a1)          ;  $2c(a1) = IO_OFFSET, secteur de départ en octet (sect*512)
move.l  #4$512,$24(a1)          ;  $24(a1) = IO_Length, longueur de l'I/O en octet (sect*512)
movea.l 4.w,a6                  ;
jsr     -$1c8(a6)               ; Appel du trackdisk device qui va donc copier ce qui se trouve en secteur 2 vers $7f000
lea     $7f000,sp               ; Secteur 2 qui contiendra notre loader/patch code principale.
jmp     $7f000                  ; Une fois le pointeur de pile mis à jour à $7f000, on saute vers celui-ci.
```

Insérez votre 1er disquette crée : **TOKI_FILE** dans le lecteur de disquette externe.
Appuyer sur la touche **<ESC>** pour revenir en ligne de commande.
Taper `a` pour assembler puis `wb DF1:newboot.bin` pour sauver le tout.
Utiliser bien sur les marqueur **start** et **end** pour les info **BEG** et **END** demandées par AsmOne

```
Line:    25  Col:   1  Bytes:     426  Free:  537/  284  A*--- Time: 09:06:18
>a
Pass 1..
Pass 2..
No Errors
>wb DF1:newboot.bin
BEG>start
END>end
File length =        157 (=$0000009D )
>
```

Voilà déjà une bonne chose de faite.

Part 12 Code - Décompacteur

Comme vous l'avez deviné, si on décompacte les fichiers rippé, tout ne tiendra pas sur une disquette.
On va donc utiliser les fichiers rippé brut mais obligatoirement, on va avoir deux choix.
Soit complétement décortiqué le code original pour trouver la routine de décompression et la pacthé avec toute la problématique que cela implique pour le reste du code original.
Soit tout simplement, créer notre propre code pour décompresser nos fichiers rippé. C'est cette dernière option que l'on a choisie.
Elle a l'avantage de ne pas interférer avec le code original.

Utiliser soit un assembleur, soit un éditeur de texte simple, de toute façon, il sera compilé plus tard.
Sauver le fichier crée sur votre disquette **TOKI_FILE**, nommé le : **tokiunice.asm**

```
; UnICE from TOKI credits part
UnICE:          movem.l d0-a6,-(sp)
                addq.l  #4,a0           ; skip "Ice!"
                ;bsr.s  sub_0_60106
                ;cmpi.l #'Ice!',d0      ; test if it's an ICE cream!
                ;bne.s  loc_0_60100
                bsr.s   sub_0_60106
                lea     -8(a0,d0.l),a5
                bsr.s   sub_0_60106
                move.l  d0,(sp)
                lea     $6C(a0),a4
                movea.l a4,a6
                adda.l  d0,a6
                movea.l a6,a3
                movea.l a6,a1
                lea     unk_0_602B6(pc),a2
                moveq   #$77,d0
loc_0_600D4     move.b  -(a1),-(a2)
                dbf     d0,loc_0_600D4
                bsr.w   sub_0_6013E
                bsr.s   sub_0_60112
                move.l  (sp),d0
                lea     -$78(a4),a1
loc_0_600E6     move.b  (a4)+,(a1)+
                dbf     d0,loc_0_600E6
                subi.l  #$10000,d0
                bpl.s   loc_0_600E6
                moveq   #$77,d0
                lea     unk_0_602B6(pc),a2
loc_0_600FA     move.b  -(a2),-(a3)
                dbf     d0,loc_0_600FA

                movem.l (sp)+,d0-a6
                rts
;---------
sub_0_60106:    moveq   #3,d1
loc_0_60108     lsl.l   #8,d0
                move.b  (a0)+,d0
                dbf     d1,loc_0_60108
                rts
;----------
sub_0_60112:    bsr.s   sub_0_6016A
                bcc.s   loc_0_60138
                moveq   #0,d1
                bsr.s   sub_0_6016A
                bcc.s   loc_0_60132
                lea     word_0_60216(pc),a1
                moveq   #4,d3

loc_0_60122     move.l  -(a1),d0
                bsr.s   sub_0_60190
                swap    d0
                cmp.w   d0,d1
                dbne    d3,loc_0_60122
                add.l   $14(a1),d1

loc_0_60132     move.b  -(a5),-(a6)
                dbf     d1,loc_0_60132

loc_0_60138     cmpa.l  a4,a6
                bgt.s   loc_0_6019E
                rts
;---------
sub_0_6013E:    moveq   #3,d0

loc_0_60140     move.b  -(a5),d7
                ror.l   #8,d7
                dbf     d0,loc_0_60140
                rts
```

```
;----------
loc_0_6014A:    move.w  a5,d7
                btst    #0,d7
                bne.s   loc_0_60158
                move.l  -(a5),d7
                addx.l  d7,d7
                bra.s   loc_0_60196

loc_0_60158     move.l  -5(a5),d7
                lsl.l   #8,d7
                move.b  -(a5),d7
                subq.l  #3,a5
                add.l   d7,d7
                bset    #0,d7
                bra.s   loc_0_60196
;---------
sub_0_6016A:    add.l   d7,d7
                beq.s   loc_0_60170
                rts
loc_0_60170     move.w  a5,d7
                btst    #0,d7
                bne.s   loc_0_6017E
                move.l  -(a5),d7
                addx.l  d7,d7
                rts
loc_0_6017E     move.l  -5(a5),d7
                lsl.l   #8,d7
                move.b  -(a5),d7
                subq.l  #3,a5
                add.l   d7,d7
                bset    #0,d7
                rts
;-----------
sub_0_60190:    moveq   #0,d1

loc_0_60192     add.l   d7,d7
                beq.s   loc_0_6014A

loc_0_60196     addx.w  d1,d1
                dbf     d0,loc_0_60192
                rts
;----------
loc_0_6019E:    lea     word_0_6022A(pc),a1
                moveq   #3,d2

loc_0_601A4     bsr.s   sub_0_6016A
                dbcc    d2,loc_0_601A4
                moveq   #0,d4
                moveq   #0,d1
                move.b  1(a1,d2.w),d0
                ext.w   d0
                bmi.s   loc_0_601B8
                bsr.s   sub_0_60190

loc_0_601B8     move.b  6(a1,d2.w),d4
                add.w   d1,d4
                beq.s   loc_0_601DE
                lea     word_0_60234(pc),a1
                moveq   #1,d2

loc_0_601C6     bsr.s   sub_0_6016A
                dbcc    d2,loc_0_601C6
                moveq   #0,d1
                move.b  1(a1,d2.w),d0
                ext.w   d0
                bsr.s   sub_0_60190
                add.w   d2,d2
                add.w   6(a1,d2.w),d1
                bra.s   loc_0_601F0
```

```
;---------
loc_0_601DE:    moveq   #0,d1
                moveq   #5,d0
                moveq   #0,d2
                bsr.s   sub_0_6016A
                bcc.s   loc_0_601EC
                moveq   #8,d0
                moveq   #$40,d2

loc_0_601EC     bsr.s   sub_0_60190
                add.w   d2,d1

loc_0_601F0     lea     2(a6,d4.w),a1
                adda.w  d1,a1
                move.b  -(a1),-(a6)

loc_0_601F8     move.b  -(a1),-(a6)
                dbf     d4,loc_0_601F8
                bra.w   sub_0_60112
;----------
                dc.w    $7FFF,14,$FF,7,7,2,3,1,3,1
word_0_60216:   dc.w    0,$10D,0,14,0,7,0,4,0,1
word_0_6022A:   dc.w    $901,$FF,$FF08,$402,$100
word_0_60234:   dc.w    $B04,$700,$120,0,$20
                dc.w    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
                dc.w    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
                dc.w    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
unk_0_602B6:
```

Part 13 Code - Trackloader

Afin de rester dans l'idée du code original, code qui fonctionne plus en *SectorLoader* qu'en *TrackLoader* et afin de maitriser le placement des fichiers sur notre disquette et d'optimiser la place, nous avons aussi besoin d'un *TrackLoader* qui va fonctionner en *Secteur.*

De plus, il nous est impossible d'utiliser le *trackloader* original car, comme vous avez pu le voir précédemment, il est multiple d'une part et complétement éclaté dans le code du jeu.

Toujours dans l'idée d'aller au plus simple, nous allons donc créer notre propre *TrackLoader* ou devrais-je dire *SecteurLoader.*
Par contre, il va nous falloir trouver en emplacement mémoire ou stocker celui-ci…

Lancer le jeu et entrer dans l'action replay sur plusieurs étapes (animation, score, Level1, Level3…) et regarder la valeur de la **SP**

	SP
pre-loader	2F8
Animation start	2F8
highscore	2F8
level1	2FC
level2	2F8

```
r
D0=00000000 00000FCA 000001B8 00000E2E  0000003C 00000003 00000001 00001B83
A0=00060A8E 00077208 0001E264 00077208  00000000 00DFF000 00072B5C 000002F8
PC = 000608E4 USP = 42E2338B SR = 2000 T=0 S=1 I=000 X=0 N=0 Z=0 V=0 C=0

d 2F8
~0002F8 ORI.B   #B4,D6
~0002FC ORI.B   #7E,D2
~000300 RTS
;=====================================
~000302 MOVE.W  A5,D7
```

Sur l'animation principale, entrer dans l'**AR**, on va remplir cette zone pour voir si et jusqu'à où elle est utilisée.
Taper : `O "AA", 302 23800-1`

Jouer au jeu, aller sur les écrans des HighScores, de l'animation du début, de la mini-intro des level, etc…
Bref… faite bosser le plus possible le code original normalement puis, **entrer dans l'AR et taper :** `F "AA", 302`
Notre variable de remplissage (**AA**) est encore visible jusqu'en **$3FC**

```
m 3FC-20
:0003DC 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
:0003EC 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
:0003FC 41 41 07 3F 89 81 89 81 89 81 9D 91 9D 91 9D 91 AA.?............
:00040C 9D 91 00 F0 0C 3F FF FF FF FF FF FF FF E1 90 00 .....?..........
:00041C 42 7F 12 00 12 00 12 00 32 48 32 48 32 48 32 48 B▒......2H2H2H2H
:00042C 1C 48 12 4D FF FF FF FF FF FF FF C1 90 00 C2 61 .H.M...........a
:00043C 24 38 24 38 24 38 29 24 29 24 29 24 29 24 0B 24 $8$8$8)$)$)$)$.$
:00044C 11 4B FF FF FF FF FF FF FF C1 08 18 A3 C0 2C 68 .K.............,h
-
```

Pas besoin de dire qu'une plage de **$300** à **$3FC** est beaucoup trop petite pour y coder un *TrackLoader*
Voyons voir plus loin.

L'utilisation de la mémoire (relevé précédemment via notre tableau et une fois les fichiers compressé, décompressé) est celle-ci :

Buffer code principale, fichier **55-74** — **23800 -> 50858** (*55-74 décompressé = 2D058, 23800+2D058=50858*)
Buffer **level**S — **50858 -> 655C8** (Level1 décompressé = 2DEE0, 50858+2DEE0 = 7E738)
buffer scores, pre_loader, Anim_End, Anim_Start, **75-76** — **60000 -> 7FD54** (*db8c max mais compressé, décompressé =1FD54*)

Le jeu doit tenir sur 512Kb de chip (il est vendu fonctionnel pour cette configuration minimum).
Donc maximum il peut adresser jusqu'à **$7FFFF**.

$7FFFF semble utilisé, **Taper :** `M 7FFFF-20`

```
m 7FFFF-20
:07FFDF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
:07FFEF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
:07FFFF 00 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
```

ça nous donne donc *: (7FFFF-1)-7FD54 = 2AA = 682 Octets*
On a donc **682 octet** de libre ici pour loger notre *Trackloader*...

Le plus petit *TrackLoader* que nous connaissons est celui **d'Alpha One en 2004**, il fait *404 octets*
On a le choix entre l'adapter en *SectorLoader* et diminuer le code, modifier et utiliser le code original ou coder le nôtre.

Utiliser soit un assembleur, soit un éditeur de texte simple, de toute façon, il sera compilé plus tard.
Sauver le fichier crée sur votre disquette **TOKI_FILE**, nommé la **trackloader.asm**

```
; raw sector mfm loader
;
; TOKI modification:
; a0=file info (buffer.L,start sector*512.L,len sector*512.L)

; bits CIAA-PRA --------------------------- bits signification
DSKRDY  =       5               ;drive ready
DSKTRK0 =       4               ;track 0 reached
DSKPROT =       3               ;disk write protected
DSKCHNG =       2               ;disk change
; bits CIAB-PRB ---------------------------
DSKMTR  =       7               ;control motor
DSKSEL3 =       6               ;select DF3
DSKSEL2 =       5               ;       DF2
DSKSEL1 =       4               ;       DF1
DSKSEL0 =       3               ;       DF0
DSKSIDE =       2               ;select side/head
DSKDIR  =       1               ;direction
DSKSTEP =       0               ;move head

TRKLEN=11*512
SYNC=$4489

DOSTRACKLOADER:
                movem.l d0-a6,-(sp)
                lea     $DFF000,a6      ; CUSTOM
                lea     $BFD100,a4      ; CIAB-PRB / +$F01 = CIAA-PRA
                ; init CIA Timer B
                move.b  #$00,$500(a4)   ; timer B lo $bfd600
                move.b  #$0d,$600(a4)   ;         hi $bfd700
                clr.b   $e00(a4)        ; crb

                lea     track(pc),a2    ; current track position
                lea     $1000.w,a1      ; MFM buffer
                ; TOKI file infos
                move.l  4(a0),d0        ; sector offset
                move.l  8(a0),d1        ; len bytes
                movea.l (a0),a0         ; dest buffer

                movea.l a0,a5
                adda.l  d1,a5           ; dest buffer+len

                move.b  #$7D,(a4)       ; start drive %01111101
                bclr    #DSKSEL0,(a4)   ; DSKSEL0 select DF0
                bsr     Delay

                clr.l   d4
                divu.w  #TRKLEN,d0      ; (start sector*$1600)/$1600 = num sector|num track (ex. 13*$1600)
                swap    d0              ; (ex. d0=$0400 0001 => $0001 0400)
                move.w  d0,d4           ; start sector in track 0-10 (rest of division= $400 = 2*512)
                swap    d0              ; num track (ex. $0400 0001 d0=1)
                bsr     SeekTrack       ; move head to track

                moveq   #6-1,d5         ; try
.load           move.l  a1,$20(a6)      ; DSKPTH MFM Buffer
                move.w  #$8210,$96(a6)  ; DMACON disk on
                move.w  #$7F00,$9E(a6)  ; erase ADKCON
                move.w  #$9500,$9E(a6)  ; correct value of ADKCON
                move.w  #$4000,$24(a6)  ; erase DSKLEN
                move.w  #SYNC,$7E(a6)   ; DSKSYNC standard Sync        Word

                bsr.s   DiskReady

                move.w  #$8000+$1900,d1 ; track size standard 9900
                move.w  d1,$24(a6)      ; DSKLEN
                move.w  d1,$24(a6)      ; 2 times to start DMA transfert
                move.w  #2,$9C(a6)      ; INTREQ dsk interrupt
```

```
.wait           btst    #1,$1F(a6)      ; INTREQR disk ready ?
                beq.s   .wait           ; not yet

                move.w  #0,$24(a6)      ; erase DSKLEN

                bsr.w   MFMUncode       ; decode mfm buffer
                bne.s   .error
                cmpa.l  a0,a5           ; end ?
                ble.s   .finished       ; yep

                bsr.w   NextTrack       ; next
                bra.s   .load

.finished       move.b  #$FD,(a4)       ; stop drive %11111101
                bclr    #DSKSEL0,(a4)   ; select DF0
                bset    #DSKSEL0,(a4)   ; deselect DF0
                bsr     Delay
                movem.l (sp)+,d0-a6
                rts

.error          subq.l  #1,d5           ; try again ?
                bne.s   .load
;- error
infiniteloop:   move.w  d0,$180(a6)     ; flash color0
                addq.l  #1,d0
                bra.s   infiniteloop
;-------------------------------------------
; Functions
;-------------------------------------------
DiskReady:      btst    #DSKRDY,$F01(a4)        ; CIAA-PRA
                bne.s   DiskReady
                rts
;--------
Seek0:          btst    #DSKTRK0,$F01(a4)       ; CIAA-PRA DSKTRK0
                beq.s   .found
                bset    #DSKDIR,(a4)    ; DSKDIR goto 0
                bsr.s   MoveHead
                bra.s   Seek0
.found          clr.b   (a2)            ; track 0
                rts
;------
NextTrack:      addq.b  #1,(a2)         ; current track+1
                move.b  (a2),d1
                btst    #0,d1           ; odd or even track ?
                bne.s   .changeSide
                bclr    #DSKDIR,(a4)    ; DSKDIR goto 79
                bsr.s   MoveHead
.changeSide     bchg    #DSKSIDE,(a4)   ; DSKSIDE switch side 0/1
                bsr.w   Delay
                rts
;------
MoveHead:       bset    #DSKSTEP,(a4)   ; inactive
                nop
                nop
                bclr    #DSKSTEP,(a4)   ; active
                nop
                nop
                bset    #DSKSTEP,(a4)   ; inactive
                bsr.w   Delay           ; max 3 ms
                bra.s   DiskReady
;-------------
SeekTrack:      movem.l d1-d2,-(sp)

                cmpi.b  #-1,(a2)        ; current track ok ?
                bne.s   .ok
                bsr.s   Seek0           ; no. seek track 0
.ok             clr.l   d1
                clr.l   d2
                move.b  (a2),d1         ; current track
                move.b  d0,d2           ; wanted track
                bset    #DSKSIDE,(a4)   ; DSKSIDE side 0
                btst    #0,d0           ; odd or even track
                beq.s   .side0
                bclr    #DSKSIDE,(a4)   ; DSKSIDE side 1

.side0          lsr.w   #1,d2           ; track/2 (80 tracks/side)
                lsr.w   #1,d1           ; current track/2
                sub.w   d1,d2           ; track to reach-current track = direction
                bcs.s   .negat          ; <0
                bhi.s   .posit          ; >0
                bra.s   .found          ; =0
```

```
.posit          bclr    #DSKDIR,(a4)    ; DSKDIR goto 79
                bra.s   .moveh

.negat          neg.w   d2              ; positive
                bset    #DSKDIR,(a4)    ; DSKDIR goto 0

.moveh          bsr.s   MoveHead
                subq.w  #1,d2           ; goto the track
                bne.s   .moveh

.found          move.b  d0,(a2)
                movem.l (sp)+,d1-d2
                rts
;-----------------------------------------------
; standard MFM format (see explanation below)
; a0=dest buffer
; a1=mfm buffer
MFMUncode:      movem.l a1-a6,-(sp)
                movea.l a0,a2           ; dest buffer
                movea.l a0,a3
                adda.l  d4,a3           ; dest buffer+start sector
                movea.l a0,a4
                cmpi.w  #SYNC,(a1)      ; find sync word
                bne.s   .error

                moveq   #11-1,d5        ; 11 sectors
.loopsector:    bsr.s   UncodeHeader
                moveq   #128-1,d6       ; 128 words to decode (512/4)
.UncodeSector   move.l  512(a1),d1      ; even bits
                move.l  (a1)+,d0        ; odd bits
                eor.l   d0,d2
                eor.l   d1,d2
                bsr.s   uncode
                cmpa.l  a3,a2           ; real start reached ?
                bge.s   .ok
                addq.l  #4,a2           ; not yet
                bra.s   .next           ; skip sector

.ok             movea.l a2,a6
                addq.l  #4,a2
                suba.l  d4,a6
                cmpa.l  a5,a6           ; end of dest buffer reached ?
                bge.s   .next           ; yes. skip sector
                move.l  d0,(a6)+        ; store decoded data

.next           dbf     d6,.UncodeSector

                andi.l  #$55555555,d2   ; mfm bits Mask %010101...
                cmp.l   d2,d7
                bne.s   .error
                dbf     d5,.loopsector

                ; sector precision :
                ; track 1 [**|*********] start at sector 13
                ; len=56320. start=(13*512)/(11*512) = 13/11 = 1.2 = track 1, 2 sector
                ; 5632-2*512. start buffer+2*512
                move.l  #TRKLEN,d0      ; standard size 11*512
                sub.l   d4,d0           ; - start sector
                adda.l  d0,a0           ; skip lower sectors
                clr.l   d0
                clr.l   d4

.exit:          movem.l (sp)+,a1-a6
                tst.l   d0
                rts

.error:         movea.l a4,a0
                moveq   #-1,d0
                bra.s   .exit
```

```
;--------------------
; a0=dest data
; a1=source mfm data
UncodeHeader:   move.w  #SYNC,d1

.GAP:           cmp.w   (a1)+,d1        ; search start of sector
                bne.s   .GAP
                cmp.w   (a1),d1         ; another sync word ?
                bne.s   .ok
                addq.l  #2,a1           ; skip second sync word
.ok:            move.l  48(a1),d0       ; Datas checksum (odd bits)
                move.l  52(a1),d1       ; Datas checksum (even bits)
                bsr.w   uncode
                move.l  d0,d7
                move.l  (a1),d0         ; ID sector (odd bits)
                move.l  4(a1),d1        ; ID sector (even bits)
                bsr.s   uncode          ; d0=format,track,sector,count
                lsr.w   #8,d0           ; shift 8 bits to right to get sector
                lea     56(a1),a1       ; point to Datas area
                add.w   d0,d0           ; *2
                asl.w   #8,d0           ; *256
                lea     (a4,d0.w),a2    ; point to sector start
                clr.l   d2
                rts
;----------------------
uncode:         asl.l   #1,d0
                andi.l  #$AAAAAAAA,d0   ; even bits Mask
                andi.l  #$55555555,d1   ; odd bits Mask
                or.l    d1,d0           ; merge odd and even bits
                rts
;----------------------------------------
; CIA TIMER DELAY
Delay           move.b  #$19,$e00(a4)   ;$bfdf00 start timer
                nop
                nop
.dly:           btst.b  #0,$e00(a4)     ; and wait
                bne.s   .dly
                rts
; var
track           dc.b    -1,0    ; current track
```

Part 14 Code - TokiStarter

Il nous reste maintenant plus qu'a coder notre petit programme principale qui va charger les fichiers du début, les décompresser.
Patcher le loader original pour l'utilisation de nos routines de décompression et notre *SectorLoader*

Encore une fois, lancer **ASMone**, réserver un peu de mémoire chip, 100ko par exemple.
Sauver le fichier crée sur votre disquette **TOKI_FILE**, nommé la **tokistarter.asm**

```
; copy trackloader before main prog
; loaded at 7f000

ADDR=$23800
ADDR2=$7fd54
                Section starter,Code_c
start
; load packed credits
        move.w  #2,$1c(a1)              ; On utilise le trackdisk device
        move.l  #$60000,$28(a1)         ; pour charger en 60000
        move.l  #1733*512,$2c(a1)       ; à l'adresse disk 1733*512 octets
        move.l  #20*512,$24(a1)         ; d'une longueur de 20*512 octet
        movea.l 4.w,a6                  ; fichier 'credit/logo ocean'
        jsr     -$1c8(a6)               ; On lance le chargement.

        move.w  #$7fff,$dff09a
        move.w  #$7fff,$dff09c

        lea     starter(pc),a0          ; On copie la routine starter en 20000
        lea     $20000,a1               ;
        move.l  #(sizet/4)-1,d7         ;
.cp     move.l  (a0)+,(a1)+             ;
        dbf     d7,.cp                  ;

        lea     $20000,a0               ; A0=$20000
        move.l  a0,$80.w                ; Copie de A0 en mémoire $80 (Adr. Standard pour le TRAP #0)
        trap    #0                      ; Et donc exécution du code en 20000, la page credits/logo apparait.

starter
        movea.l #$60000,a0              ; Code 'principale', on décompresse ce qui est en 60000
        moveq   #0,d0                   ; En l'occurrence, le fichier 75-76, 'credit/logo ocean'
        bsr     Decrunch                ; préalablement chargée. On lance la sous routine decrunch
                                        ; car le fichier est compacté. (voir tableau)
        ; patch credits - trackload
        lea     $6009e,a0               ; On patch le saut vers le trackloader d'origine
        move.w  #$4ef9,(a0)+            ; voir 'Part 10', du fichier 75-76
        pea     loadmain(pc)            ; Routine loadmin chargé (chargement des fichier :
        move.l  (sp)+,(a0)+             ; trackloader.asm, tokiunice.asm, 55-74
                                        ; et décompression de ce dernier.

                                        ; start credits -> download main prog at 23800 / return
        jmp     $60000                  ; Et on exécute le tout.

decr    movea.l #$23800,a0              ; Sous routine decr qui va position A0 à $23800
        moveq   #0,d0                   ; (adr. Source à décompacter) ainsi que D0 a 0
        bsr     Decrunch                ; et lancer la décompression en position $23800

; patch main
        ; patch trackloader             ; On patch le saut vers le trackloader d'origine
        lea     $2b72e,a0               ; voir 'Part 8', du fichier 55-74.
        move.w  #$4ef9,(a0)+            ;
        move.l  #ADDR2,(a0)             ; vers notre propre trackloader position en #ADDR2
                                        ; à savoir 7fd54 (voir Part 13).

        ; patch files infos             ; Sans oublier de patcher l'intégralité du tableau orginal
        lea     $29ee6+4,a0             ; commençant en mémoire 29ee6 (voir Part 08)
        lea     Files(pc),a1            ; Avec les nouvelles positions 'standard' AmigaDos
        moveq   #11-1,d7                ; 11 entrée au total à modifier.
.loop   move.l  (a1)+,(a0)+             ; offset
        ;move.l (a1)+,(a0)+             ; len sectors or tracks
        lea     8(a0),a0                ; skip addr and len if original len used
        dbf     d7,.loop                ;
```

```
; do a copy of trackloader at safe place
        lea     Load(pc),a0             ; Bien sûr, il ne faut oublier de copier notre trackloader
        lea     ADDR2,a1                ; à la position calculée/désirée (voir Part 13)
        move.l  #(sizetk/4)-1,d7        ;
cp      move.l  (a0)+,(a1)+             ;
        dbf     d7,cp                   ;

        jmp     $23800                  ; start
;----------------------------------------
loadmain
        lea     FileMain(pc),a0
        bsr.s   Load
        bra     decr
;------------------------------------- Files list
FileMain dc.l   $23800
        dc.l    1485*512        ; 55-74         ; Noté que la 'position' des fichiers suis l'ordre
        dc.l    $1EE60          ; 248*512       ; original (voir notre tableau Part 08)
Files   dc.l    11*512          ; Level1        ;
file2   dc.l    195*512         ; Level2        ; On serre les fichiers le plus possible pour un gain
file3   dc.l    358*512         ; Level3        ; de place, exemple : Fichier Level3 = taille de $142F0
file4   dc.l    520*512         ; Level4        ; 82672 en décimal, ce qui nous donnes 82672/512=161,47
file5   dc.l    675*512         ; Level5        ; Soit 162 secteurs nécessaire.
file6   dc.l    842*512         ; Level6        ; début Level3 = 358 + 162 = Début du level4
file7   dc.l    963*512         ; LevelFinal    ; à savoir secteur 520
File8   dc.l    1087*512        ; Pre-Loader    ; ce qui nous donnes la ligne dc.l    520*512 ; Level4
File9   dc.l    1161*512        ; Scores        ;
File10  dc.l    1191*512        ; 16-25         ;
File11  dc.l    1301*512        ; Anim-End      ;
;------------------------------------- TrackLoader
Load:   include "TOKI_FILE:trackloader.asm"
sizetk=*-Load

;---------------------------------------- DePacker
; ICE Depacker
; a0=source
Decrunch:
        include "TOKI_FILE:tokiunice.asm"

sizet=*-starter

end:
```

Il nous reste plus qu'à assembler tout ça sous **ASMone**. réserver un peu de mémoire chip, 100ko suffira.

```
ASM-One V1.20 By T.F.A. Source » tokistarter.asm

  ----  ASM-One V1.20 MC680x0/MC6888x Macro Assembler (KS 1.2/1.3) ----

     Original coding by Rune Gram-Madsen          (1990-1991)
     Additional coding by T.F.A.                  (1991-1992)
     Additional 680x0/6888x coding by T.F.A.      (1992-1993)
     Release date 19-09-1993 by T.F.A.

        Changed standard directory to » SOURCES:

ALLOCATE Fast/Chip/Publ/Abs>chip
WORKSPACE (Max,386) KB>100
>r TOKI_FILE:tokistarter.asm
File length =       2020 (=$000007E4 )
>
```

Charger notre fichier tout fraichement crée qui devrait se trouver sur la disquette **TOKI_FILE**
Taper : `r TOKI_FILE:tokistarter.asm`

Bien sûr, insérez la disquette **TOKI_FILE** contenant notre/nos fichiers quand demandé.

```
Line:    59  Col:   1  Bytes:    2020  Free:  525/  285  -*--- Time: 08:59:17
>a
Pass 1..
Pass 2..
No Errors
>wb TOKI_FILE:tokistarter.exe
BEG>start
END>end
File length =       1334 (=$00000536 )
>
```

Toujours en mode command Line, **Taper** `a` pour assembler puis `wb TOKI_FILE:tokistarter.exe` pour sauver le tout.
Utiliser bien sur les marqueur **start** et **end** pour les info **BEG** et **END** demander par AsmOne

Part 15 Construction de notre disquette

Il est temps pour nous de créer notre disquette.
Pour cette étape, nous allons devoir placer chaque fichier à l'endroit désiré sur notre disquette.
Il existe plusieurs outils pour faire cela comme le **SectorTrasher** de Silents ou **Mega-mon**…
Dans notre cas, on va utiliser ce dernier !

Il est aussi plus que conseillé d'avoir une second lecteur de disquette connecté à votre Amiga.
Sinon, vous allez devoir swaper de disquette souvent.

Une fois Mega-Mon de lancer, insérer une disquette vierge dans le lecteur DF0, ça sera notre disk CRACKé
et dans **DF1** notre disquette **TOKI_FILE**

L'ordre des fichiers suis logiquement la disposition original des fichiers.
Et l'adresse physique (en secteur) suit logiquement ce que l'on a écrit dans notre fichier **tokistarter.asm**

```
… … …
        move.l   #1733*512,$2c(a1)        ; à l'adresse disk 1733*512 octets
        move.l   #20*512,$24(a1)          ; d'une longueur de 20*512 octet
        movea.l  4.w,a6                   ; fichier 'credit/logo ocean'
… … …


;-------------------------------------- Files list
FileMain dc.l    $23800
        dc.l     1485*512 ; 55-74         ;                  ; Noté que la 'position' des fichiers suis l'ordre
        dc.l     $1EE60                   ; 248*512          ; original (voir notre tableau Part 08)
Files   dc.l     11*512                   ; Level1           ;
file2   dc.l     195*512                  ; Level2           ; On serre les fichiers le plus possible pour un gain
file3   dc.l     358*512                  ; Level3           ; de place, exemple : Fichier Level3 = taille de $142F0
file4   dc.l     520*512                  ; Level4           ; 82672 en décimal, ce qui nous donnes 82672/512=161,47
file5   dc.l     675*512                  ; Level5           ; Soit 162 secteurs nécessaire.
file6   dc.l     842*512                  ; Level6           ; début Level3 = 358 + 162 = Début du level4
file7   dc.l     963*512                  ; LevelFinal       ; à savoir secteur 520
File8   dc.l     1087*512                 ; Pre-Loader       ; ce qui nous donnes la ligne dc.l 520*512 ; Level4
File9   dc.l     1161*512                 ; Scores
File10  dc.l     1191*512                 ; 16-25
File11  dc.l     1301*512                 ; Anim-End
;-------------------------------------- TrackLoader
```

On commence par charger notre *bootsecteur* préalablement compilé.

Taper :

```
L TOKI_FILE:newboot.bin 10000          ; charge le fichier newboot.bin en adr. mémoire $10000
B 10000                                ; calcul le checksum de boot à l'adresse indiqué
>b 10000 0 2                           ; écrit le contenu de l'adresse 10000 au secteur 0, longueur 2 secteurs
```

Taper :

```
L TOKI_FILE:tokistarter.exe 10000      ; Charge le fichier tokistarter.exe en adr. mémoire $10000
>b 10000 2 4                           ; écrit le contenu de l'adresse 10000 au secteur 2, longueur 4 secteurs
                                       ; secteur 2 = à la suite du bootsecteur en adresse +$400
                                       ; 1 block = 1 secteur = 512 octets
                                       ; 2 block = 1024 octet = $400
```

Enlever la disquette TOKI_FILE du lecteur DF1 et insérer la disquette TOKI_FILE2 à sa place.

```
MON>L TOKI_FILE2:Level1 50000
Load $00050000 - $00066ef8
MON>>b 50000 11 184
MON>
```

Taper :

```
L TOKI_FILE2:Level1 50000          ; Charge le fichier Level1 en adr. mémoire $50000
>b 10000 11 184                    ; écrit le contenu de l'adresse 50000 au secteur 11, longueur 184 secteurs
                                   ; Taille du fichier Level1 = $16EF8 = 93944 octets
                                   ; 93944 / 512 = 183.48
                                   ; Donc, 184 secteurs sont nécessaires.
```

```
MON>L TOKI_FILE2:Level2 50000
Load $00050000 - $00064500
MON>>b 50000 195 163
MON>
```

Taper :

```
L TOKI_FILE2:Level2 50000          ; Charge le fichier level2 en adr. mémoire $50000
>b 10000 195 163                   ; écrit le contenu de l'adresse 50000 au secteur 195, longueur 163 secteurs
                                   ; Démarres-en 195 car le Level à commencer en secteur 11 avec une longueur de 184
                                   ; 11+184 = 195
```

Etc pour chaque autre fichier (plus d'image jointes) ➔

Taper :

```
L TOKI_FILE2:Level3 70000
>b 70000 358 162

L TOKI_FILE2:Level4 70000
>b 70000 520 155

L TOKI_FILE2:Level5 70000
>b 70000 675 167

L TOKI_FILE2:Level6 70000
>b 70000 842 121

L TOKI_FILE2:LevelFinal 70000
>b 70000 963 124

L TOKI_FILE2:Pre-Loader 70000
>b 70000 1087 74

L TOKI_FILE2:Scores 70000
>b 70000 1161 30
```

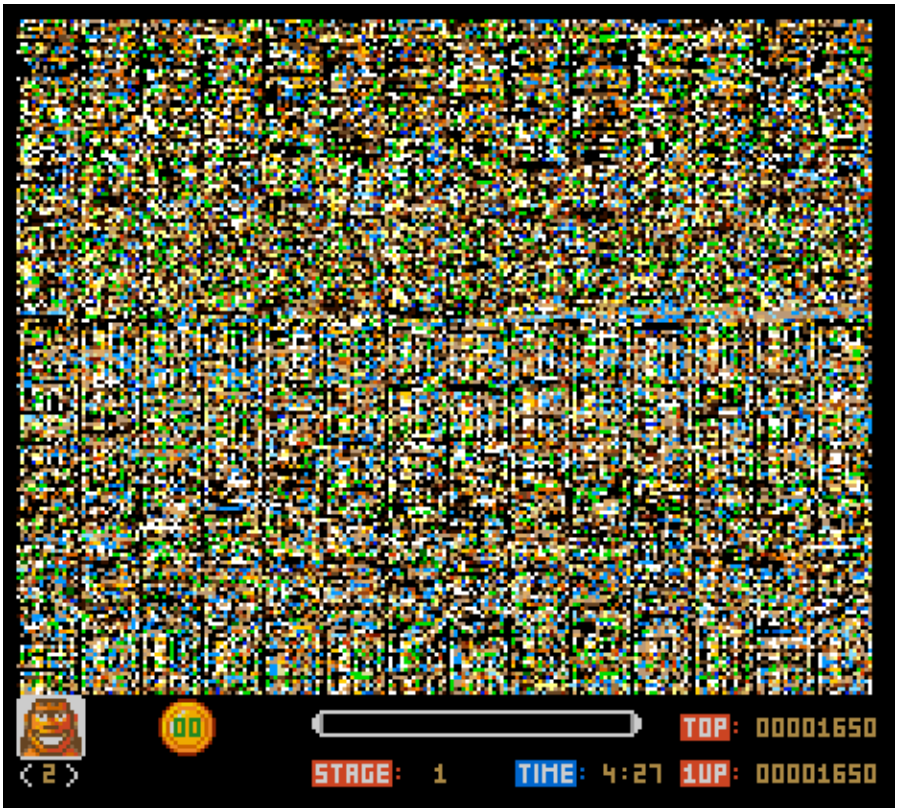
Enlever la disquette TOKI_FILE2 du lecteur DF1 et insérer la disquette TOKI_FILE à sa place.

```
L TOKI_FILE:16-25 70000
>b 70000 1191 110

L TOKI_FILE2:Anim-End 70000
>b 70000 1301 73

L TOKI_FILE:55-74 70000
>b 70000 1485 247

L TOKI_FILE:75-76 70000
>b 70000 1733 20
```

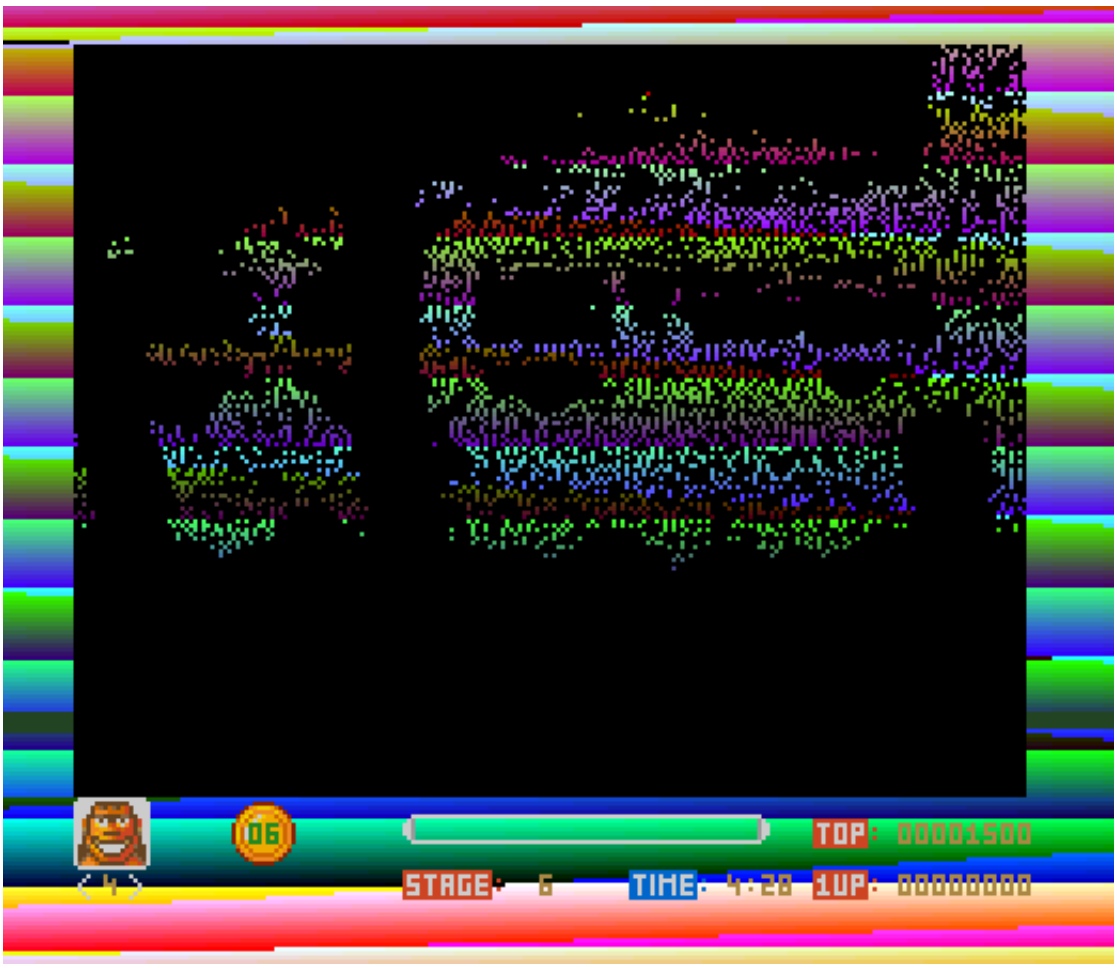
Part 16 Test – part #1

On reboot sur notre disquette fraichement crée.

Comme le montre les images ci-dessus, le jeu se charge et on arrive sur le **Level1**.
Tout semble fonctionner mais…après un certain temps de jeu :



Si on charge directement le **Level2**, autre problème, le jeu commencer à buger directement.



Nous voilà en présence d'une protection ou d'un problème de cohérence liée à notre hack ?!

Part 17 Première protection - Patch Level2 ne fonctionne pas

Rebooter votre amiga, passer en mode *cheatMode*, **appuyer sur la touche F2**
Une fois le **level2** chargé, **appuyer sur FIRE** pour entrer dans celui-ci.

Le jeu part en bug, **entrer dans l'AR**, **taper : R**

```
r
D0=FFFF4416 00000004 00000008 00000000  00000500 00003000 FFFF8095 00000003
A0=00000108 00036FE7 00036F8F 00026528  00036FE2 0000078A 00000676 000002EA
PC = 00FC30D6 USP = 0007F000 SR = 270C T=0 S=1 I=111 X=0 N=1 Z=1 V=0 C=0

m 2EA
:0002EA 00 00 00 03 20 75 40 FE 21 81 20 70 20 0A 00 03 .... u@.!. p ...
```

Quand on voie l'adresse du prochain code à exécuter…c'est clair qu'on a un soucis.
D0 et **D6** attire mon intention, leurs valeurs sort de l'ordinaire par rapport à ce que l'on a pu voir jusqu'à maintenant.
Idem pour la valeur de PC, ***$FC30D6***…étrange

Rebooter votre amiga, passer en mode *cheatMode*, **appuyer sur la touche F2**
Une fois le **level2** chargé, **appuyer sur FIRE** pour entrer dans celui-ci
Et **AVANT que ça crash**, donc assez rapidement **Entrer dans l'AR, taper : R** puis **D**

```
r
D0=BC4DD3EF 696E7366 39CE0C39 0000F000  00001000 00000001 0000FFFF 00002A18
A0=000244A0 0002CD36 0002C2DC 00026528  0007A18E 00026AE4 0002612A 000002F8
PC = 0003DB06 USP = 0007F000 SR = 2000 T=0 S=1 I=000 X=0 N=0 Z=0 V=0 C=0
d
~03DB06 MOVE.L  -(A1),D2

m 2F8
:0002F8 00 03 C2 AC 00 02 A5 8C 00 00 00 00 00 00 00 00 ................
```

D0 toujours avec une valeur étrange.
SP en ***2F8***, prochaine instruction sera en ***$03C2AC*** et on est quelque part en ***$3DB06***

On désassemble en ***03C2AC*** et on remonte jusqu'au début de la routine.

```
~03C268 RTS
;=====================================
~03C26A MOVE.W  #0,00026172
~03C272 MOVEA.L 00023A9E,A4
~03C278 TST.B   00023CD9
~03C27E BNE     0003C286
~03C280 LEA     00027574,A4
~03C286 JSR     0002BED6
~03C28C MOVE.W  #1,00026172
~03C294 TST.B   00023CD9
~03C29A BEQ     0003C2A8
~03C29C MOVE.W  00026174,D0
~03C2A2 MOVE.W  D0,00026172
~03C2A8 BSR     0003CA66
~03C2AC JSR     00037168
~03C2B2 JSR     0003703A
~03C2B8 MOVEA.L #4E806,A6
;=====================================
```

On s'attarde sur les **JSR** et **BSR,** on regarde un peu qu'elle partie du code font appel à elles.
Taper : FA 2BED6 FA 3CA66 FA 37168 FA 3703A

```
fa 2BED6
Search from: 000000 to: 080000
02A444 JSR      0002BED6
02EFDA JSR      0002BED6
02F342 JSR      0002BED6
02F8CA JSR      0002BED6
02FD0C JSR      0002BED6
03013E JSR      0002BED6
03046E JSR      0002BED6
03C286 JSR      0002BED6

fa 3CA66
Search from: 000000 to: 080000
03C2A8 BSR      0003CA66

fa 37168
Search from: 000000 to: 080000
03B472 JSR      00037168
03B79C JSR      00037168
03C2AC JSR      00037168

fa 3703A
Search from: 000000 to: 080000
036EF6 JSR      0003703A
03C2B2 JSR      0003703A
```

Tous son appelé plusieurs fois visiblement sauf ***$3CA66*** qui sort du lot car appelé un seul fois.
On va aller jeter un coup d'œil.

Taper : `D 3CA66-4`

```
d 3CA66-4
~03CA62 MOVEA.W (A4),A5
~03CA64 RTS
!=====================================
~03CA66 LEA     00024482,A0
~03CA6C LEA     000258E6,A1
~03CA72 CLR.B   (A0)+
~03CA74 CMPA.L  A0,A1
~03CA76 BNE     0003CA72
~03CA78 MOVEQ   #0,D0
~03CA7A MOVE.W  000239C8,D0
~03CA80 SUBQ.W  #1,D0
~03CA82 ADD.W   D0,D0
~03CA84 ADD.W   D0,D0
~03CA86 JMP     2(PC,D0.W)
!=====================================
~03CA8A BRA     0003D7C2
!=====================================
~03CA8E BRA     0003DA1E
!=====================================
~03CA92 BRA     0003DCD0
!=====================================
~03CA96 BRA     0003DFB8
!=====================================
~03CA9A BRA     0003E264
!=====================================
~03CA92 BRA     0003DCD0
!=====================================
~03CA96 BRA     0003DFB8
!=====================================
~03CA9A BRA     0003E264
!=====================================
~03CA9E BRA     0003E518
!=====================================
~03CAA2 BRA     0003E660
!=====================================
~03CAA6 ORI.B   #EE,D3
~03CAAA ORI.B   #C4,D3
```

Bon, déjà la routine commence bien en ***$3CA66***
Noter que l'on utilise l'adresse ***$D239C8*** en ***WORD*** pour calculer le saut à effectuer.
On sait déjà qu'à cette adresse se trouve le Numéro de Level (**voir Part3, CheatMode/Trainer...**)

On peut voir que plusieurs opérations sont effectuées avec en grande majorité une utilisation de **D0**
pour ensuite être utiliser comme adresse pour le **Jump** en ***$3CA86***

Au vu du code qui se sert entre autres de **PC** ainsi que les branchements qui suivent, pas besoin de calculer l'adresse du saut en ***$3CA86***
le **JMP** sautera dans la liste des **BRA** en dessous du code à coup sûr.

Rebooter votre amiga, une fois l'animation de départ à l'écran, entrer dans l'**AR**, on va poser quelques *BreaPpoints*
Les deux premiers de la liste des **BRA** ci-dessus par exemple.

```
bs 3D7C2
Breakpoint inserted
Ready.
bs 3DA1E
Breakpoint inserted
Ready.
x
```

Taper `BS 3D7C2` puis `BS 3DA1E` et on retourne au code avec `X`

Notre 1er *BreakPoints* est atteint juste avant que le **Level1** démarre et on entre automatiquement dans l'**AR**
On retourne au code **Taper** `X`
Entrer en mode *cheatMode*, **et appuyer sur la touche F2**

Notre seconde *BreakPoints* est atteint juste avant que le **Level2** commence.

Nous avions vue juste, la liste des **BRA** que l'on a vue au-dessus correspond au branchement des Levels/intro/anim.
Dans tous les cas… pour le **Level2** qui nous pose problème, c'est en *$3DA1E* que ça se passe.

On va désassembler tout ça !

Taper `D 3DA1E`
On va regarder si on trouve des opérations effectuées sur **D0** et si d'autre registres sont aussi utilisés avec **D0**

```
~03DAC2 MOVE.L  000259A2,D0
~03DAC8 LEA     00024482,A0
~03DACE MOVE.L  #35A52,14(A0)
~03DAD6 MOVE.B  #4,A(A0)
~03DADC MOVE.B  #1,9(A0)
~03DAE2 MOVE.B  #4,E(A0)
~03DAE8 MOVE.L  #2420082,10(A0)
~03DAF0 LEA     1E(A0),A0
~03DAF4 MOVE.B  #4,A(A0)
~03DAFA MOVE.B  #1,9(A0)
~03DB00 MOVE.B  #3,E(A0)
~03DB06 MOVE.L  -(A1),D2          D2
~03DB08 MOVE.L  #2C40130,10(A0)
~03DB10 EOR.L   D2,D0
~03DB12 MOVE.L  #35A52,14(A0)
~03DB1A SUBQ.L  #4,D7
~03DB1C BNE     0003DB06
~03DB1E LEA     1E(A0),A0
~03DB22 MOVE.L  #35A52,14(A0)
~03DB2A MOVE.B  #1,A(A0)
~03DB30 MOVE.B  #1,9(A0)
~03DB36 MOVE.L  #2C1037F,10(A0)
~03DB3E MOVE.B  #0,E(A0)
~03DB44 LEA     1E(A0),A0
~03DB48 MOVE.L  #35A52,14(A0)
~03DB50 MOVE.B  #4,A(A0)
~03DB56 MOVE.B  #1,9(A0)
~03DB5C MOVE.B  #4,E(A0)
~03DB62 MOVE.L  #4B6002E,10(A0)
~03DB6A LEA     1E(A0),A0
~03DB6E MOVE.L  #35A52,14(A0)
~03DB76 MOVE.B  #3,A(A0)
~03DB7C MOVE.B  #1,9(A0)
~03DB82 MOVE.B  #2,E(A0)
~03DB88 MOVE.L  #1A009D,10(A0)
~03DB90 LEA     1E(A0),A0
~03DB94 MOVE.L  #35A52,14(A0)
~03DB9C MOVE.B  #6,A(A0)
~03DBA2 MOVE.B  #1,9(A0)
~03DBA8 MOVE.B  #1,E(A0)
~03DBAE MOVE.L  #AC01B0,10(A0)
~03DBB6 TST.L   D0
~03DBB8 BEQ     0003DBCA
~03DBBA LEA     0001EB53,A1       A1
~03DBC0 MOVE.W  0003D7C0,D1       D1
~03DBC6 MOVE.L  D0,0(A1,D1.W)
~03DBCA LEA     1E(A0),A0
~03DBCE MOVE.L  #35A52,14(A0)
~03DBD6 MOVE.B  #2,A(A0)
~03DBDC MOVE.B  #1,9(A0)
~03DBE2 MOVE.L  #3F00030,10(A0)
~03DBEA MOVE.B  #5,E(A0)
~03DBF0 JSR     00037F38
~03DBF6 JSR     0003849E
~03DBFC LEA     00023D98,A0
~03DC02 MOVEA.L 00023A86,A1
~03DC08 LEA     3DC14(PC),A2
~03DC0C MOVE.W  #2E,D0
~03DC10 BRA     0003E802
;=========================================
```

Le move en ***$3DAC2*** ne semble pas plus suspect que ça.
On voie une boucle en ***$3DB06*** avec une jolie fonction **EOR** avec les registres **D2** et **D0** en ***$3DB10***
Plus loin, en ***$3DBB6*** un test est effectué et selon la condition on effectue ou pas la partie entre ***$3DBBA*** **->** ***$3DBC6***

On va poser plusieurs *BreakPoints* pour voir les valeurs des registres.

Rebooter votre amiga, une fois l'animation de départ à l'écran, entrer dans l'**AR**, **taper**
BS 3DB00 puis **BS 3DB1E** et **BS 3DBB6** sans oublier **BS 3DBCA** et pour finir **X**

Le **level1** se charge et fonctionne normalement, passer en mode *cheatMode* et **appuyer sur la touche la touche F2**
Le **Level2** se charge, **appuyer sur fire** pour le lancer, notre *BreakPoint* 3DB00 est atteint, **taper R**

```
r
D0=1A239842 696E7366 696E7374 0000F000  00001000 00000001 0000FFFF 000149A0
A0=000244A0 0003ECBE 0002C2DC 00026528  0007A18E 00026AE4 0002612A 000002F8
PC = 0003DB00 USP = 0007F000 SR = 2000 T=0 S=1 I=000 X=0 N=0 Z=0 V=0 C=0
```

Taper X

Notre *BreakPoint* 3DB1E est atteint très rapidement, **taper R**

```
r
D0=B18F4BFD 696E7366 00023A8E 0000F000  00001000 00000001 0000FFFF 00000000
A0=000244A0 0002A31E 0002C2DC 00026528  0007A18E 00026AE4 0002612A 000002F8
PC = 0003DB1E USP = 0007F000 SR = 2004 T=0 S=1 I=000 X=0 N=0 Z=1 V=0 C=0
```

On peut voir l'impact sur **D0** et **D1** de la boucle ***$3DB06***, **taper X**

Notre *BreakPoint* 3DBB6 est atteint très rapidement, **taper R** puis **ST** et encore **ST**

```
Breakpoint raised at address: 0003DBB6
r
D0=B18F4BFD 696E7366 00023A8E 0000F000  00001000 00000001 0000FFFF 00000000
A0=00024518 0002A31E 0002C2DC 00026528  0007A18E 00026AE4 0002612A 000002F8
PC = 0003DBB6 USP = 0007F000 SR = 2000 T=0 S=1 I=000 X=0 N=0 Z=0 V=0 C=0
st
D0=B18F4BFD 696E7366 00023A8E 0000F000  00001000 00000001 0000FFFF 00000000
A0=00024518 0002A31E 0002C2DC 00026528  0007A18E 00026AE4 0002612A 000002F8
PC = 0003DBB8 USP = 0007F000 SR = 2008 T=0 S=1 I=000 X=0 N=1 Z=0 V=0 C=0
~03DBB8 BEQ     0003DBCA
st
D0=B18F4BFD 696E7366 00023A8E 0000F000  00001000 00000001 0000FFFF 00000000
A0=00024518 0002A31E 0002C2DC 00026528  0007A18E 00026AE4 0002612A 000002F8
PC = 0003DBBA USP = 0007F000 SR = 2008 T=0 S=1 I=000 X=0 N=1 Z=0 V=0 C=0
~03DBBA LEA     0001EB53,A1
```

La commande `TST.L D0` en ***$3DBB6*** est exécutée et on entre dans notre petite partie spécifique ***$3DBBA*** **->** ***$3DBC6***
Taper X

Notre *BreakPoint* 3DBCA est atteint, **taper R** afin d'afficher l'état des registres en sortie de notre petite routine.
Taper X, le jeu semble figé ou crash.

Insérez la disquette original dans le lecteur de l'amiga, rebooter et **effectuer les mêmes tests**.

```
r
D0=1A239842 696E7366 696E7374 0000F000  00001000 00000001 0000FFFF 000149A0
A0=000244A0 0003ECBE 0002C2DC 00026528  0007A18E 00026AE4 0002612A 000002F8
PC = 0003DB00 USP = 42E2338B SR = 2000 T=0 S=1 I=000 X=0 N=0 Z=0 V=0 C=0
```

1er *BreakPoint* Atteint, RAS au niveau de **D0** et **D2**, exactement comme notre crack.

```
r
D0=04C80000 696E7366 00023A8E 0000F000  00001000 00000001 0000FFFF 00000000
A0=000244A0 0002A31E 0002C2DC 00026528  0007A18E 00026AE4 0002612A 000002F8
PC = 0003DB1E USP = 42E2338B SR = 2004 T=0 S=1 I=000 X=0 N=0 Z=1 V=0 C=0
```

Second *BreakPoint* Atteint, **D0** n'a pas la même valeur que notre crack.

```
Breakpoint raised at address: 0003DBB6
r
D0=04C80000 696E7366 00023A8E 0000F000  00001000 00000001 0000FFFF 00000000
A0=00024518 0002A31E 0002C2DC 00026528  0007A18E 00026AE4 0002612A 000002F8
PC = 0003DBB6 USP = 42E2338B SR = 2000 T=0 S=1 I=000 X=0 N=0 Z=0 V=0 C=0
st
D0=04C80000 696E7366 00023A8E 0000F000  00001000 00000001 0000FFFF 00000000
A0=00024518 0002A31E 0002C2DC 00026528  0007A18E 00026AE4 0002612A 000002F8
PC = 0003DBB8 USP = 42E2338B SR = 2000 T=0 S=1 I=000 X=0 N=0 Z=0 V=0 C=0
~03DBB8 BEQ     0003DBCA
st
D0=04C80000 696E7366 00023A8E 0000F000  00001000 00000001 0000FFFF 00000000
A0=00024518 0002A31E 0002C2DC 00026528  0007A18E 00026AE4 0002612A 000002F8
PC = 0003DBBA USP = 42E2338B SR = 2000 T=0 S=1 I=000 X=0 N=0 Z=0 V=0 C=0
~03DBBA LEA     0001EB53,A1
```

On voie clairement la différence entre les deux. Après la sortie de la boucle ***3DB06 -> 3DB1C*** , **D2** n'a plus du tout la même valeur.
On va empêcher toute modification de **D2** durant cette boucle**,** on va donc supprimer la fonction en ***$3DB06***
Et afin de passer le test plus loin en ***$3DBB6*** , on va changer le **BEQ 3DBCA** en **BRA 3DBCA** à adresse ***$3DBB8***

Ré-inserez notre disquette de crack dans le lecteur de l'amiga et rebooter celui-ci.
Une fois l'animation de départ à l'écran, entrer dans l'**AR**, **taper :**

```
A 3DB06
^03DB06 NOP <RETURN>
^03DB08 <RETURN>

A 3DBB8
^03DBB8 BRA 3DBCA <RETURN>
^03DB1E <RETURN>
```

Sans oublier de retourner au code, taper : **X**

```
a 3DB06
^03DB06 NOP
^03DB08

a 3DBB8
^03DBB8 BRA 3DBCA
^03DBBA
```

Le **level1** se charge et fonctionne normalement, passer en mode *cheatMode* et **appuyer sur la touche la touche F2**
Le **Level2** se charge, **appuyer sur fire** pour le lancer.
BINGO, ça fonctionne, nous étions donc bien en présence d'une protection.

Part 18 Seconde protection - Patch Level1 bug graphique sur TIMER<1 minute

On est passé un peut vite sur le **Level1**, **revenons sur celui-ci et jouons un petit peu**. (**passer par un reboot de l'amiga**)

Quand le temps arrive environ vers 1minute, des problèmes apparaissent sur le sprite du personnage principal.
Et si on persiste, ces problèmes graphiques deviennent de plus en plus importants.

Si on pouvait avoir des doutes dans l'étape précédente entre protection ou Bug , ici on peut clairement dire que c'est encore une protection.
Je vous invite à tester les autres niveaux à l'aide du ***CheatMode***, vous pourrez voir que ce problème n'apparait pas.

Nous avons vue en **PART 16** qu'une routine était initialisée pour sauter dans le bon niveau, plusieurs opérations étaient effectuées avec en grande majorité une utilisation de **D0** pour calculer l'adresse du saut et qu'elle utilisait le numéro du niveau en cours **($239C8)**

Regardons donc si il existe d'autre routine similaire en mémoire**.**
Cela devrais ressembler à ce que l'on a déjà vue, à savoir :

```
MOVE.W    $239C8,D0          ; 3039000239C8
ADD.W     D0,D0              ; D0D40
ADD.W     D0,D0
```

Rebooter votre Amiga, lancer le jeu normalement au **LEVEL1**, **entrer** normalement dans l'**AR**

Taper : `F 30 39 00 02 39 C8 D0 40`

```
f 30 39 00 02 39 C8 D0 40
Search from: 000000 to: C80000
02A292  02BC56  02EB48  03251E  0369C4  03C9DA
Ready.
```

On va regarder un par un les adresses en question, **taper :** `D 2A292`

```
d 2A292
~02A292 MOVE.W  000239C8,D0
~02A298 ADD.W   D0,D0
~02A29A ADD.W   D0,D0
~02A29C MOVEA.L 30(PC,D0.W),A0
~02A2A0 MOVE.L  #50858,(A0)
~02A2A6 CLR.B   0007FFFF
~02A2AC MOVE.W  000239C8,D0
~02A2B2 LEA     0002B72E,A1
~02A2B8 LEA     00029F6A,A2
~02A2BE JSR     0000D300
~02A2C4 MOVE.B  0007FFFF,000239D9
~02A2CE BRA     0002A2EE
;=====================================
~02A2D2 ORI.B   #E6,D2
~02A2D6 ORI.B   #F2,D2
~02A2DA ORI.B   #FE,D2
~02A2DE ORI.B   #A,D2
~02A2E2 ORI.B   #16,D2
```

Pas de saut ici 'calculé', suivant.

Taper : `D 3251E`

```
d 3251E
~03251E MOVE.W  000239C8,D0
~032524 ADD.W   D0,D0
~032526 MOVE.B  2E(PC,D0.W),E(A1)
~03252C MOVE.B  29(PC,D0.W),8(A1)
~032532 RTS
;=====================================
~032534 MOVE.B  D0,8(A0)
~032538 ADDQ.B  #1,A(A0)
~03253C RTS
;=====================================
```

Pareille, pas de saut ici 'calculé.

Taper : `D 3C9DA`

```
d 3C9DA
~03C9DA MOVE.W  000239C8,D0
~03C9E0 ADD.W   D0,D0
~03C9E2 ADD.W   D0,D0
~03C9E4 MOVEA.L 6(PC,D0.W),A0
~03C9E8 MOVEA.L 0(A0,D2.W),A0
~03C9EC JSR     (A0)
~03C9EE BRA     0003CA0C
;=====================================
~03C9F0 ORI.B   #A6,D3
~03C9F4 ORI.B   #D6,D3
```

Toujours pas bon.

Taper : `D 369C4`

```
d 369C4
~0369C4 MOVE.W  000239C8,D0
~0369CA ADD.W   D0,D0
~0369CC ADD.W   D0,D0
~0369CE JMP     0(PC,D0.W)
;=====================================
~0369D2 RTS
;=====================================
~0369D4 BRA     000369F0
;=====================================
~0369D8 BRA     00036B3C
;=====================================
~0369DC BRA     00036BA0
```

Ici on a bien un saut 'calculé' avec **D0**, **PC** mais, comme on cherche un problème uniquement sur le **Level1**, c'est sur le **RTS** que l'on va tomber donc, Nop, celui-là n'est pas non plus celui que l'on cherche

Taper : `D 2BC56`

```
d 2BC56
~02BC56 MOVE.W  000239C8,D0
~02BC5C ADD.W   D0,D0
~02BC5E ADD.W   D0,D0
~02BC60 JMP     -2(PC,D0.W)
;=====================================
~02BC64 BRA     0002BC80
;=====================================
~02BC68 BRA     0002BD7C
;=====================================
~02BC6C BRA     0002BE34
```

Ici on a quelque chose d'intéressant. Nous allons modifier ce saut avec juste un **RTS**.
Taper : `A 2BC64`
`^02BC64 RTS <RETURN>`
`^02BD66 <RETURN>`

Sans oublier de retourner au code, taper : `X`

Oups…Les ennemies deviennent invulnérable à nos tirs, pas bon du tout ça.

Bon, il nous reste plus que ***$2EB48*** à tester.
N'oublier pas d'annuler notre **RTS**, de remettre le **BRA 2BC80** en ***$2BC64***
Taper : `D 2EB48`

```
d 2EB48
^02EB48 MOVE.W  000239C8,D0
^02EB4E ADD.W   D0,D0
^02EB50 ADD.W   D0,D0
^02EB52 JMP     -2(PC,D0.W)
;=====================================
^02EB56 BRA     0002EB72
;=====================================
^02EB5A BRA     0002EFF0
;=====================================
^02EB5E BRA     0002F372
```

Ici aussi on a quelque chose d'intéressant. Modifier donc comme précédemment le saut en **$2EB56**
Taper : `A 2EB56`
`^02EB56 RTS <RETURN>`
`^02EB58 <RETURN>`

Sans oublier de retourner au code, taper : `X`
Chose assez drôle, plus aucun ennemies n'apparait à l'écran. Attendez la fin du **Timer** afin de voir les bugs graphiques persistent.
Plus de Bug graphique ☺

Nous somme sur la bonne piste.

Allons jeter un coup d'œil sur la routine en ***$2EB72***
Taper : **D 2EB72**

Après un nombre impressionnant de **LEA** et **BSR**, on arrive sur un premier test :

```
~02EF14 TST.L   00023A82
~02EF1A BNE     0002EF3C
~02EF1C MOVE.L  00023C56,D0
~02EF22 SUB.L   00023AAE,D0
~02EF28 BEQ     0002EF3C
~02EF2A LEA     0003EB4E,A0
~02EF30 ADD.L   D0,(A0)
~02EF32 LEA     1E(A0),A0
~02EF36 MOVE.L  A0,0002EF2C
~02EF3C LEA     00024572,A0
~02EF42 LEA     00029DEA,A2
```

```
02EF14  TST.L   23A82          ; un test sur 23A82, adresse qui contient quoi ?
02EF1A  BNE     2EF3C          ; On branche si condition en 2EF3C, donc on bypass le Move et le Sub
02EF1C  MOVE.L  23C56,D0       ; 23C56 ?
02EF22  SUB.L   23AAE,D0       ; 23AAE ?
02EF28  BEQ     2EF3C          ; Encore un branchement en 2EF3C si condition.
02EF2A  LEA     3EB4E,A0       ; 3EB4E
02EF30  ADD.L   D0,(A0)        ; Remarquer que l'on travaille beaucoup avec A0 et D0
02EF32  LEA     1E(A0),A0      ;
02EF36  MOVE.L  A0,2EF2C       ; 2EF2C ?
02EF3C  LEA     24572,A0       ; 24572 ?
02EF42  LEA     29DEA,A2       ; 29DEA ?
```

Regardons ça de plus près, dans l'ordre des choses, tout d'abord l'adresse du **TST.**
Taper : **FA 23A82**

```
02ACE6  TST.L   00023A82       ; Encore un test sur cette même adresse, bof…
02AFC0  MOVE.L  23A82,D2       ; On définit D2 avec ce que contient en LongWord l'adresse $23A82, à voir…
02C8B2  MOVE.L  D2,23A82       ; Ah intéressant, on va modifier le contenue de $23A82 par rapport à D2
02EF14  TST.L   23A82          ; Ici c'est notre adresse vue au-dessus.
```

Regardons ça de plus près, **taper :** **F 02C8B2**

```
f 02 C8 B2
Search from: 000000 to: C80000
Ready.
```

Rien du tout, essayons autrement.

Taper : **FA 2C8B2**

```
fa 2C8B2
Search from: 000000 to: C80000
02AFD4 BRA      0002C8B2
02C842 BSR      0002C8B2
Searched up to adr: C80000
Ready.
```

Déjà mieux ; on va regarder tout ça :

Taper : **D 2AFD4-8**

```
d 2AFD4-10
~02AFC4 MOVE.W  D2,(A5)
~02AFC6 MOVE.L  000239A0,D0
~02AFCC BSR     0002D05A
~02AFD0 BSR     0002A828
~02AFD4 BRA     0002C8B2
!=========================================
_02AFD8 MOVEA.W 10(A0),A1
```

Une suite de saut, bof…
On continue.

Taper : `D 2C842-10`

```
d 2C842-10
~02C832 BSET    #0,1C(A0)
~02C838 CLR.B   A(A0)
~02C83C MOVE.W  #FFFF,0002616E
~02C844 MOVE.W  #A,0002616C
~02C84C RTS
;=====================================
```

Tiens donc… ou t'es qu'il est notre **2C842** ???
Taper : `D 2C842`

```
d 2C842
~02C842 BSR     0002C8B2
~02C844 MOVE.W  #A,0002616C
~02C84C RTS
;=====================================
_02C84E LEA     1E(A1),A1
```

Il est bien là mais caché.
Hazard des choses ou volontairement caché du code principal ?!
Vue que l'on est en pleine recherche de protection, on pencherait plus pour un code caché.

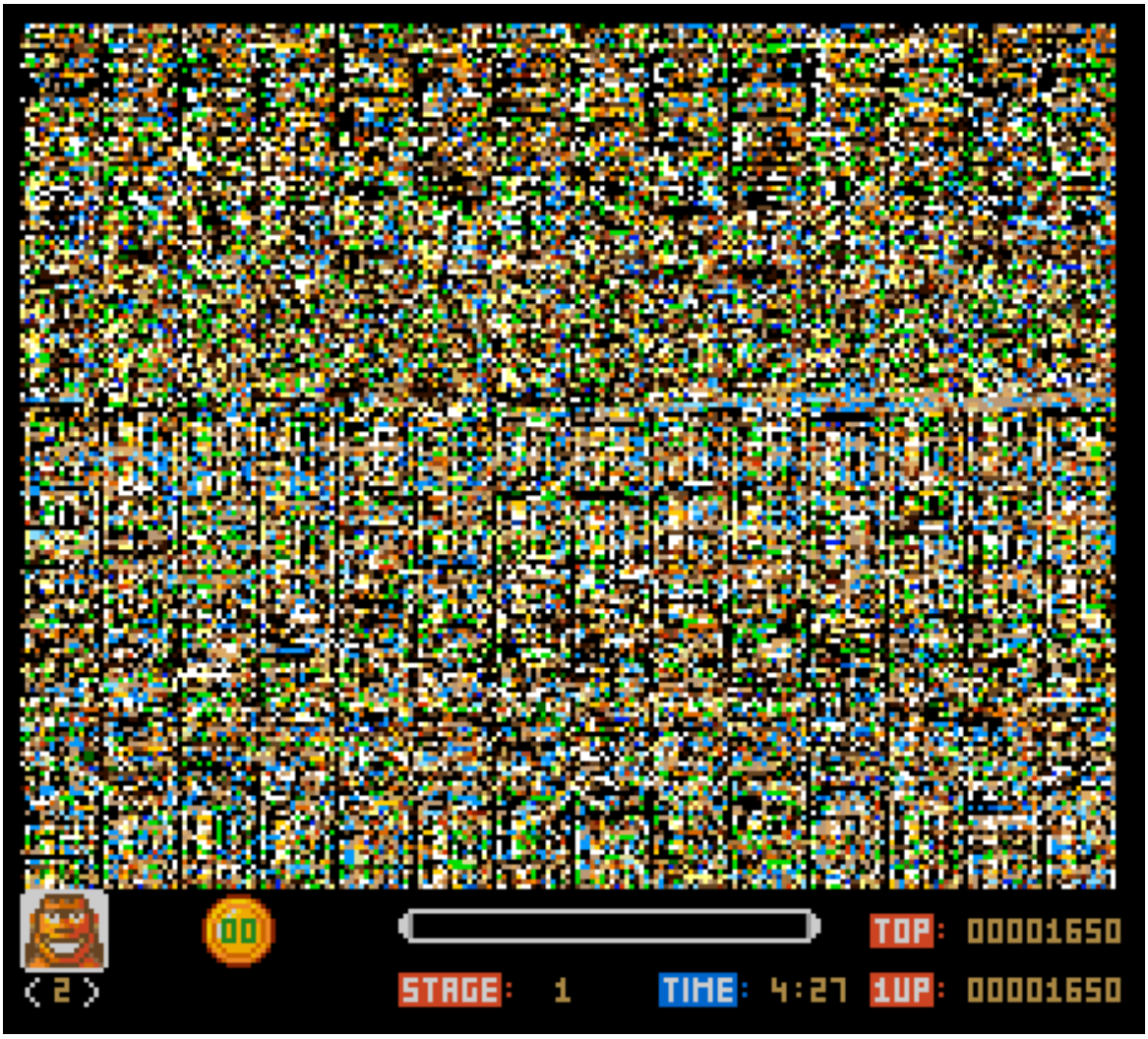
On va supprimer ces tests et effectuer ces sauts en ***$2EF1A*** *et* ***$2EF28***
Taper : `A 2EF1A`
`^02EF1A BRA 2EF3C` `<RETURN>`
`^02EF1C` `<RETURN>`

Taper : `A 2EF28`
`^02EF28 BRA 2EF3C` `<RETURN>`
`^02EF2A` `<RETURN>`

Sans oublier de retourner au code, taper : `X`
Bon déjà, bon signe, les modifications que l'on vient de faire n'ont pas apporter de bug.
Attendez que le compteur arrive en dessous des 1 minutes et…
BINGO… plus de problème graphique sur les sprite and co.
Pas besoin d'investiguer plus loin sur cette seconde protection du coup.

Part 19 Troisième protection - Patch général bug graphique au bout de 8 minutes

Comme on dit, jamais deux sans trois et bein, il semblerait hélas que ce soit le cas.
Après environ 8 minutes de jeux, un bon gros bug graphique général apparait dans le jeu.



Entrer dans l'AR, taper : `R` **puis** `M 2F0`

```
r
D0=FFFF3345 00000002 000014A0 00002C0F  00000000 00000000 000000E0 0000000B
A0=00000108 B9000847 000211F0 D0AAB184  00036FE2 00C0038A 00023D3E 000002F0
PC = 00FC30D6 USP = 0007F000 SR = 270C T=0 S=1 I=111 X=0 N=1 Z=1 V=0 C=0

m 2f0
:0002F0 00 00 00 0B 20 08 00 06 00 26 20 08 00 02 A7 C6 .... ....& ...§.
```

Bon, **D0** semble encore hors limite…

Dans la **Part17** nous avons utilisé une recherche sur un type spécifique de code que l'on avait vue en **part16**
A savoir :

```
MOVE.W    $239C8,D0          ; 3039000239C8
ADD.W     D0,D0              ; D0D40
ADD.W     D0,D0
```

Il est assez courant que la même technique de protection soit utilisée plusieurs fois dans un jeu mais il est aussi assez courant que d'une protection à l'autre (sur le même jeu), le code diffère légèrement, subtilement je dirais.
On va donc chercher la sequence suivante :

```
ADD.W     D0,D0              ; D0D40
ADD.W     D0,D0              ; D0D40
```

Lancer le jeu sur le **Level1** puis **entrer** dans **l'AR, taper** : `F D0 40 D0 40`

Cela nous donne, assez logiquement, une jolie liste.

```
F D0 40 D0 40
Search from: 000000 to: C80000
00DD14  02A298  02BC5C  02C208  02CBD4  02CBE2  02EB4E  0323A0  034FE0  035160
035162  035C9E  0369CA  038548  038870  038E38  0395C8  03B7F2  03BC42  03C378
03C9E0  03CA38  03CA82
Ready.
```

$DD14 est hors limite de la plage mémoire du jeu mais bon, on va quand même tester.
On va commencer par les quatre premier.

Taper : `BS DD14` puis `BS 2A298` et `BS 2BC5C` sans oublier `BS 2C208`
Sans oublier de retourner au code, **taper** : `X`

```
bi
Breakpoints at addresses:  00DD14  02A298  02BC5C  02C208
Ready.
x
```

Notre *BeakPoint $2BC5C* est atteint rapidement, **taper** : `D 2BC5C-10`

```
d 2BC5C-10
~02BC4C ST      -4231(A5)
~02BC50 BRA     0002B35C
;=====================================
~02BC54 MOVEQ   #0,D0
~02BC56 MOVE.W  000239C8,D0
~02BC5C ADD.W   D0,D0
~02BC5E ADD.W   D0,D0
~02BC60 JMP     -2(PC,D0.W)
;=====================================
~02BC64 BRA     0002BC80
;=====================================
~02BC68 BRA     0002BD7C
;=====================================
~02BC6C BRA     0002BE34
```

Intéressant, encore un saut calculé avec **D0** et le **Level**, mais on a déjà vue cette adresse d'ailleurs en **PART17**
De plus le problème que l'on recherche**,** bug graphique/plantage n'apparait pas spécifique dans un niveau.
Il doit forcément avoir un compteur quelque part.

On enlève notre *BreakPoint* et on retourne au code, **taper** : `BD 2BC5C` puis `X`

Notre *BeakPoint $2C208* est atteint, **taper** : `D 2C208-10`
Cela semble aussi assez intéressant, remonter un peu plus pour trouver le début de cette routine.

```
~02C1F2 BEQ     0002C258
~02C1F4 LEA     0002612A,A6
~02C1FA MOVE.W  -616E(A2),D1
~02C1FE BEQ     0002C236
~02C200 CLR.W   -616E(A2)
~02C204 ADD.W   D0,D0
~02C206 MOVE.W  D0,D1
~02C208 ADD.W   D0,D0
~02C20A ADD.W   D0,D0
~02C20C ADD.W   D1,D0
~02C20E EXT.L   D0
~02C210 LEA     00026AE4,A0
~02C216 ADDA.L  D0,A0
~02C218 MOVE.W  (A0)+,D0
~02C21A MOVE.B  D0,3A(A6)
~02C21E MOVE.L  (A0)+,1A(A6)
~02C222 MOVE.L  (A0),1E(A6)
~02C226 MOVE.L  #7000000,22(A6)
~02C22E MOVE.W  #8,00DFF096
~02C236 MOVE.B  22(A6),D7
~02C23A BTST    #2,D7
~02C23E BNE     0002C246
~02C240 CLR.W   -6170(A2)
~02C244 BRA     0002C258
;=====================================
```

Pas mal de chose très intéressante mais pas de **SUB.** On peut voir aussi une forte utilisation de **D0**
Et une un branchement en **2C258** en fin de routine.

Voyons voir s'il existe d'autre appel a ***$2C258*** en mémoire.
Taper : `FA 2C258`

```
fa 2C258
Search from: 000000 to: C80000
02C1F2 BEQ      0002C258
02C244 BRA      0002C258
Searched up to adr: C80000
Ready.
```

Il existe bien un autre appel mais c'est un branchement avec condition et notre protection **CRC** devrait logiquement fonctionner avec un compteur qu'elle incrémente tout le temps donc pas sur une condition.

Quoi qu'il en soit, on va regarder en ***$2C258***
Taper : `D 2C258`

```
d 2C258-10
~02C248 ORI.B   #76,D2
~02C24C BSR     0002C366
~02C250 MOVE.B  D7,22(A6)
~02C254 ORI.W   #8,D5
~02C258 LEA     -5D44(A2),A6
~02C25C TST.L   (A6)
~02C25E BEQ     0002C26C
~02C260 SUBQ.L  #1,(A6)+
~02C262 MOVEA.L (A6),A1
~02C264 MOVE.L  (A1)+,D0
~02C266 MOVE.L  A1,(A6)+
~02C268 SUB.L   D0,(A6)
~02C26A BRA     0002C274
;=========================================
```

Ah, là on a tout ce qu'il nous faut. Le **TST**, Le **BEQ** juste après, sans oublier le **SUB** et les **MOVE** qui mettent à jour une adresse.
On va supprimer tous nos *BreakPoints* et en poser 1 seul à l'endroit du **TST**

Taper : `BDA` puis `BS 2C25C`

```
Breakpoint raised at address: 0002C25C
r
D0=00000001 00000080 00154000 000000FF  0000000A 0000800F 00000018 0000FF04
A0=00026136 00026176 0002C2DC 00026824  00036FE2 00DFF0D8 00026598 000002B6
PC = 0002C25C USP = 0007F000 SR = 2308 T=0 S=1 I=011 X=0 N=1 Z=0 V=0 C=0
```

```
TST.L (A6)                ; Ou A6 = $26598
```

Qui y'a-t-il à cette adresse ? **Taper** : `M 26598`

```
m 26598
:026598 00 00 52 36 00 02 A4 CE 2B 7C C9 97 C0 C0 D0 D8 ..R6....+|......
```

Ok.. !
Supprimons nos *BreakPoints* et retournons au jeu.
Taper : `BDA` puis `X`

Attendez 1 seconde, **entrer dans l'AR**
Taper : `M 26598`

```
m 26598
:026598 00 00 52 1C 00 02 A5 36 95 1F 06 C0 C0 C0 D0 D8 ..R....6........
```

La valeur à changer, elle est plus petit que précédemment.
Essayer plusieurs fois et vous verrez qu'elle ne fait que descendre, d'où d'ailleurs le **TST** en **$2C25C**

Et une fois à zéro, il se passe quoi ?
Taper : `M 26598` on va réduire cette valeur, modifier la par exemple par **FF**

```
m 26598
:026598 00 00 00 FF 00 02 A1 96 33 1A 5C D4 C0 C0 D0 D8 ..$.....3.\.....
:0265A8 E0 E8 F0 F8 00 F8 F0 E8 E0 D8 D0 C8 3F 37 2F 27 .............?7/'
```

Retourner au code, **taper** : `X`

Bingo en 5 secondes, le jeu bug graphiquement et crash

Il semblerait bien que notre compteur qu'utilise la routine de protection que l'on cherche soit en **$26598**
Et que l'on soit bien tombé sur le code qui décrémente cette valeur.

Si on reprend le code en question, nous avons plusieurs solutions pour parer à ce problème.
La plus rapide étant de changer le branchement conditionnel **BEQ** après le **TST** du compteur.

```
d 2C258-10
~02C248 ORI.B   #76,D2
~02C24C BSR     0002C366
~02C250 MOVE.B  D7,22(A6)
~02C254 ORI.W   #8,D5
~02C258 LEA     -5D44(A2),A6
~02C25C TST.L   (A6)
~02C25E BEQ     0002C26C
~02C260 SUBQ.L  #1,(A6)+
~02C262 MOVEA.L (A6),A1
~02C264 MOVE.L  (A1)+,D0
~02C266 MOVE.L  A1,(A6)+
~02C268 SUB.L   D0,(A6)
~02C26A BRA     0002C274
;=====================================
```

Rebooter votre Amiga pour arriver sur le **Level1** puis **entrer** dans **l'AR.**

Bien sûr, n'oublions pas le patch trouvé en 17, sinon, le jeu partira en bug graphique avant les 8 minutes

Taper : `A 2EF1A`
`^02EF1A BRA 2EF3C` <RETURN> ; Patch bug graphique quand TIME <1 minute
`^02EF1C` <RETURN>

Taper : `A 2EF28`
`^02EF28 BRA 2EF3C` <RETURN> ; Patch bug graphique quand TIME <1 minute
`^02EF2A` <RETURN>

Taper : `A 2C25E` ; Patch corruption graphique quand compteur > 8 minutes
`^02EF1A BRA 2C274` <RETURN>
`^02EF1C` <RETURN>

On saute ainsi la partie qui décrémente notre compteur.
Retourner au code, **taper** : `X`

Laisser le jeu tourner pendant plus de 8 minutes en restant par exemple sur place et en perdant quelques vie, peu importe.
BINGO… plus de corruption graphique.

Part 20 Modification du code tokistarter et de notre disk hacké

Maintenant il nous faut modifier notre code de hack principal et ajouter ces patchs.
Lancer **ASMone** sur votre Amiga, réserver un peu de mémoire chip, 100ko.
Charger le fichier préalablement crée et sauvé sous le nom de **tokistarter.asm** présent sur la disquette **TOKI_FILE.**
Insérer nos patchs juste avant la routine de copy du ***trackloader***

… …
… … …

```
dbf     d7,.loop                ;
        <ICI>
; do a copy of trackloader at safe place
```

… … …
… …

A savoir :

```
        ; skip checksum at level 2
        lea     $3db06,a0
        move.w  #$4e71,(a0)     ; nop
        lea     $3dbb8,a0
        move.b  #$60,(a0)       ; bra

        ; checksum 2 (in level 1 routine)
        lea     $2ef1a,a0       ;
        move.b  #$60,(a0)       ; bra
        lea     $2ef28,a0       ;
        move.b  #$60,(a0)       ; bra

        lea     $2c25e,a0
        move.w  #$6014,(a0)     ; bra
```

Comme dans la **PART18** de ce tuto, assembler et sauver le tout sous le nom de **tokistarter.exe** toujours sur notre disk **TOKI_FILE**
Il nous reste plus qu'à récrire celui-ci sur notre disk cracké**.**
Lancer Mega-Mon, charger tokistarter.exe en mémoire, lancer la copie de celui-ci sur notre disque cracké.

Taper :

```
L TOKI_FILE:tokistarter.exe 10000
>b 10000 2 4
```

Bien sûr, n'oublier pas de changer le disk TOKI_FILE avec notre disk cracké avant de lancer la commande d'écriture.
Ou comme dans la **PART15** utiliser un lecteur externe.

Rebooter ensuite votre Amiga avec notre disquette cracké dans celui-ci et apprécier le jeu ☺