Tutoriel	**AmigaCracking : RodLand**
Protection	**COPYLOCK(s) + CRC**
Auteur Original	**Gi@nts**
Source original	**http://flashtro.com**
Version	**28/10/2019 Gi@nts**
Vérification/Correction	**V1.0, Testé et fonctionnel de A à Z**

* RODLAND - CRACK TUTORIEL *

Table des matières

Matériels nécessaires 3
Général Info 3
Agencement des disquettes Amiga 5
Le Format MFM 8
Les registres CIA-A et CIA-B 10
WinUAE 12
Part 1 X-copy 13
Part 2 Analyse de l'image IPF 14
Part 3 Cheats 15
Part 4 Comportement des chargements du jeu et test de notre Backup 16
Part 5 Analyse et modification du bootblock 17
Part 6 Let's Trace 19
Part 7 Copylock ou pas finalement ? 23
Part 8 Bypasser en Live le Copylock 24
Part 9 Organigramme des diverses étapes 25
Part 10 Crack Copylock 'Test' 26
Part 11 Protection(s) CRC 27
Part 12 tentative de crack 30
Part 13 Test de notre crack 31
Part 14 Mise en place de notre crack 32
Part 15 Let's Crack it 34

Matériels nécessaires

1) Un AMIGA ou l'émulateur WINUAE.
2) Une Carte ACTION REPLAY (ou ça ROM Image) selon configuration utilisé.
3) Le jeu Original en disquette ou son image CAPS (SPS 1509)

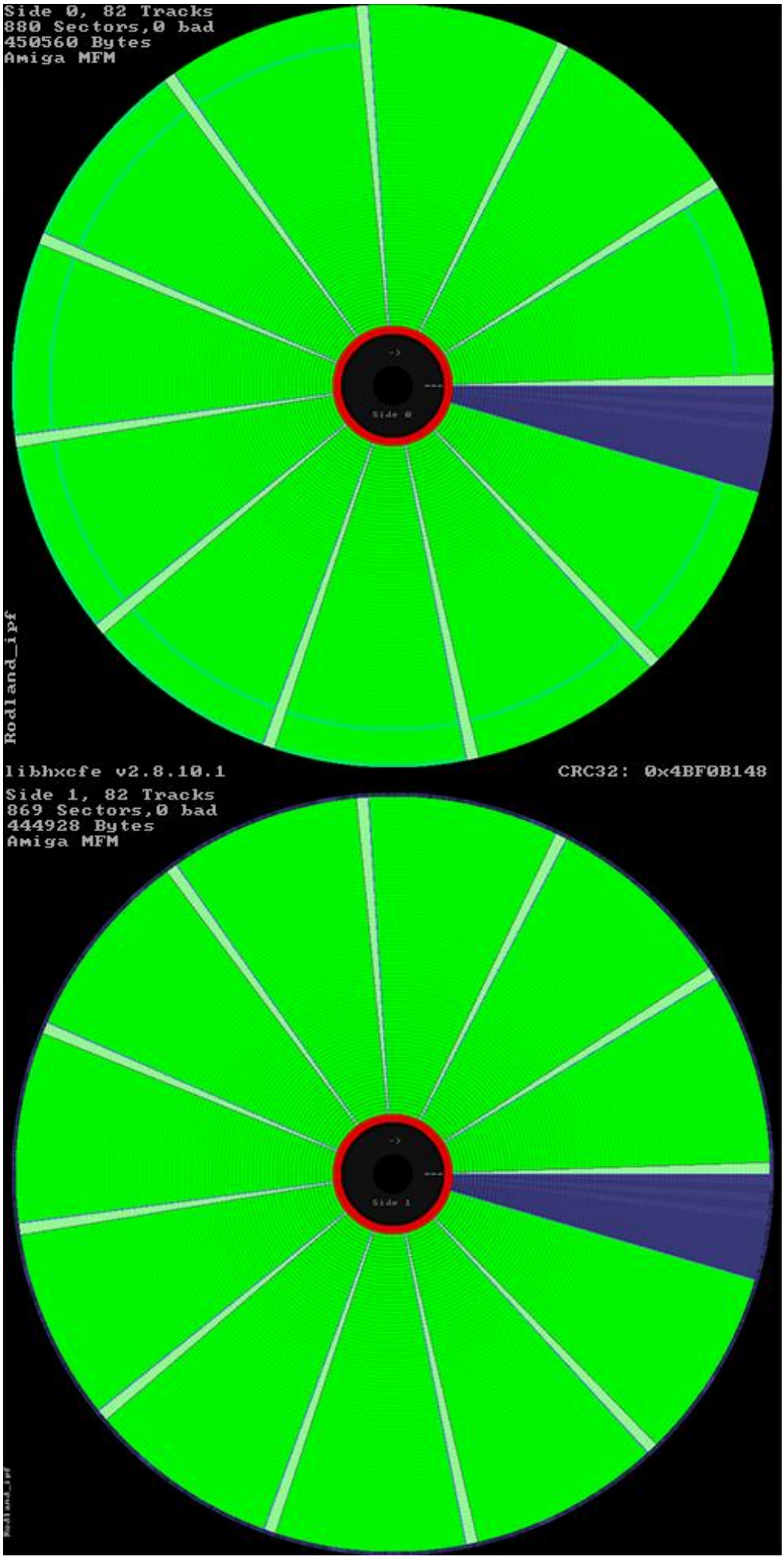
Général Info

On va essayer à travers ce tuto d'être le plus complet possible.
Ce tuto est basé sur mon travail d'analyse ainsi que le tuto original de Galahad / Scoopex.
Il est à mon sens assez complet. Il est fortement conseillé de le suivre sinon vous risquer d'être perdu dans sa compréhension.
A noter que c'est un tutorial de crack et non pas un document sur 'comment apprendre l'assembleur sur Amiga'.
Le but est de comprendre les instructions sans pour autant toutes les détaillées, de pouvoir comprendre le fonctionnement global du code afin de pouvoir isoler les parties en rapport avec le Hack (Trackloader, routine de CheckSum, CopyLock…)

Bon Tuto.

Gi@nts

Disk 1

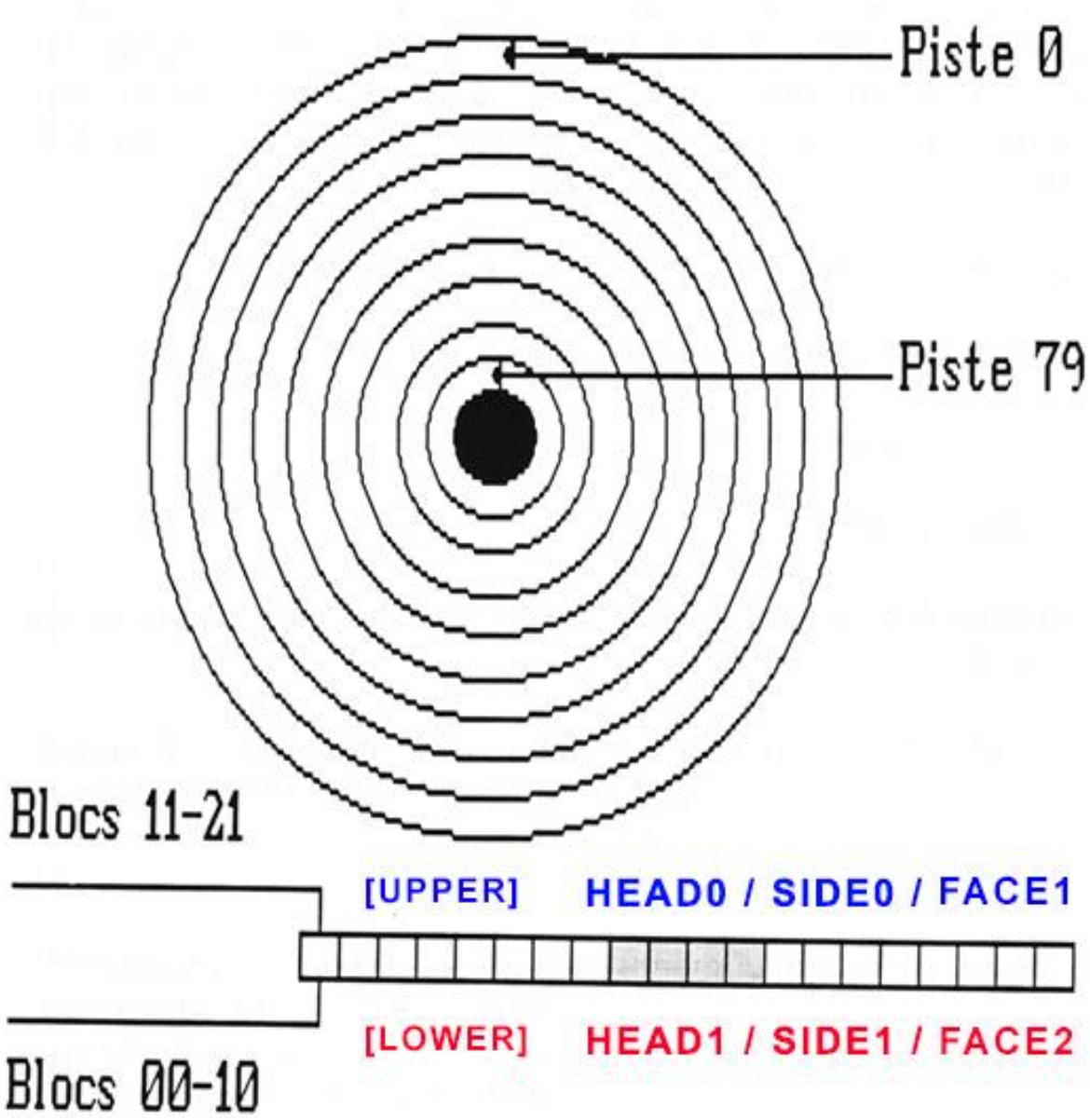

Side 0, 82 Tracks
880 Sectors,0 bad
450560 Bytes
Amiga MFM
Rodland_ipf
libhxcfe v2.8.10.1
CRC32: 0x4BF0B148
Side 1, 82 Tracks
869 Sectors,0 bad
444928 Bytes
Amiga MFM

Agencement des disquettes Amiga

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 sûr) 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 piste (Track donc) à copier 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.

Le Format MFM

Les DATA sur les *tracks* d'une disquette AmigaDos sont codées/décodées au format **MFM**.
Chaque *tracks* contient 11 secteurs de 512 octets chacun.
Chaque *secteur* à un *header* qui nous donnes le numéro de track, le numéro de secteur et d'autres données.

Le contenu d'un disque normal AmigaDos est le suivant :

...	GAP	...	TRACK	...	GAP	...

+ Gap A normalement une valeur de **00** octet soit $AAAA au format MFM.
+ Track Contient normalement 11 secteurs qui sont :

- S E C T O R -

00	00	Sync	Sync	SectorHeader	Fill	Headerchecksum	Datachecksum	Data

00

2 octets
00 Octet soit **$AAAA** au format **MFM**

Sync

2 octets
(**A1**) converti au format **MFM** et 'Clock pulse' n'est pas pris en compte
Donc le résultat nous donnes : **$4489** qui est le 'SyncWord'.
Aucune DATA ne sera jamais convertie dans ce pattern. (en **MFM**)

SectorHeader

1 longWord **[8 octets MFM]**
Header du secteur

Format	Track	Sector	Length
Amiga 1.0 format : $FF **1 octet**	Numéro de track **1 octet**	Numéro de secteur **1 octet**	Nombre de secteur avant Gap de fin **1 octet**

Fill

16 octets **[32 octets MFM]**
Zone destinée pour l'AmigaDos OS 'Recovery' mais jamais utilisée.
donc zone remplie de Zero.

HeaderChecksum

1 longWord **[8 octets MFM]**
Checksum de la zone Header.
Il est calculé en utilisant un XOR et ne contient que des 'databits'

DataChecksum

1 longWord **[8 octets MFM]**
Checksum de la zone Data.
Il est calculé en utilisant un XOR et ne contient que des 'databits'

Data

512 octets **[1024 octets MFM]**
Block de DATA

Noter qu'il n'y a pas de **GAP** entre chaque secteur.

La conversion en **MFM** est faite selon le principe suivant :
Prenez 2 bits de Data et ajouter 1 de Clock entre les deux.
Le bit de Clock est à **1** si les 2 bits de Data sont à 0 sinon, le bit de Clock est à **0**
Ce système permet de ne pas avoir de longue série de 0 ou de 1 qui se suivent.

Un Exemple :

```
Bit de Data  :   0 0 0 1 1 0 1 1 ...
Bit de Clock :  ? 1 1 0 0 0 0 0 0...
ENCODAGE MFM :  ?0101001010001010...
```

Chaque octet est converti en word.
Comme l'extension d'octets est assez compliqué à réaliser sur Amiga, les données sont d'abords divisées en deux moitiés.
Les impaires et les paires.
Les premiers bits à être convertis sont les bits pairs et ensuite les bits impaires.

```
Binaire       : 01001110
Bits pairs    : 0 0 1 1
bits impaires :  1 0 1 0
```

Cela permet un traitement des données plus rapides car l'extraction des moitiés paires et impaires peuvent être réaliser facilement via des opérations logiques '**AND**'

```
Moitié pairs   : DATA 'AND' 0xAAAA
Moitié impairs : DATA 'AND' 0x5555
```

Pour ripper les Data d'un disque vous devez en premier trouver le 'SyncWord' qui est l'endroit ou démarre-les **Data** sur le **Track.**
Le 'SyncWord' est en fait un marqueur.

Les registres CIA-A et CIA-B

Il existe deux registres, CIA-A et CIA-B que vous devez comprendre et apprendre par cœur.
Une fois que vous savez comment fonctionne **$BFD100** et **$BFD001**, vous serez capable de décoder la majorité des loaders.

La plupart des *TrackLoader* n'ont pas de compteur de track pour savoir qu'elle track la tête de lecture à terminer de lire ou écrire.

$BFE001 PRA *Peripheral Data Register for Data Port A :: Status: Read/Write. Chip:* ***CIA-A***

Bit 0: OVL: Bit de 'Memory Overlay' , toujours à 0. Ne change pas.

Bit 1: LED: Bit de 'Power Led/cutoff filter'

- **Valeur1** Led Lecteur diminué et 'cutoff filter' inactivé
- **Valeur0** Led lecteur pleine puissance et 'cutoff filter' activé

Bit 2: CHNG: Changement de disque (1 = aucun changement effectuée, 0 = changement effectué)

Bit 3: WPRO: Disque protégé en écriture (1 = pas protégé ; 0 = protégé)

Bit 4: TK0: Disque track Zero (1 = pas sur track ;0 = positionné sur track 0)

Bit 5: **RDY**: Disque prêt (1 = pas prêt; 0 = prêt)

Bit 6: FIR0: Bouton Fire port1 (1 = pas appuyé; 0 = appuyé)

Bit 7: DIR1: Bouton Fire port2 (1 = pas appuyé; 0 = appuyé)

$BFD100 PRB *Peripheral Data Register for Data Port B :: Status: Read/Write. Chip:* ***CIA-B***

Bit 0: STEP: Déplace la tête du lecteur d'une track dans une direction.
DIR bit (mis à 1, puis 0, et encore à 1 pour déplacer la tête)

Bit 1: DIR: Direction to move drive head (1 =vers l'extérieur, 0 =vers l'intérieur)

Bit 2: SIDE: Sélection de la tête du lecteur (1 = bas ; 0 = haut)

Bit 3: SEL0: Sélection DF0: (1 = pas sélectionné; 0 = sélectionné)

Bit 4: SEL1: Sélection DF1: (1 = pas sélectionné; 0 = sélectionné)

Bit 5: **SEL2**: Sélection DF2: (1 = pas sélectionné; 0 = sélectionné)

Bit 6: SEL3: Sélection DF3: (1 = pas sélectionné; 0 = sélectionné)

Bit 7: MTR: Motor on-off status (1 = motor off; 0 = motor on)

Bit 0: Ce bit contrôle le déplacement de la tête de tous les lecteur sélectionnés.
Pour déplacer la tête, vous devez basculer la valeur de ce bit de 1 à 0 puis, de revenir à 1.
Cette opération déplace la tête d'une distance d'une **Track**
Avant de déplacer la tête du lecteur, vous devez sélectionner une direction '**Bit1**'

Après le déplacement de la tête de lecture il est important d'attendre 3ms avant de faire une nouvelle action sur le lecteur de disquette. Comme les boucles de synchro logiciel ne sont pas précise (dépends de la vitesse d'horloge de l'ordinateur qui varie d'un ordinateur à l'autre), il est recommandé d'utiliser un des timers des chipset **CIA** pour attendre les 3ms nécessaire et il est de 18 ms entre un changement de direction.

Bit 1: La valeur de ce bit détermine la direction tête sur les lecteurs de disquette sélectionné.

- **Valeur1** Vers l'extérieur en direction de la piste 0
- **Valeur0** Vers l'intérieur en direction de la piste 79

Ce **Signal** doit être mis avant l'impulsion **STEP**, de plus pour être sûr de la direction de déplacement effectué, positionner d'abord ce bit à 0 et n'essayer jamais de déplacer une tête de lecteur plus loin que la piste 79 ou avant la piste 0. Vous pouvez vérifier si la tête du lecteur sélectionné est sur la **Track** 0 en lisant le bit 4 du **CIA-A** située à l'adresse : **$BFE001**

Bit 2: Les lecteurs de disquettes amiga sont double faces. Cela veut dire que les lecteurs doivent avoir 2 têtes.
Une tête de lecture/écriture pour la face du haut, et une autre tête pour la face du bas.
Ce bit détermine quelle tête du lecteur doit être utilisé quand une opération de lecture ou d'écriture est demandée.

- **Valeur1** Sélection de la tête du bas
- **Valeur0** Sélection de la tête du haut

La valeur de ce bit affecte seulement le(s) lecteur(s) sélectionné(s).
SIDE doit être maintenant pendant 100 microSecondes avant une opération d'écriture et pendant 1,3 MilliSeconde entre un changement de face.

Bit 3-6: Ces 4 bits permettent de sélectionner quel lecteur(s) de disquette est utilisé(s).
Seulement les lecteurs sélectionnés sont affectés par les valeurs stocké dans les registres précédent.
Le Hardware de l'amiga permet de supporter quatre lecteur de disquettes 3p1/2.
Sur l'Amiga500 et 1000, le lecteur de disquette interne est connu sous le nom de **driver 0** (DF0)
Les lecteurs de disquettes externes sont connectés en séries.
Le lecteur connectée directement a l'ordinateur est le **drive 1** (DF1)
Le lecteur connectée au drive 1 est le **drive 2** (DF2) et le lecteur connecté au drive 2 est le **drive 3** (DF3)
Sur les ordinateurs Amiga 2000, 2500 et 3000, les deux lecteurs de disquette interne sont les **drive 0** et **1**
Le premier lecteur de disquette externe est le **drive 2** (DF2), même si un seul des lecteurs de disquette interne est connecté. N'importe quel lecteur de disquette connecté à ce lecteur de disquette externe sera le drive 3.

Pour sélectionner un lecteur de disquette l'on doit positionner son bit correspondant à 0
Pour désélectionner un lecteur de disquette l'on doit positionner son bit correspondant à 1
Bit 3 drive 0 (**DF0**)
Bit 4 drive 1 (**DF1**)
Bit 5 drive 2 (**DF2)**
Bit 6 drive 3 (**DF3**)

Toutes les combinaisons des lecteurs de disquette peuvent être sélectionné à tout moment.
Les autres Bit de ce registre affectent TOUS les lecteurs sélectionnés. Il est donc possible de réaliser des taches simultanément comme le déplacement de tête sur plus d'un lecteur de disquette.

Bit 7: Ce Bit active ou pas le moteur du lecteur sélectionné.
- **Valeur1** Moteur OFF
- **Valeur0** Moteur ON

L'état ON/OFF peut être visualisé par la petit lumière présente sur le devant du lecteur de disquette.
Ce Bit doit être défini avant de choisir un lecteur. Si un lecteur est déjà sélectionné et vous désirer changer L'état de son moteur, vous devez désélectionner le lecteur, définissez le **bit 7**, puis resélectionnez le lecteur souhaité.

Quand le moteur d'un lecteur est passé à **ON**, vous devez attendre que celui-ci atteigne sa pleine vitesse de rotation avant d'effectuer d'autre opération. Vous pouvez vérifier ceci en lisant le bit 5 du **CIA-A** située à l'adresse : **$BFE001.** Il passe à 0 quand le lecteur a atteint sa pleine vitesse de rotation et que le lecteur est prêt à recevoir de nouvelles commandes.

Le code suivant utilise ce Bit pour passer le moteur a ON sur le drive 0 puis le passer à OFF
Ce programme part bien sur principe que tout le système multitache est désactivé et que vous avez le contrôle total de l'ordinateur.

```
CIAAPRA         equ     $BFE001
CIABPRB         equ     $BFD100

                or.b    #$08,CIABPRB    ;S'assure que le lecteur DFO: est désélectionné
                and.b   #S7F,CIABPRB    ;Motor ON en effaçant le bit 7
                and.b   #$F7,CIABPRB    ;Sélectionne DFO: pour passer le moteur à ON

Wait:           btst.b  #5,CIAAPRA      ;Vérifie le Bit Check RDY
                bne.s   Wait            ;On attend que la pleine vitesse de rotation soit atteinte
                or.b    #S88,CIABPRB    ;Motor Off et désélection de DF0:
                and.b   #$F7,CIABPRB    ;Sélectionne DFO: pour passer le moteur à ON
                or.b    #$08,CIABPRB    ;Désélectionne DFO: pour plus de sécurité.
```

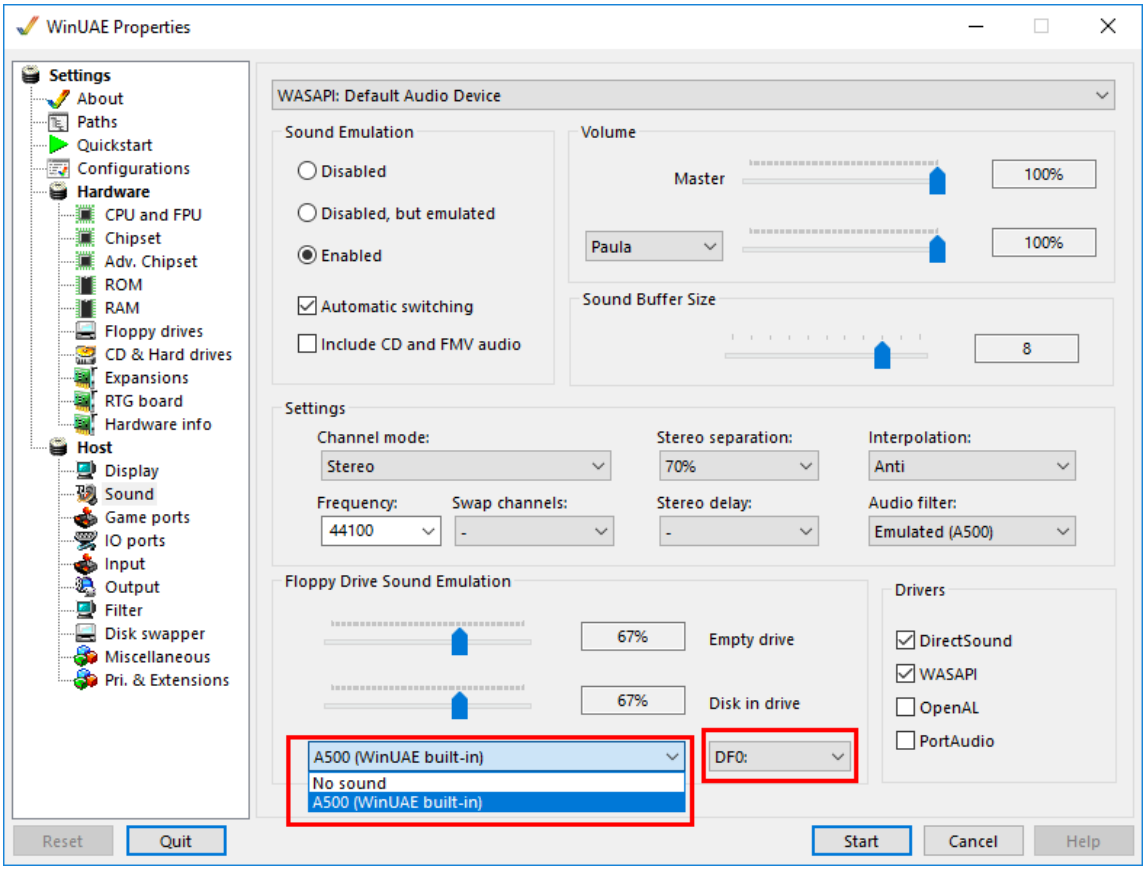
Extra Info :

Le registre **$DFF016 POTGOR**

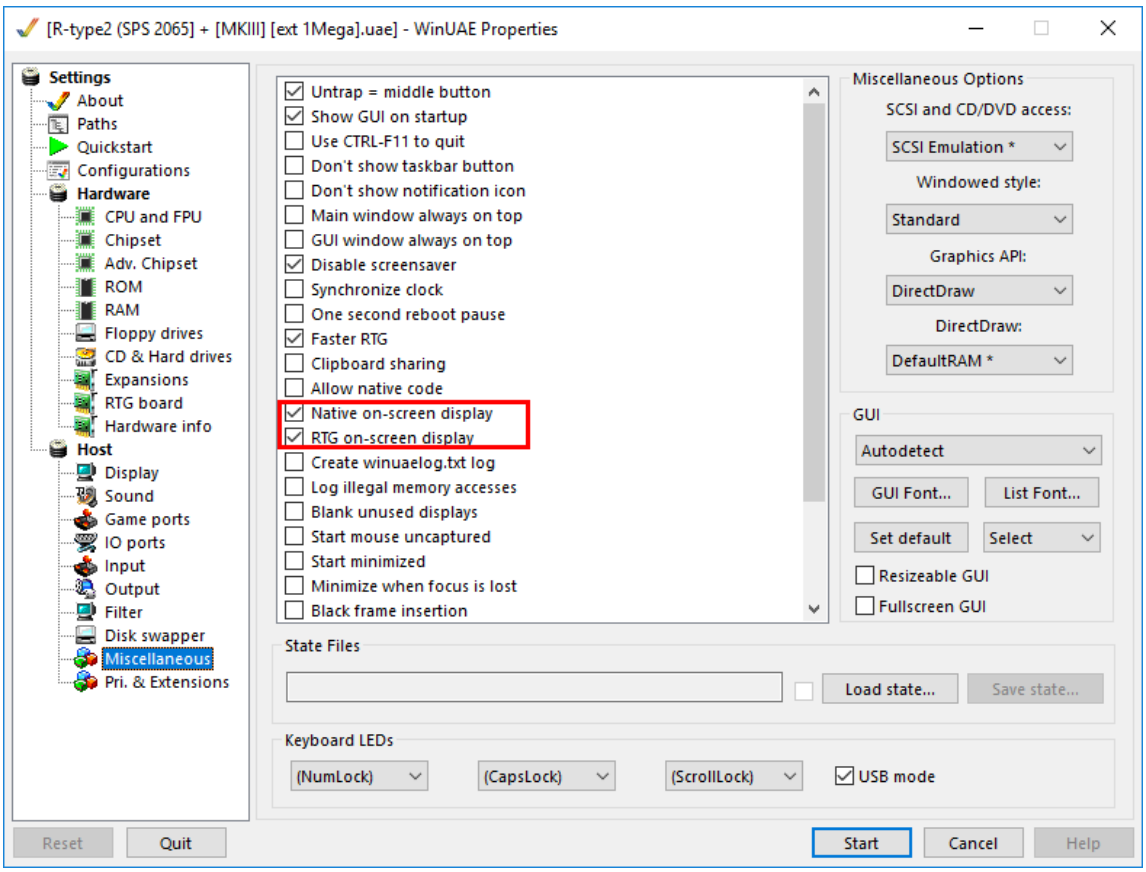
On va juste noter que sont **bit 10** sert pour **tester l'appuie sur le Bouton Droit de la souris**
Exemple : `btst #10, $dff016`

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

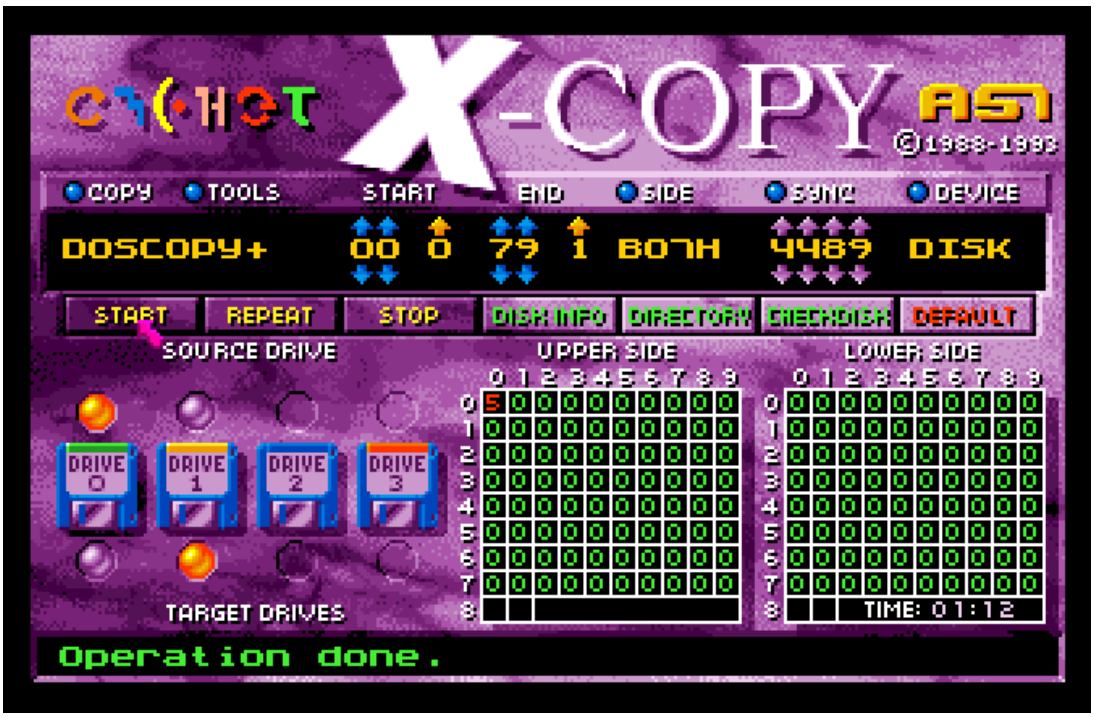


Voir même, pour plus d'information. Par exemple afficher sur quelle face l'on se trouve, d'activer :
Host -> Miscellaneous -> Native on-screen display AND RTG on-screen display



Part 1 X-copy

La première chose à faire et d'essayer de faire un backup de notre disquette.
Pour cela on va utiliser X-Copy.



Hormis la piste 00 de la disquette, l'ensemble semble copiable.
Il est fort probable qu'à cause de la Piste 00 notre backup ne fonctionne pas **mais gardons quand même cette copie sous le coude**.
Ce genre de 'format' ressemble à une simple protection *copylock*.

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

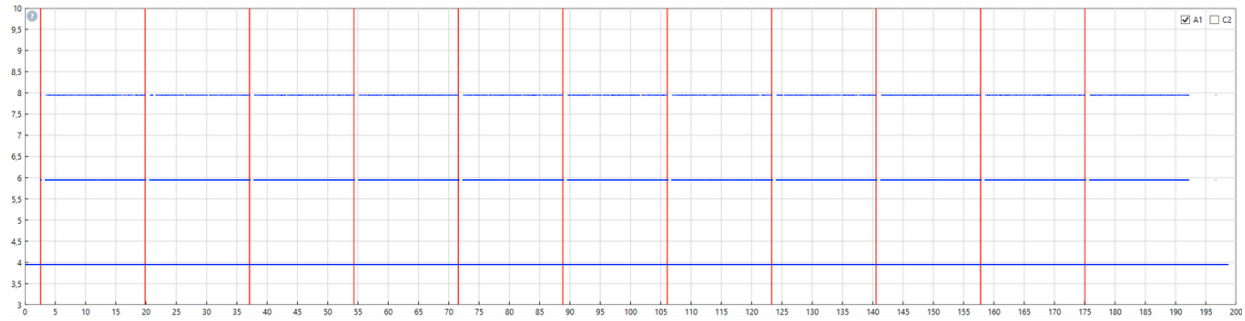
FILENAME	1509_Rodland_AMIGA.ipf
TYPE	Floppy_Disk
ENCODER	CAPS(V1)
FILE	1509(V1)
DISK	1
TRACK	00-83
SIDE	0-1
PLATFORM	Amiga
REVOLUTION	5
PROTECTION	COPYLOCK [T00.1]

Comme prévue, L'ensemble du disque est au format Standard *AmigaDOS* (11 blocks standard)
Ensuite on retrouve notre 'Track' incopiable vu sous *X-Copy*. Elle est identifiée dans le format IPF comme track : **Amiga_Copylock** ainsi que par mon Analyseur d'image IPF.

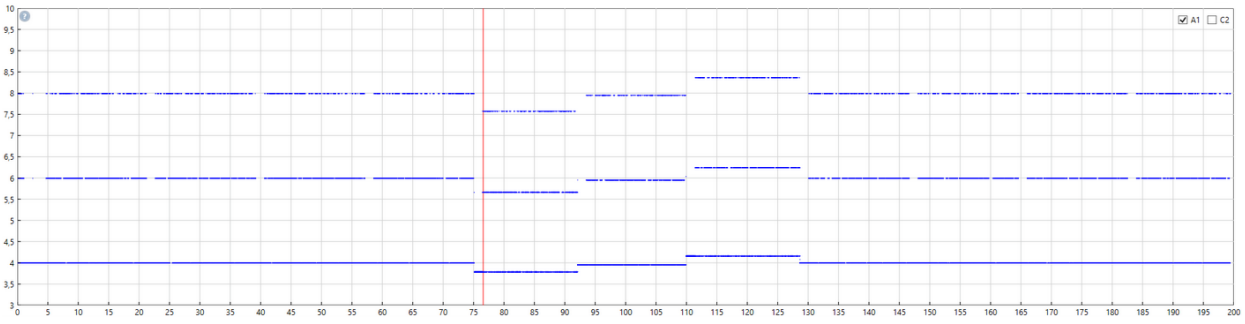
TrackNumber	Size Record (bytes)	Crc	Status	Track Size	Detail Tr. Size	Start Byte	Bit	DataKey	Block	Density	Signal	Encoder	Flag	Aufit Track View
T00.0	80	7AC83653	Good	12534	100272 bits = Data=95744 + Gap=4528	297	2382	001	11	Auto	cell_2us	0	None	
T00.1	80	35E010A4	Good	12500	100000 bits = Data=90992 + Gap=9008	136	1088	002	11	Copylock_Amiga	cell_2us	0	None	
T01.0	80	7269415F	Good	12534	100272 bits = Data=95744 + Gap=4528	297	2383	003	11	Auto	cell_2us	0	None	
T01.1	80	C5CC0D15	Good	12536	100280 bits = Data=95744 + Gap=4536	293	2351	004	11	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				B6FC6982			13256-19701
#0	9	8704	545	512	0	1	MFM	0352	0352	15	13256-13287
#1	10	8704	545	512	0	1	MFM	0906	0906	87	13288-13319
#2	0	8704	545	512	0	1	MFM	1460	1460	92	13320-13351
#3	1	8704	545	512	0	1	MFM	2014	2014	126	13352-13383
#4	2	8704	545	512	0	1	MFM	2568	2568	161	13384-13415
#5	3	8704	545	512	0	1	MFM	3122	3122	196	13416-13447
#6	4	8704	545	512	0	1	MFM	3676	3676	230	13448-13479
#7	5	8704	545	512	0	1	MFM	4230	4230	265	13480-13511
#8	6	8704	545	512	0	1	MFM	4784	4784	300	13512-13543
#9	7	8704	545	512	0	1	MFM	5338	5338	334	13544-13575
#10	8	8704	545	512	4528	284	MFM	5892	5892	369	13576-13607
[T00.1]		6138		49104				5214A82B			19730-25867
#0	N/A	8272	518	N/A	720	46	MFM	0352	0352	15	19730-19761
#1		8272	518		720	46	MFM	0878	0878	55	19762-19793
#2		8272	518		720	46	MFM	1404	1404	88	19794-19825
#3		8272	518		720	46	MFM	1930	1930	121	19826-19857
#4		8272	518		720	46	MFM	2456	2456	154	19858-19889
#5		8272	518		720	46	MFM	2982	2982	187	19890-19921
#6		8272	518		720	46	MFM	3508	3508	220	19922-19953
#7		8272	518		720	46	MFM	4034	4034	253	19954-19985
#8		8272	518		720	46	MFM	4560	4560	286	19986-20017
#9		8272	518		720	46	MFM	5086	5086	318	20018-20049
#10		8272	518		1808	114	MFM	5612	5612	351	20050-20081

Ci-dessous, la vue d'une taille normale *AmigaDos* (que l'on retrouve sur l'ensemble du disque donc)



Et ci-dessous, la vue de la track 00.1, détectée en tant que *Copylock*



Part 3 Cheats

Divers Cheat trouvés avec l'AR et ces fonctions de recherche + recherche dans le code (les marqueurs).
PS : Voir le manuel de l'AR pour l'explication des commandes de recherche de Trainer.

```
$20D48                    : Nombre de vie Player1    (trouver avec l'option TS)
$B360                     : SUBQ.W  #1,44(A4)        Code qui enlève une vie P1(trouver avec la commande TFD $20D48)

Vie infinie               $B360    SUB.W    #1,44(A4)        à remplacer par   TST.W 44(A4)
```

Part 4 Comportement des chargements du jeu et test de notre Backup

Avant d'entrer dans le vif du sujet on va s'attarder 5 minutes pour voir comment le jeu se comporte au niveau des chargements.
Insérez la disquette originale 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 ou info	Piste Lue WinUAE
00➔	**BOOT**	00
9	**TRACKLOAD #01**	01-09
00🡐	***Retour T00 - CopyLock***	**N/A**
➔ 09	*GoTo T09*	N/A
➔13	*GoTo T13*	N/A
3	**N/A**	13-15
➔21	*GoTo T21*	N/A
➔27	*GoTo T27*	N/A
➔79	*GoTo T79*	N/A
15🡐	*Retour T15*	N/A
➔40	*GoTo T40*	N/A
3	**N/A**	40-42
Menu (PRESS FIRE BUTTON TO PLAY GAME)		
PRESS FIRE		
3	**N/A**	42-44
INTRO/ANIMATION		
🡐15	Retour T15	N/A
N/A	**N/A**	15-15
🡐20	Retour T20	N/A
4	**N/A**	20-23
➔31	*GoTo T31*	N/A
3	**N/A**	31-33
LEVEL 1		
GAME OVER		
🡐15	**N/A**	N/A
1	**N/A**	15-16
➔40	**N/A**	N/A
3	**N/A**	40-42
Menu (PRESS FIRE BUTTON TO PLAY GAME)		
PRESS FIRE		

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.

Il est temps maintenant de tester notre copie réalisée avec X-copy.
Insérez notre backup dans le lecteur et booter dessus.

Nbr Piste Lu	Affichage Après chargement ou info	Piste Lue WinUAE
00➔	**BOOT**	00
9	**TRACKLOAD #01**	01-09
00🡐	***Retour T00 - CopyLock***	**N/A**
➔ 09	*GoTo T09*	N/A
- PLANTAGE de l'AMIGA -		

On peut facilement en déduire que le '**Retour T00**' préalablement vue est en fait notre fameux code *CopyLock*

Part 5 Analyse et modification du bootblock

Allons donc voir et analyser notre *bootblock.*
Toujours avec la disquette de backup dans le lecteur.

#RT alias Read Track, permet le chargement de track <Track Start> <Count> <Destination_memoire>
#D, alias Désassemble
Taper : `RT 0 1 20000` puis `D 20000+c`

```
rt 0 1 20000
Disk ok

d 20000+c
~02000C MOVEA.L A1,A5
~02000E MOVE.W  #0,00DFF180
~020016 MOVE.L  #32000,D0
~02001C MOVEQ   #3,D1
~02001E MOVEA.L 00000004.S,A6
~020022 JSR     -C6(A6)
~020026 TST.L   D0
~020028 BEQ     00020064
~02002A MOVEA.L D0,A6
~02002C LEA     100(A6),A6
~020030 LEA     1A(A6),A4
~020034 MOVE.L  00000004.S,(A6)
~020038 MOVEA.L A5,A1
~02003A MOVE.L  A4,28(A1)
~02003E MOVE.L  #1600,24(A1)
~020046 CLR.L   2C(A1)
~02004A MOVE.W  #2,1C(A1)
~020050 MOVE.L  A6,-(A7)
~020052 MOVEA.L (A6),A6
~020054 JSR     -1C8(A6)
~020058 MOVEA.L (A7)+,A6
~02005A TST.L   D0
~02005C BNE     00020064
~02005E JMP     6C(A4)
;=====================================
~020062 ORI.B   #F8,D0
~020066 ORI.B   #70,D2
~02006A JMP     (A0)
;=====================================
```

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

Regardons ça en détail :

```
============================
2000C    MOVEA.L  A1,A5             ; Sauvegarde de A1 dans A5
2000E    MOVE.W   #0,DFF180         ; Fond D'écran noir (pas vraiment utile)
20016    MOVE.L   #32000,D0         ; D0=32000, taille de la mémoire à réserver
2001C    MOVEQ    #3,D1             ; D1=3      (req)
2001E    MOVEA.L  4.S,A6            ; ExecBase dans A6 (pas vraiment utile car il est déjà mais bon…)
20022    JSR      -C6(A6)           ; Appel AllocMem  (Réservation mémoire)
                                    ;
20026    TST.L    D0                ; Test de D0 qui doit contenir l'adresse de début réservée.
20028    BEQ      20064             ; si égale à Zero, c'est que l'allocation n'a pas fonctionnée.
                                    ; On branche alors en 20064(PC) qui provoque un crash.
                                    ;
2002A    MOVEA.L  D0,A6             ; Sinon A6=D0=Adr_de_la_Zone mémoire réservée
2002C    LEA      100(A6),A6        ; A6=$100+A6
20030    LEA      1A(A6),A4         ; A4=$1A+A6
20034    MOVE.L   4.S,(A6)          ; ExecBase remis dans (A6)
20038    MOVEA.L  A5,A1             ; On restaure A1
                                    ;
                                    ; Préparation pour l'appel au TrackDisk.Device
2003A    MOVE.L   A4,28(A1)         ; 28(A1) = $A4            Adr. Mémoire de destination.
2003E    MOVE.L   #1600,24(A1)      ; 24(A1) = $1600          Taille à lire, à savoir 1 Track.
20046    CLR.L    2C(A1)            ; 2C(A1) = 0              Adr. disk de départ.
2004A    MOVE.W   #2,1C(A1)         ; 1C(A1) = 2              Mode lecture.
20050    MOVE.L   A6,-(A7)          ; Sauve A6 dans la pile.
20052    MOVEA.L  (A6),A6           ; A6=(A6)
20054    JSR      -1C8(A6)          ; Appel du TrackDisk.Device qui va charger donc 1 Track
                                    ;
20058    MOVEA.L  (A7)+,A6          ; Restaure A6
2005A    TST.L    D0                ; si égale à Zero, c'est que le TrackDisk.Device n'a pas fonctionné.
2005C    BNE      20064             ; On branche alors en 20064(PC) qui provoque un crash.
                                    ;
2005E    JMP      6C(A4)            ; Sinon, exécution du code chargé, à savoir GoTo #1er_Saut
============================
20062    ORI.B    #F8,D0            ;
20666    ORI.B    #70,D2            ;
2006A    JMP      (A0)              ;
============================
2006C    BSR      200EE             ; #1er_Saut
...
```

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 sur ce code.

Taper :
#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 'Track', ak $1600

On va désactiver le `MOVE.W #0,DFF180` en *$2000E*

```
A 2000E
$02000E  BRA    2000E           ; Notre Dead-Loop
$020010  NOP
$020012  NOP
$020014  NOP
$020016  <RETURN>
```

On recalcul le checksum de notre code final
```
BOOTCHK 20000
```

Et on écrit le nouveau *bootblock*
```
WT 0 1 20000
```

Rebooter ensuite votre Amiga normalement.

Part 6 Let's Trace

Le *bootblock* se charge et nous sommes maintenant dans notre *DeadLoop*, **Entrer** dans l'**AR, taper :** `D`

Normalement, avec un **Amiga 500** sans extension mémoire vous devriez être en **$5C4E** et avec extension mémoire, en **$1566**
Adapter le tuto en conséquence bien sûr.

1er chose, on désactive notre *DeadLoop*, **taper :** `A 5C4E`

```
A 5C4E
$005C4E  NOP
$005C50  <RETURN>
```

```
d
~005C4E BRA     00005C4E
;=====================================
^005C50 NOP

a 5C4E
^005C4E nop
^005C50
```

On pause un *BreakPoint* sur le **TST.L** en **5C66** et le dernier **JMP** en **$5C9E**
#BS, alias BreakPoint. Permet, dès que l'adresse mémoire indiquée est atteinte, d'effectuer un arrêt du code.

Taper : `BS 5C66` puis `BS 5C9E` et on retourne au code avec la commande `X`

```
d 5C66
~005C66 TST.L   D0
~005C68 BEQ     00005CA4
~005C6A MOVEA.L D0,A6
~005C6C LEA     100(A6),A6
~005C70 LEA     1A(A6),A4
~005C74 MOVE.L  00000004.S,(A6)
~005C78 MOVEA.L A5,A1
~005C7A MOVE.L  A4,28(A1)
~005C7E MOVE.L  #1600,24(A1)
~005C86 CLR.L   2C(A1)
~005C8A MOVE.W  #2,1C(A1)
~005C90 MOVE.L  A6,-(A7)
~005C92 MOVEA.L (A6),A6
~005C94 JSR     -1C8(A6)
~005C98 MOVEA.L (A7)+,A6
~005C9A TST.L   D0
~005C9C BNE     00005CA4
~005C9E JMP     6C(A4)
;=====================================

bs 5C66
Breakpoint inserted
bs 5C9E
Breakpoint inserted
Ready.
x
```

Notre 1er *BreakPoint* est atteint, on entre automatiquement dans l'AR.

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

D0	0000A498	00000003	00000001	00000000	00000000	00000000	FFFFFFFF	00000000
A0	000008C2	0000A498	FFFFFFFF	00FE86EE	00005C40	000018E2	00000676	00080000

En **D0**, notre adresse de début réservée, à savoir *$A498*, ce qui nous donnes comme adr. de destination pour la Trackload :

```
2002A   MOVEA.L  D0,A6          ; A6=D0=(dans notre cas)=$A498
2002C   LEA      100(A6),A6     ; A6=$100+A6=$100+$A498=$A598
20030   LEA      1A(A6),A4      ; A4=$1A+A6=$1A+$A598=$A5B2
...
2003A   MOVE.L   A4,28(A1)      ; Adr. de destination mémoire = A4 = $A5B2
```

On retourne au traitement du code avec la commande **X**, très vite notre second *BreakPoint* est atteint, on entre automatiquement dans l'AR.
Taper : R

D0	00000000	0000001F	00000001	00000000	00000000	00000000	FFFFFFFF	00000000
A0	0000192A	000018E2	FFFFFFFF	00FE86EE	0000A5B2	000018E2	00000A598	00080000

Le jump qui va suivre devrait donc sauter en **6C(A4)**, à savoir $6C+$A5B2=$A61E

#ST, alias TRACE. Permet de tracer le code avec les sous-routine, il est possible de lui indiquer le nombre de 'pas'
Taper : ST

```
A61E    BSR     A6A0            ; GoSub ➔ #TrackLoad_#1
```

Petit calcul simple : $A6A0-$A5B2=**$EE**

Nous sommes donc toujours dans la **Track0** du disque.
On regarde le code en $A6A0, **Taper : D A6A0** et on suit le code.

```
============================        ; #TrackLoad_#1
A6A0     LEA      A852(PC),A4       ; A4=A852
A6A4     MOVEA.L  A4,A2             ; A2=A4
A6A6     MOVE.L   A6EC(PC),D0       ; D0=(A6EC)
A6AA     LEA      0(A2,D0.L),A3     ; 0+$A852+$162BD=$20B0F            A3=$20B0F
A6AE     MOVE.L   #2C00,D2          ; D2=2C00
                                    ;
                                    ; ➔
A6B4     MOVEA.L  A5,A1             ; A5=A1
A6B6     MOVE.L   A2,28(A1)         ; 28(A1) = A2 = $A852      Adr. Mémoire de Destination
A6BA     MOVE.L   #1600,24(A1)      ; 24(A1) =      $1600      Taille à lire
A6C2     MOVE.L   D2,2C(A1)         ; 2C(A1) =      $2C00      Adr. disk de départ.
                                    ;                          Début de la 2nd track, en partant de 0 bien sur
                                    ;                          Track0= boot, Track1=Copylock, Track2=Code
                                    ;
A6C6     MOVE.W   #2,1C(A1)         ; 1C(A1) = 2               Mode lecture
A6CC     MOVE.L   A6,-(A7)          ;
A6CE     MOVEA.L  (A6),A6           ;
A6D0     JSR      -1C8(A6)          ; Appel du TrackDisk.Device
                                    ;
A6D4     MOVEA.L  (A7)+,A6          ;
A6D6     TST.L    D0                ; si égale à Zero, c'est que le TrackDisk.Device n'a pas fonctionné.
A6D8     BNE      A616              ; On branche alors en A616(PC) qui provoque un crash.
                                    ;
A6DC     ADDI.L   #1600,D2          ; D2=D2+$1600     On avance d'une track sur la position disk à lire
A6E2     ADDA.W   #1600,A2          ; A2=A2+$1600     On avance de la taille d'une track en mémoire de destination
A6E6     CMPA.L   A3,A2             ; On compare le pointeur actuel, à savoir A2, avec A3 (à savoir 20B0F)
A6E8     BCS      A6B4              ; ➔ On boucle tant que notre compteur A3 n'est pas égale à A2
                                    ;
                                    ; Va donc charger $20B0F-$A852=$162BD=!90813
                                    ; $1600=!5632     !90813/!5632=16   Track Lues
                                    ; En partant de la Piste 1 (puisque début en Track2),
                                    ; ça nous donne fin de lecture en piste 9
                                    ;
                                    ; 16 Tracks = 8 pistes, 8+1=9
                                    ;
A6EA     RTS                        ; E.T Retour Maison
============================
```

Utilisation encore du **Trackdisk.device** pour le chargement de plusieurs Tracks et on retourne en *$A622*

```
============================
A61E     BSR      A6A0              ; GoSub ➔ #TrackLoad_#1
                                    ;
                                    ; #Kill_Sys_and_co
A622     MOVEA.L  A5,A1             ;
A624     MOVE.W   #9,1C(A1)         ;
A62A     CLR.L    24(A1)            ;
A62E     MOVE.L   A6,-(A7)          ;
A630     MOVEA.L  (A6),A6           ;
A632     JSR      -1C8(A6)          ;
A636     MOVEA.L  (A7)+,A6          ;
A638     MOVE.L   A4,-(A7)          ;
A63A     MOVEA.L  A4,A0             ;
A63C     BSR      A6F4              ; GoSub ➔ #Decrunch
         ================================================================================
         A6F4     MOVEM.L  D1-D7/A0-A6,-(A7)          ; #Decrunch
         A6F8     MOVEA.L  A0,A1                      ;
         A6FA     BSR      A754                       ;
         A6FC     CMP.L    #524E4301,D0               ; ← Signature RCN Décomp. $52 = R   $4E = N   $43 = C
         A702     BNE      A74E                       ;
         A704     ...                                 ;
         A744     BRA      A71C                       ;
         ================================================================================
A640     MOVEA.L  (A7)+,A4          ;
A642     MOVE.L   #32000,D0         ;
A648     MOVEA.L  A4,A0             ;
A64A     MOVE.W   #7FFF,DFF096      ; Conf DMACON
A652     MOVE.W   #7FFF,DFF09A      ; Conf POTINP
A65A     LEA      A664(PC),A1       ; A1=$A664
A65E     MOVE.L   A1,84.S           ; On met A1 en $84
A662     TRAP #1                    ; Et on TRAP à cette adresse
                                    ;
A664     MOVE.W   #2700,SR          ; Conf. Status register
A668     LEA      A690(PC),A1       ;
A66C     LEA      7FF00,A2          ; A2=$7FF00
A672     MOVEA.L  A2,A7             ;
A674     MOVE.L   (A1)+,(A2)+       ; Recopie de la 'boucle de copie du code', vers (A2), à savoir $7FF00
A676     MOVE.L   (A1)+,(A2)+       ; 'boucle de copie du code' = recopie vers l'adr. Mémoire $100
A678     MOVE.L   (A1)+,(A2)+       ;
A67A     MOVE.L   (A1)+,(A2)+       ;
A67C     MOVE.L   (A1)+,(A2)+       ;
A67E     MOVE.L   (A1)+,(A2)+       ;
A680     MOVE.L   (A1)+,(A2)+       ;
A682     MOVE.L   (A1)+,(A2)+       ;
A684     MOVE.W   #8F,dff180        ; Changement de la couleur de fond
A68C     JMP      -20(A2)           ; GoTo #7FF00
============================
```

```
============================
7FF00    LEA      100.S,A1          ; A1=$100
7FF04    LSR.L    #2,D0             ; Décale de 2bits vers la Droite de D0
                                    ; Avant traitement D0=$32000 =     1100100000 00000000
                                    ; Après traitement D0=$C800  =     0011001000 00000000

7FF06    MOVE.L   (A0)+,(A1)+       ; ➔ Copie (A0) vers (A1), puis A0=A0+4 et A1=A1+4     et A0 au début=$A852
7FF08    DBF      D0,7FF06          ; ← D0=D0-1, tant que D0 est différent de -1, on boucle en $7FF06
                                    ; Donc ($C800+1)*4= $32004 Octets de copié
                                    ;
7FF0C    JMP      100.S             ; GoTo $100
============================
```

```
============================        ;
100      BRA      3A6E              ; GoTo ➔ #CopyLock_Base
============================        ;
```

```
============================        ; #CopyLock_Base
3A6E     BSR      4F0C              ; GoSub ➔ #Register_conf_01
3A72     BSR      4AF2              ; GoSub ➔ #Cleanup
3A76     MOVEQ    #FFFFFFFF,D0      ;
3A78     MOVE.L   D0,6A(A6)         ;
3A7C     MOVE.L   D0,6E(A6)         ;
3A80     MOVEQ    #0,D0             ;
3A82     MOVEQ    #0,D1             ;
3A84     MOVEQ    #0,D3             ;
3A86     PEA      3A92(PC)          ; #CopyLock
3A8A     MOVE.L   (A7)+,10          ;
3A90     ILLEGAL                    ; ... ... ...
3A92     MOVEM.L  D0-D7/A0-A7,(-A7) ; ... ... ... ...
3A96     PEA      3AB2(PC)          ; ... ... ... ... ...
3A9A     MOVE.L   (A7)+,10          ; ... ... ... ... ... ... Ça ressemble à du code copylock
...
============================
```

On regarde vite fait les deux **BSR**

```
============================        ; #Register_conf_01
4F0C     LEA      DFF000,A4         ;
4F12     MOVE.W   #7FFF,9A(A4)      ; Conf POTINP
4F18     MOVE.W   #7FFF,96(A4)      ; Conf DMACON
4F1E     MOVE.W   #8640,96(A4)      ; Conf DMACON
4F24     MOVE.W   #C000,9A(A4)      ; Conf POTINP
4F2A     MOVE.W   #0,DFF180         ; Couleur de fond=Noir
4F32     BSET     #1,BFE001         ; Conf CIA-A (led & filtre passe bas)
4F3A     RTS                        ;
============================        ; E.T Retour Maison
```

```
============================        ; #Cleanup
4AF2     LEA      1F562,A6          ; A6=$1F562
4AF8     LEA      (A6),A0           ; A0=(A6)
4AFA     MOVE.W   #1CC3,D7          ; D7=$1CC3
4AFE     CLR.W    (A0)+             ; ➔ Clear (A0) en Word puis A0=A0=2
4B00     DBF      D7,4AFE           ; ← D0=D0-1, tant que D0 est différent de -1, on boucle en $4AFE
                                    ; $1CC3+1=$1CC4   $1CC4*2=$3988 Octet de 'nettoyé'
                                    ;
4B04     ST       34FA(A6)          ; $34FA(A6) = %11111111
4B08     RTS                        ; E.T Retour Maison
============================
```

Part 7 Copylock ou pas finalement ?

On retourne à notre code, en $3A90, **Taper : M 3A90**

```
M 3A90
:003A90 4A FC 48 E7 FF FF 48 7A 00 1A 23 DF 00 00 00 10 JüH...Hz..#ß....
```

Le code ressemble vraiment à celui d'une routine de *copylock*. En général, en tout cas dans sa version 1, elle fait environ $520 de long.
On regarde donc logiquement en **$3A9A+$520=$3FBA**, **Taper : M 3FBA**

On déroule le code jusqu'à trouver une 'fin' probable du code probable *copylock*.
Bon... plus long que prévu. On trouve la fin en **$43B8**

```
============================
...
43A8    MOVEA.W  -(A7),A2          ; ...
43AA    MOVE.L   D0,3A6A.S         ; Copy de D0 à l'adresse mémoire $3A6A
43AE    ADDI.L   #51C4631D,D0      ; On ajoute 51C4631D à D0
43B4    BEQ      43BA              ; Si résultat = 0 alors GoTo → $43BA
43B6    ADDQ.W   #1,A7             ; Ajoute 1 a la pile, ce qui donnera une adresse impaire au retour
                                   ; et donc à coup sûr un crash de l'amiga lors du prochain RTS
43B8    RTS                        ;
============================       ; #Start_Base_Game
43BA    LEA      3634(A6),A7       ;
43BE    MOVE.W   #20,DFF09A        ; Conf POTINP
...
```

Retirez la disquette de backup du lecteur pour l'instant **et insérez la disquette originale dans le lecteur.**
On va changer le code en $43AA et crée une DeadLoop.

Taper :

```
A 43AA
$0043AA  BRA    43AA            ; Notre Dead-Loop
$0043AC  <RETURN>
```

Puis, on retourne au code : **X**
Après un retour en **T00**, suivi d'un retour en **T09,** nous sommes bloqués dans notre *DeadLoop*
Entrer dans l'**AR, taper : R**

D0	AE3B9CE3	00000000	0001A200	00000000	00000000	00000000	FFFFFFFF	0000FFFF
A0	00022EEA	00032104	0007FF20	00020B0F	00DFF000	000018E2	0001F562	0007FF00

D0 = AE3B9CE3
On reprend le code en $43AE, à savoir le **ADD**
$51C4631D+$AE3B9CE3=$1 0000 0000 Le registre étant en 32bits, le résultat sera **$0000 0000**

L'instruction suivante est un branchement en **$43BA** si le résultat = **0** (ce qui est le cas. Logique, nous sommes avec l'original)
Nous sommes donc bien ici en présence d'un code *CopyLock*
Je vous invites à faire le même test avec notre disque de Backup, vous verrez que **D0** donne comme valeur de retour **00000000**
Le **BEQ** ne s'effectuera donc pas (en effet $51C4631+0 n'est pas égale à Zéro)

La *clé_Copylock* de notre originale est donc : $AE3B9CE3
Noté qu'on aurait aussi pu la calculer car si 51C4631D+*clé_Copylock*=0
Alors mathématiquement, *clé_Copylock*=0-$51C4631D = **$AE3B9CE3**

Noter que cette clé est copiée à l'adresse mémoire **$3A6A** (MOVE.L D0,3A6A.S)

On peut aussi déduire que l'on est ici dans la version **2 du copylock** (car la taille de la routine est plus grande que $520)

	Taille du code (environ)	Adr. Mémoire pour la clé
CopyLock V1	**$520**	**$24**
CopyLock V2	**$920**	**$60**

D'ailleurs, on vérifie. **Taper : M 60**

```
M 60
:000060 AE 3B 9C E3 00 FC 0C 8E 00 FC 0C E2 00 FC 0D 14 .;...ü...ü...ü..
```

On retrouve bien notre clé à cette adr. mémoire.

Si on en croit le chapitre sur le '**comportement des chargements**', le code de *CopyLock* est effectué uniquement une fois.
(du moins, pour l'instant… à voir sur le test complet du jeu si une autre partie du code fera appel à la vérification de cette piste protégée.)

Part 8 Bypasser en Live le Copylock

Retirez la disquette originale du lecteur et **insérez notre disquette de backup dans le lecteur.**
On va tester si on peut bypasser facilement ce *CopyLock.* Mais déjà, on va désactiver *notre DeadLoop* que nous avons créé sur le *BootSecteur*
Taper : `RT 0 1 20000` puis `D 20000+c`

Puis : `A 2000E`
`$02000E  NOP`
`$020010  <RETURN>`

On recalcul le checksum de notre code final
`BOOTCHK 20000`

Et on écrit le nouveau *bootblock*
`WT 0 1 20000`

Rebooter ensuite votre Amiga normalement.
Laisser le *TrackLoad* commencer mais **juste avant d'atteindre la 9em Piste, entrer dans l'AR** et poser un *BreakPoint* en $A68C

Taper : `BS A68C` puis, on retourne au code avec la commande `X`
Rapidement notre *BreakPoint* en $A68C est atteint, **on entre automatiquement dans l'AR.**
Les données sont maintenant chargées puis décompressées et le code copié en *$7FF00*

Taper : `BS 7FF0C` puis, on retourne au code avec la commande `X`
Rapidement notre *BreakPoint* en $7FF0C est atteint, **on entre automatiquement dans l'AR.**
Le code est maintenant copiée en *$100* et on s'apprête à 'sauter' vers `#CopyLock_Base`

Taper : `BDA`
#BDA, alias AllBreakPoint Delete. Permet de supprimer tous les breakpoints

On remet une *DeadLoop* en sortie du code de *copylock*

Taper :
`A 43AA`
`$0043AA  BRA  43AA` ; Notre Dead-Loop
`$0043AC  <RETURN>`

Rappel :
`43AA  MOVE.L  D0,3A6A.S` ; Copy de la clé CopyLock à l'adresse mémoire $3A6A

Puis, on retourne au code : `X`
Après un retour en **T00**, suivi d'un retour en **T09,** nous sommes logiquement bloqués dans notre *DeadLoop.*
Entrer dans l'**AR, taper : R**

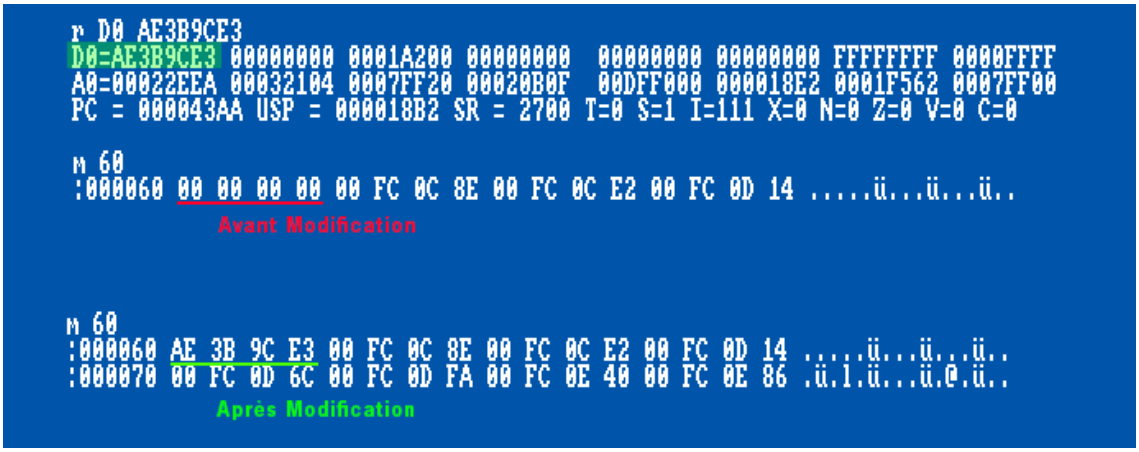
Comme attendu et comme nous ne sommes pas sur la disquette originale, pas de clé *CopyLock* en **D0**

D0	00000000	00000000	0001A200	00000000	00000000	00000000	FFFFFFFF	0000FFFF
A0	00022EEA	00032104	0007FF20	00020B0F	00DFF000	000018E2	0001F562	0007FF00

Nous allons remédier à ça, **Taper :**
`A 43AA`
`$0043AA  MOVE.L  D0,3A6A.S` ; On remet le code d'origine
`$0043AE  <RETURN>`

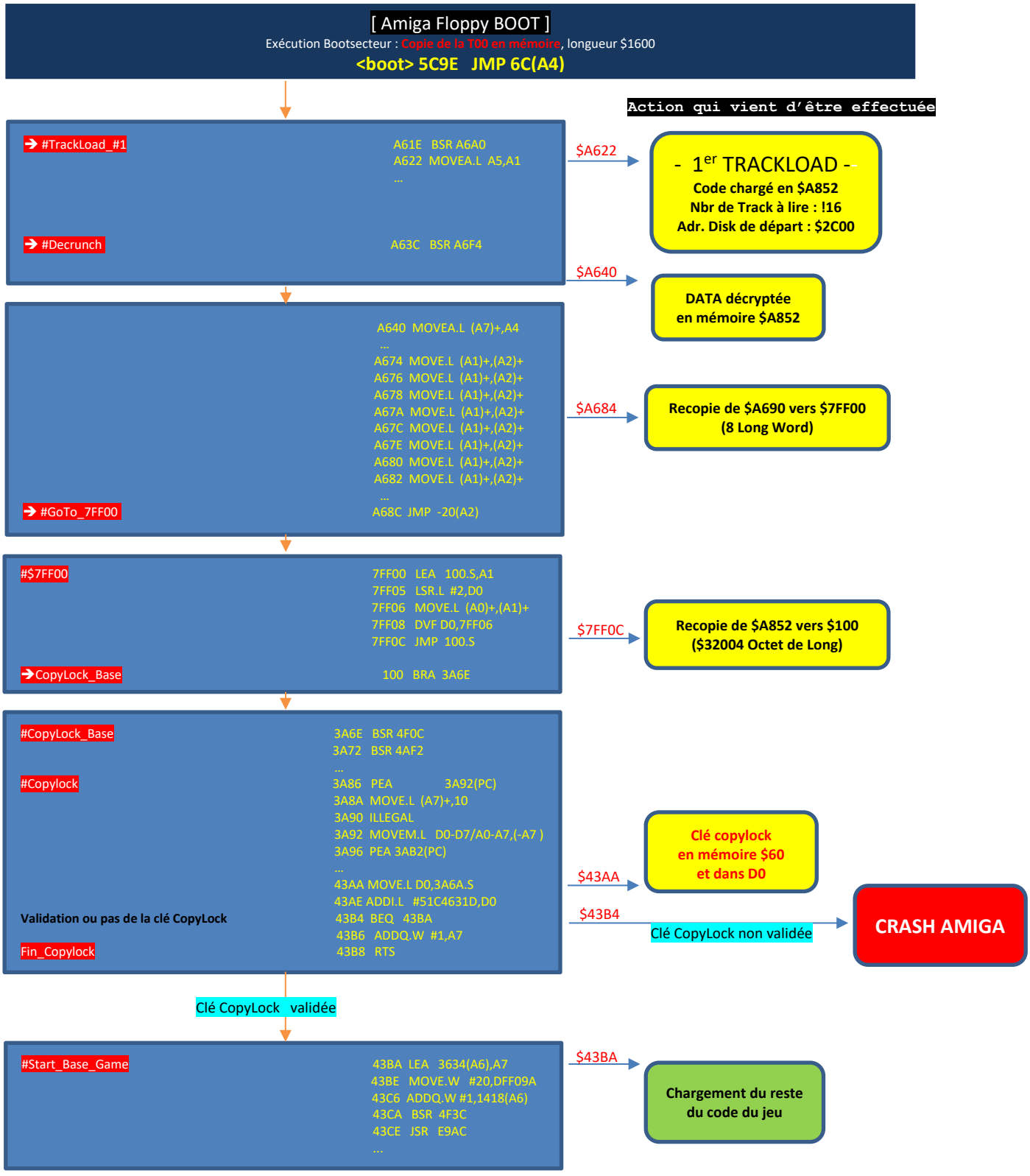
Et on patch en direct la clé, **taper :**
`R D0 AE3B9CE3`

Sans oublier de l'écrire aussi en mémoire à l'adresse *$60*, **Taper :** `M 60` et entrer la valeur de la clé *CopyLock*



Et on retourne au code : `X`
Cette fois-ci l'Amiga ne crash pas et le chargement s'effectue, nous arrivons rapidement sur le menu principal du jeu.
On lance une partie, le jeu fonctionne. 😉

Part 9 Organigramme des diverses étapes



Part 10 Crack Copylock 'Test'

On va tester si ça fonctionne en modifiant le code d'origine, **Rebooter votre Amiga**

Laisser le *trackload* **commencer** et **avant d'atteindre la 9eme piste, Entrer dans l'AR**

Taper : `BS A68C` puis, on retourne au code avec la commande `X`
Rapidement notre *BreakPoint* en $A68C est atteint, **on entre automatiquement dans l'AR.**
Les données sont maintenant chargées puis décompressées et le code copié en *$7FF00*

Taper : `BS 7FF0C` puis, on retourne au code avec la commande `X`
Rapidement notre *BreakPoint* en $7FF0C est atteint, **on entre automatiquement dans l'AR.**
Le code est maintenant copiée en *$100* et on s'apprête à 'sauter' vers `#CopyLock_Base`

Taper : `BS 3A86` puis, on retourne au code avec la commande `X`
Rapidement notre *BreakPoint* en $3A86 est atteint, **on entre automatiquement dans l'AR.**
Nous sommes exactement au début du code du *CopyLock*

Taper : `BDA`

Il est maintenant temps de créer notre bout de code en remplacement de la routine de *copylock.*

Taper :

```
A 3A86
$003A86   MOVE.L   #AE3B9CE3,D0          ; Met la clé CopyLock dans D0
$003A8C   MOVE.L   D0,60.S               ; Copie celle-ci en mémoire $60
$003A90   BRA      43AA                  ; Branche en sortie de la routine CopyLock
$003A94   <RETURN>
```

Et on retourne au code : `X`

Le code semble fonctionné, le chargement continu mais…

Très rapidement, nous avons droit à un plantage de l'Amiga.

Part 11 Protection(s) CRC

La seule différence entre les deux méthodes que l'on vient d'utiliser pour ByPasser le *CopyLock*
est que d'un côté on a modifié en mémoire et de l'autre on a Bypassé le *CopYlock* fait en 'direct'.

La probabilité que l'on est ici une **seconde protection** de type *Checksum* est très probable.
Il est aussi impossible que celle-ci soit dans le code *CopyLock.*

On va donc s'intéresser au code qui suit la validation du *CopyLock* à savoir #Start_Base_Game dans notre organigramme.
Il commence en *$43BA.* **Rebooter votre Amiga et revenez au point précédent**, juste après avoir créé notre bout de code en *$3A86*
Bien sûr, n'effectuez pas un retour avec la commande **X**
Taper : D 43BA

```
================================
43BA    LEA     3634(A6),A7     ;
43BE    MOVE.W  #20,DFF09A      ; Conf POTINP
43CC6   ADDQ.W  #1,1418(A6)     ; (A6)+1418=$2097A
43CA    BSR     4F3C            ;
43CE    BSR     E9AC            ;
43D4    BSR     BC00            ;
43D8    BSR     5196            ;
43DC    BSR     B98C            ;
43E0    JSR     C552            ;
43E6    JSR     E8E6            ;
43EC    BSR     BE76            ;
43F0    SUBQ.W  #1,1418(A6)     ;
43F4    BNE     43FE            ;
43F6    MOVE .W #8020,DFF09A    ;
43FE    LEA     4412(PC),A0     ;
4402    MOVEQ   #0,D0           ;
4404    MOVE.W  #116,D1         ; Rien de spécial jusqu'ici, pas de crash de l'Amiga
                                ; Tests effectuées avec des BreakPoints sur les divers BSR/JSR ci-dessus
4408    BSR     51D8            ; Saut vers (PC)+
440C    BSR     5124            ;
4410    BRA     440C            ;
================================
5124    LEA     10C0(A6),A5     ;
5128    BSR     513C            ; GoSub $513C
512A    RTS                     ; E.T Retour Maison
================================
513C    MOVE.L  A7,E(A5)        ; Portion de code assez intéressante
5140    MOVEA.L (A5),A5         ; car on effectue une modification de A7 par rapport à E(A5)
5142    MOVEA.L E(A5),A7        ; Ce qui potentiellement, peut faire cracher l'Amiga. (et c'est le cas)
5146    RTS                     ; E.T Retour Maison
================================
```

Il nous est bien sur impossible de désactiver le **BSR 513C**, sinon le code bouclerait à l'infini. Les sauts sont calculés.
Nous allons donc chercher une routine de CheckSum.
En général c'est une fonction **ADD** d'un registre d'adresse vers un marquer ou un registre de donnée le tout dans une boucle.

On donc regarde qu'est ce qui pointe vers **$513C, Taper : FA 513C**
La liste est longue, nous allons regarder chaque adresse minutieusement, que ce soit le code lui-même à cette adresse.
Comme les appels à ces 'points d'adresse', Comme les appels au début de l'adresse de ces 'points d'adresse'.
à savoir :

- En direct : Analyse du code à l'adresse en question (au-dessus et en dessous)
- En appel Direct : **FA** sur l'adresse en question.
- En appel du débutPotentiel : **FA** sur le 'début' potentiel du code à l'adresse en question.

Exemple avec : $44C4

En direct
D 44C4 **et on remonte dans le code.**

```
$0044B4     …                       ;
$0044B8     BSR     44CA            ; Un BSR n'est pas l'instruction que l'on recherche
$0044BA     BSR     45A6            ; idem
$0044BE     JSR     CA14            ; idem
$0044C4     BSR     513C            ; idem
$0044C8     BRA     44B4            ; idem
```

En Appel direct
FA 44C4 **et on analyse tous les points.**
Dans notre exemple, aucun point d'appel.

En Appel du débutPotentiel
D 44C4 **et on remonte dans le code jusqu'a trouver un 'début de routine possible'**
Normalement, c'est juste quelques lignes au-dessus.
Dans notre exemple, pas de 'début de routine potentiel' trouvé, code beaucoup trop long au-dessus.

Un autre exemple avec : $4944

En direct
D 4944 et on remonte dans le code.

```
$004938         …                               ;
$00493A         BSR     49A6                    ; Un BSR n'est pas l'instruction que l'on recherche
$00493E         MOVE.W  3982(A6),3980(A6)       ; Bof…, pas de compteur, pas de boucle… rien
$004944         BSR     513C                    ; idem
$004948         BSR     463E                    ; idem
$00494C         BEQ     4954                    ; idem
$00494E         TST.B   3523(A6)                ; idem
$004952         BEQ     4944                    ; idem
$004954         MOVEQ   #1E,D0                  ; idem
$004956         BSR     5174                    ; C'est pas ce que l'on recherche...
```

En Appel direct
FA 4944 et on analyse tous les points.
4938 BNE 4944 → déjà analysé au-dessus
4952 BEQ 4944 → déjà analysé au-dessus

En Appel du débutPotentiel
D 4944 et on remonte dans le code jusqu'a trouver un 'début de routine possible'
Normalement, c'est juste quelques lignes au-dessus.
et on remonte dans le code.

```
; =====================================
$004924         BSR     C128            ;
$004928         ...                     ;
```

Donc, **FA 4924** comme début de la routine
491C BCS 4924

Et on regarde ce bout de code
D 491C et on remonte dans le code.

```
; =====================================
$00490E         MOVEA.L 157E(A6),A0             ; nop
$004912         BSR     BB6E                    ; tjrs pas
$004916         CMPI.W  #2,352E(A6)             ; nop
$00491C         BCS     4924                    ; et re nop, c'est pas ici.
$00491E         ...                             ;
```

Il nous reste donc plus qu'à 'analyser' chaque appel. (Tableau ci-dessous)

Points d'Adr.	En direct	En Appel direct	En Appel du débutPotentiel
44C4	Rien de probant	Rien	Code trop long, ne correspond pas
4794	Rien de probant	Rien de probant	Code trop long, ne correspond pas
4944	Rien de probant	Rien de probant	Rien de probant
5128	Rien de probant	Rien	Rien de probant
5130	Rien de probant	Rien	Rien de probant
5138	Rien de probant	Rien	Rien
517C	Rien de probant	Rien de probant	Rien de probant
518C	Rien de probant	Rien de probant	Code Intéressant en 4EDE (eu dessous)
51C4	Rien de probant	Rien	Rien de probant
51E8	Rien de probant	Rien	Rien de probant
520C	Rien de probant	Rien	Rien
52EE	Oui	Rien	Code Intéressant en 52F8 (au-dessus)
53F6	Rien de probant	Rien	Beaucoup trop d'appel, et ne correspond pas
54D2	Rien de probant	Rien	Rien de probant
54F6	Rien de probant	Rien de probant	Rien de probant
6128	Rien de probant	Rien de probant	Rien de probant
70DA	Rien de probant	Rien de probant	Rien
76EE	Rien de probant	Rien de probant	Rien de probant
7B66	Rien de probant	Rien de probant	Rien de probant
8312	Rien de probant	Rien de probant	Rien de probant
9C66	Rien de probant	Rien	Rien
9D14	Rien de probant	Rien de probant	Rien de probant
A022	Rien de probant	Rien de probant	Rien de probant
AF56	Rien de probant	Beaucoup trop d'appel, et ne correspond pas	
A126	Rien de probant	Rien de probant	Rien
A2A4	Rien de probant	Rien de probant	Code trop long, ne correspond pas
A3A6	Rien de probant	Rien de probant	Rien
AF56	Rien de probant	Beaucoup trop d'appel, et ne correspond pas	
BA78	Intéressant mais boucle trop petite		
BDD6	Rien de probant	Rien de probant	Beaucoup trop d'appel, et ne correspond pas
CA72	Rien de probant	Rien de probant	Code trop long, ne correspond pas
CB60	Rien de probant	Rien de probant	Rien de probant
CE08	Rien de probant	Rien	Beaucoup trop d'appel, et ne correspond pas
DD8C	Rien de probant	Beaucoup trop d'appel, et ne correspond pas	
F5B4	Rien de probant	Rien de probant	Code Intéressant en F564 (au-dessus)

On revient donc sur les adresses probables préalablement trouvées : **4EDE, 52EE, 52F8** et **F5B4**

```
=========================================
4ED8    MOVEM.L D0-D1/A0,-(A7)          ; ➔
4EDC    MOVEQ   #3,D0                   ;
4EDE    BSR     518A                    ;
4EE2    MOVEM.L (A7)+,D0-D1/A0          ;
EDD6    ROL.W   #1,D0                   ;
4EE8    ADD.W   (A0)+,D0                ; intéressant, incrémentation de D0 avec (A0)
4EEA    DBF     D1,4ED8                 ; ← Boucle de travail avec (A0)+ pour incrémenter D0
4EEE    ADD.L   D0,10B4(A6)             ; Fin de la boucle, mise à jour d'un marqueur en 10B4(A6), 1er CRC ?
4EF2    BRA     5148                    ; GoTo 5148
=========================================
5148    MOVEQ   #0,D0                   ; Raz de D0
514A    MOVE.W  114(A5),D0              ;
514E    MOVEA.L A5,A0                   ; On retrouve le même type de code que précédemment.
5150    MOVEA.L (A5),A5                 ;
5152    MOVEA.L E(A5),A7                ; Modification de A7 avec E(A5)
...
=========================================
```

```
=========================================
52EA    MOVEM.L D0-D1/A0,-(A7)          ; ➔
52EE    BSR     513C                    ;
52F2    MOVEM.L (A7)+,D0-D7/A0          ;
52F6    ADD.W   (A0)+,D0                ; intéressant, incrémentation de D0 avec (A0)
52F8    DBF     D7,52EA                 ; ← Boucle de travail avec (A0)+ pour incrémenter D0
52FC    ADD.L   D0,4(A5)                ; Fin de la boucle, mise à jour d'un marqueur en 4(A5), 2em CRC ?
5300    BRA     5148                    ; GoTo 5148
=========================================
5148    MOVEQ   #0,D0                   ; Raz de D0
514A    MOVE.W  114(A5),D0              ;
514E    MOVEA.L A5,A0                   ; On retrouve le même type de code que précédemment.
5150    MOVEA.L (A5),A5                 ;
5152    MOVEA.L E(A5),A7                ; Modification de A7 avec E(A5)
...
=========================================
```

```
=========================================
F552    ROL.W   #1,D1                   ; ➔
F554    ADD.W   (A0)+,D1                ; intéressant, incrémentation de D1 avec (A0)
F556    DBF     D0,F552                 ; ← Boucle de travail avec (A0)+ pour incrémenter D1
F55A    EXT.L   D1                      ;
F55C    ADD.L   D1,14A0(A6)             ; Fin de la boucle, mise à jour d'un marqueur en 14A0(A6), 3em CRC ?
...
F590    BRA     F5B4                    ; GoTo F5B4
...
F5B4    JSR     513C                    ; GoSub 513C
=========================================
513C    MOVE.L  A7,E(A5)                ;
5140    MOVEA.L (A5),A5                 ; Déjà vu plus haut,
5142    MOVEA.L E(A5),A7                ; Modification de A7 avec E(A5)
5146    RTS                             ;
=========================================
```

Il nous reste plus qu'à tester/patcher ces 3 routines.

Part 12 tentative de crack

La routine de 'crash', de 'saut', semble assez complexe ou plus exactement bien mélangée dans le code.
Le plus simple pour nous et afin de gagner du temps serait de la désactiver directement depuis ces 3 routines, désactiver la mise à jour de ses 'marqueurs'.

```
4EEE    ADD.L   D0,10B4(A6)
52FC    ADD.L   D0,4(A5)
F55C    ADD.L   D1,14A0(A6)
```

On va essayer en supprimant tout simplement ses instructions.
Taper :

```
A 4EEE
$004EEE   NOP
$004EF0   NOP
$004EF2   <RETURN>

A 52FC
$0052FC   NOP
$0052FE   NOP
$005300   <RETURN>
```

Taper :

```
A F55C
$00F55C   NOP
$00F55E   NOP
$00F560   <RETURN>
```

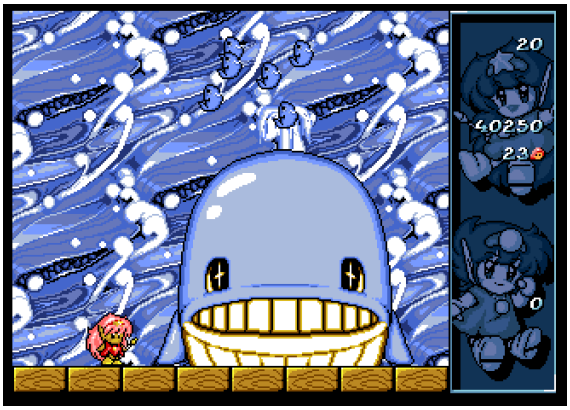
Et on retourne au code avec la commande `X`
Le patch semble fonctionner puisque le jeu continu son chargement et nous arrivons sur la page du menu.
Lancez une partie et testez le jeu.
Avec les vies infinies, ça sera plus simple 😊

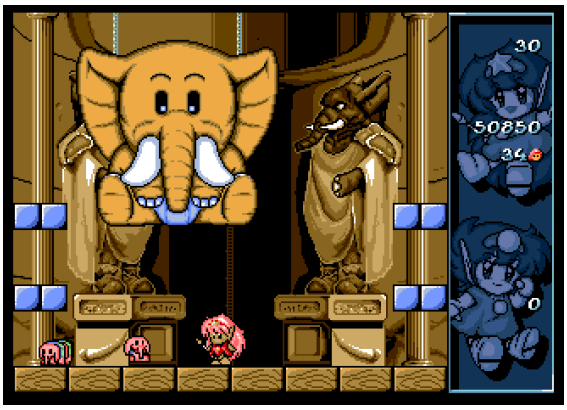
Retourner dans l'AR et **Taper :**

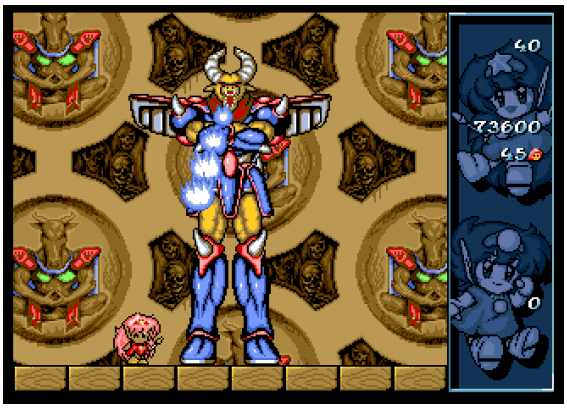
```
A B360
$00B360   TST.W 44(A4)
$00B364   <RETURN>
```

Sans oublier de retourner au code avec la commande `X`

Le jeu semble fonctionner jusqu'au bout et sans crash ou bug.







Part 13 Test de notre crack

Avec les adresses : 4EEE, 52FC, F55C, on est clairement dans la zone copiée en *$7FF00-$7FF0C* , juste avant le **BRA $100**
Juste avant l'appel à **CopyLock_Base** et c'est tant mieux, on va pouvoir patcher la protection CRC et le *CopyLock* en même temps.
Il nous reste plus qu'à trouver une zone mémoire libre pour créer et sauver notre bout de code.

On sait que, voir plus haut, du code est copié à partir de l'adresse mémoire **$100**, on va donc essayer d'éviter cette zone.
Mais quand est t'il de la zone mémoire juste avant ?

Rebooter votre Amiga **avec la disquette originale** dans le lecteur.
Laisser le *TrackLoad* commencer mais **juste avant d'atteindre la 9em Piste, entrer dans l'AR.**

Taper : `M 0` et descendez un peu dans la zone mémoire, jusqu'à maximum **$100**

```
M 0
:000000 00 00 00 00 00 00 06 76 00 FC 08 18 00 FC 08 1A .......v.ü...ü..
:000010 00 FC 08 1C 00 FC 08 1E 00 FC 08 20 00 FC 08 22 .ü...ü...ü. .ü."
:000020 00 FC 09 0E 00 FC 08 26 00 FC 08 28 00 FC 08 2A .ü...ü.&.ü.(.ü.*
:000030 00 FC 08 2C 00 FC 08 2E 00 FC 08 30 00 FC 08 32 .ü.,.ü...ü.0.ü.2
:000040 00 FC 08 34 00 FC 08 34 00 FC 08 34 00 FC 08 34 .ü.4.ü.4.ü.4.ü.4
:000050 00 FC 08 34 00 FC 08 34 00 FC 08 34 00 FC 08 34 .ü.4.ü.4.ü.4.ü.4
:000060 00 FC 08 34 00 FC 0C 8E 00 FC 0C E2 00 FC 0D 14 .ü.4.ü...ü...ü..
:000070 00 FC 0D 6C 00 FC 0D FA 00 FC 0E 40 00 FC 0E 86 .ü.l.ü...ü.@.ü..
:000080 00 FC 08 36 00 FC 08 38 00 FC 08 3A 00 FC 08 3C .ü.6.ü.8.ü.:.ü.<
:000090 00 FC 08 3E 00 FC 08 40 00 FC 08 42 00 FC 08 44 .ü.>.ü.@.ü.B.ü.D
:0000A0 00 FC 08 46 00 FC 08 48 00 FC 08 4A 00 FC 08 4C .ü.F.ü.H.ü.J.ü.L
:0000B0 00 FC 08 4E 00 FC 08 50 00 FC 08 52 00 FC 08 54 .ü.N.ü.P.ü.R.ü.T
:0000C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
:0000D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
:0000E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
:0000F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
:000100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
:000110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

On peut apercevoir une zone libre à partir de **$C0.** Reste à valider qu'elle le reste tout le temps.

#O, alias Fill mémory. Permet de remplir une zone mémoire avec une valeur.
Taper : `O "LiBrE ", C0 100`

```
o "LiBrE", C0 100
Ready.
M C0-10
:0000B0 00 FC 08 4E 00 FC 08 50 00 FC 08 52 00 FC 08 54 .ü.N.ü.P.ü.R.ü.T
:0000C0 4C 69 42 72 45 4C 69 42 72 45 4C 69 42 72 45 4C LiBrELiBrELiBrEL
:0000D0 69 42 72 45 4C 69 42 72 45 4C 69 42 72 45 4C 69 iBrELiBrELiBrELi
:0000E0 42 72 45 4C 69 42 72 45 4C 69 42 72 45 4C 69 42 BrELiBrELiBrELiB
:0000F0 72 45 4C 69 42 72 45 4C 69 42 72 45 4C 69 42 72 rELiBrELiBrELiBr
:000100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

Et on retourne au chargement du jeu avec la commande `X`
Une fois sur le menu principal du jeu, entrer dans l'AR, on va jetez un coup d'œil en mémoire si notre zone à changée.

Entrer dans l'AR.

Taper : `M C0-10` et descendez dans la zone mémoire, jusqu'à **$100**

```
M C0-10
:0000B0 00 FC 08 4E 00 FC 08 50 00 FC 08 52 00 FC 08 54 .ü.N.ü.P.ü.R.ü.T
:0000C0 4C 69 42 72 45 4C 69 42 72 45 4C 69 42 72 45 4C LiBrELiBrELiBrEL
:0000D0 69 42 72 45 4C 69 42 72 45 4C 69 42 72 45 4C 69 iBrELiBrELiBrELi
:0000E0 42 72 45 4C 69 42 72 45 4C 69 42 72 45 4C 69 42 BrELiBrELiBrELiB
:0000F0 72 45 4C 69 42 72 45 4C 69 42 72 45 4C 69 42 72 rELiBrELiBrELiBr
:000100 60 00 39 6C FF E0 FF FF FF FF 80 00 FF FF BF FF `.9l............
```

Impeccable, notre zone n'a pas changé, on peut donc s'en servir pour créer notre bout de code.

Rebooter votre Amiga **avec la disquette de Backup** dans le lecteur.
Laisser le *TrackLoad* commencer mais **juste avant d'atteindre la 9em Piste, entrer dans l'AR** et poser un *BreakPoint* en $A68C

Taper : `BS A68C` puis, on retourne au code avec la commande `X`
Rapidement notre *BreakPoint* en $A68C est atteint, **on entre automatiquement dans l'AR.**
Les données sont maintenant chargées puis décompressées et le code copié en *$7FF00*

Taper : `BS 7FF0C` puis, on retourne au code avec la commande `X`
Rapidement notre *BreakPoint* en $7FF0C est atteint, **on entre automatiquement dans l'AR.**
Le code est maintenant copiée en *$100* et on s'apprête à 'sauter' vers `#CopyLock_Base`

Part 14 Mise en place de notre crack

On changer tout ça.

Taper : A C0

```
^0000C0         BSR     4F0C                ; ← On reprend le code d'origine présent en $3A6E, #CopyLock_Base
^0000C4         BSR     4AF2                ; ← Juste avant le code 'réel' de CopyLock en $3A86
^0000C8         MOVEQ   #FF,D0              ; ←
^0000CA         MOVE.L  D0,6A(A6)           ; ←
^0000CE         MOVE.L  D0,6E(A6)           ; ←
^0000D2         MOVEQ   #0,D0               ; ←
^0000D4         MOVEQ   #0,D1               ; ←
^0000D6         MOVEQ   #0,D3               ; ←
                                            ;
^0000D8         SUB.L   43B0.S,D0           ; Ici on désactive le futur ADDI.L #51C4631D,D0
```

En effet, en sortie de la routine *CopyLock* la clé lue présente en **D0** est copiée en mémoire *$3A6A*

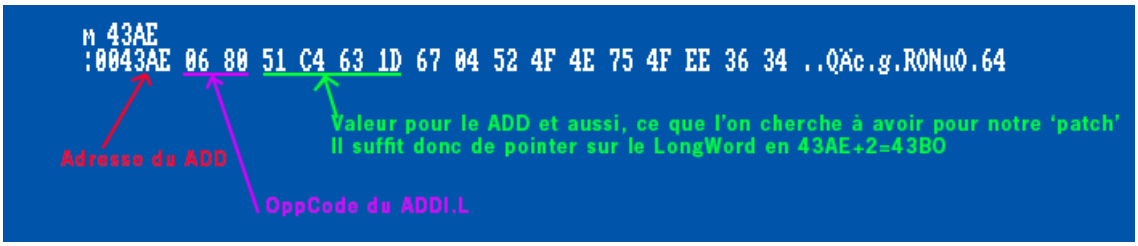
```
43AA      MOVE.L   D0,3A6A.S
```

Un traitement est alors effectué sur **D0** afin d'obtenir 'normalement' un résultat=0 afin de pouvoir effectuer le Saut.

```
43AE      ADDI.L   #51C4631D,D0
43B4      BEQ      43BA
```

Pour obtenir un résultat à 0, il suffit tout simplement **d'annuler ce ADD**. D'effectuer l'instruction inverse juste avant. A savoir **SUB.L #51C4631D,D0** mais on peut faire plus simple et surtout plus court au niveau de l'**OPCODE** car, n'oublions pas, nous sommes limité en mémoire pour notre 'code' de *$C0 à $100*

La valeur **51C4631D** est déjà en mémoire... si ! regarder bien, elle est dans le **ADD en 43AE**
Il suffit donc de pointer sur elle :




```
^0000DC         MOVE.L  D0,60.S             ; On copie la clé en mémoire $60 car CopyLock V2
                                            ;
                                            ; 4E71 est l'OPCODE d'un NOP
^0000E0         MOVE.L  #4E714E71,4EEE.S    ; On met 2 NOP en 4EEE   (.s pour gagner 1 word en mémoire)
^0000E8         MOVE.L  #4E714E71,52FC.S    ; On met 2 NOP en 52FC   (.s pour gagner 1 word en mémoire)
^0000F0         MOVE.L  #4E714E71,F55C      ; On met 2 NOP en F55C
                                            ; Ce qui va désactiver l'ensemble des CRC préalablement trouvés.
                                            ;
^0000FA         BRA     43AA                ; Et ensuite, on branche sur le code d'origine pour continuer le d'job.
^0000FE         <RETURN>
```

Rappel :

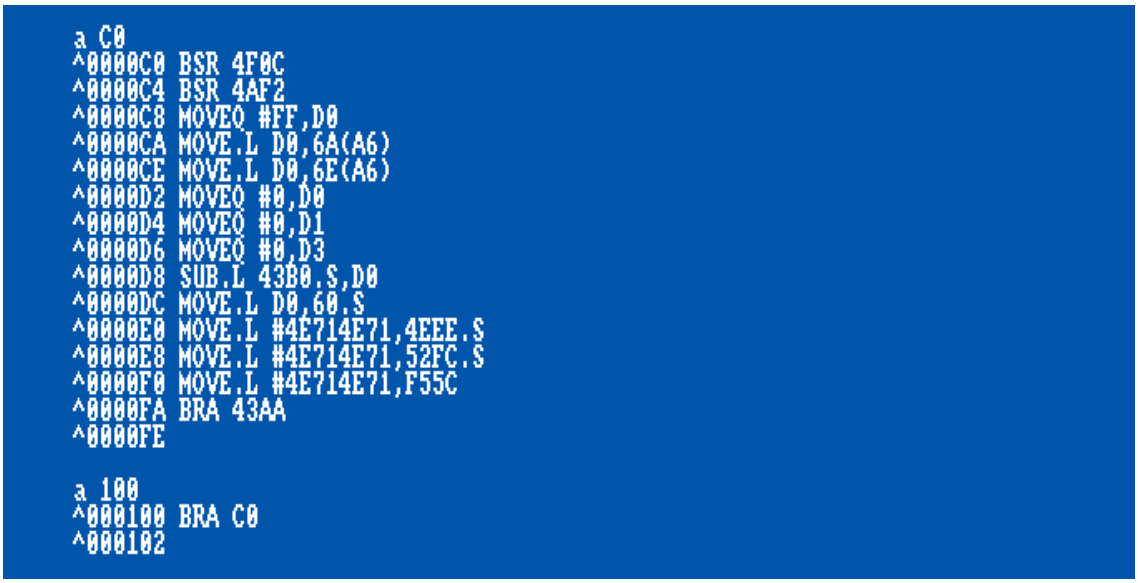
```
============================
...
43A8      MOVEA.W  -(A7),A2          ; ...
43AA      MOVE.L   D0,3A6A.S         ; Copy de D0 à l'adresse mémoire $3A6A
43AE      ADDI.L   #51C4631D,D0      ; On ajoute 51C4631D à D1
43B4      BEQ      43BA              ; Si résultat = 0 alors GoTo ➔ $43BA
43B6      ADDQ.W   #1,A7             ; Ajoute 1 a la pile, ce qui donnera une adresse impaire au retour
                                     ; et donc à coup sûr un crash de l'amiga lors du prochain RTS
43B8      RTS                        ;
============================         ; #Start_Base_Game
43BA      LEA      3634(A6),A7       ;
43BE      MOVE.W   #20,DFF09A        ; Conf POTINP
...
```

Sans oublier de modifier aussi le saut effectué à l'adresse mémoire $100, il faut le remplacer vers notre patch.

Taper : A 100

```
^000100 BRA     C0              ; Saute vers notre petit bout de code, notre 'patch'
^000102 <RETURN>
```

Une jolie capture d'écran :



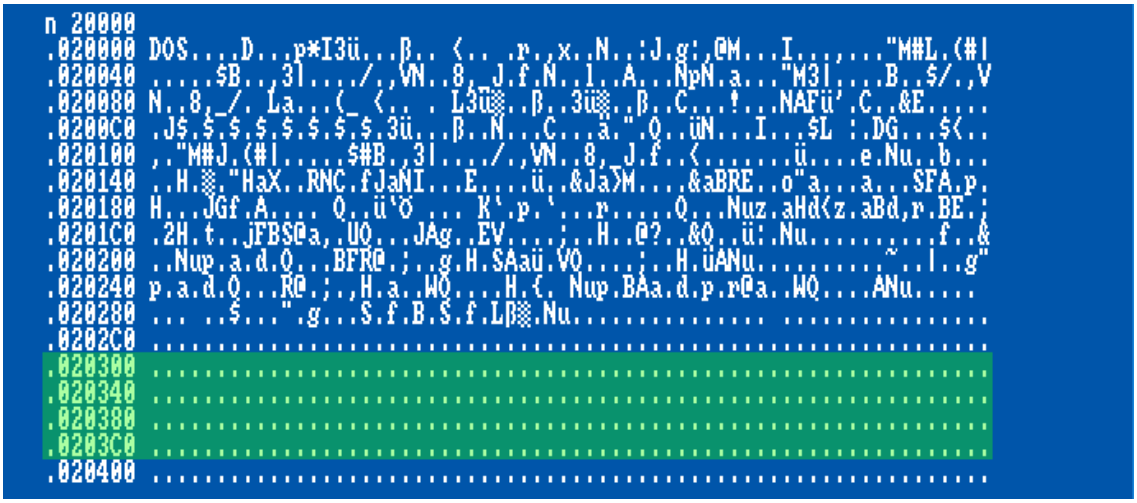
Et on retourne au code avec la commande **x**

Le jeu continu son chargement sans planter, on peut lancer une partie et jouer. Aucun bug ou crash à l'horizon.
Il nous reste plus qu'à rendre la modification permanente.

Part 15 Let's Crack it

Toujours avec la disquette de backup dans le lecteur, on recharge le *BootSecteur* et regarder l'occupation mémoire du code.
#N, alias memory read. Permet de voir/écrire les données en mémoire au format uniquement ASCII.

Taper : `RT 0 1 20000` puis `N 20000`



Si la disquette est bootable, l'amiga va charger le bootsecteur en mémoire (taille 1024 octets donc $400 en hexa)
On peut voir clairement une grosse zone libre de *$202C0 jusqu'à $20400*.

On va se laisser une petite marge et commencer notre code en *$20300*.
On commence par la section '**patch**'

Petit rappel des bouts de code d'origine recopiés en **$C0** (les sauts), voir section précédente si besoin.

```
=======================                 ; #CopyLock_Base
C0      BSR     4F0C                    ; BSR travail tjrs en relatif donc 4F0C-C0=4E4C
C4      BSR     4AF2                    ; BSR travail tjrs en relatif donc 4AF2-C4=4A2E
```

et le début du code en sortie de la routine *CopyLock*

```
FA      BRA     43AA                    ; BRA travail tjrs en relatif donc 43AA-FA = 42B0
```

Taper : `A 20340`

```
^20340          BSR     2518C           ; $20340 + 4E4C = 2518C
^20344          BSR     24D72           ; $20344 + 4A2E = 24D72
^20348          MOVEQ   #FF,D0          ; Code d'origine
^2034A          MOVE.L  D0,6A(A6)       ; Code d'origine
^2034E          MOVE.L  D0,6E(A6)       ; Code d'origine
^20352          MOVEQ   #0,D0           ; Code d'origine
^20354          MOVEQ   #0,D1           ; Code d'origine
^20356          MOVEQ   #0,D3           ; Code d'origine
^20358          SUB.L   43B0.S,D0       ; Voir explication chapitre précédent.
^2035C          MOVE.L  D0,60.S         ;
^20360          MOVE.L  #4E714E71,4EEE.S ;
^20368          MOVE.L  #4E714E71,52FC.S ;
^20370          MOVE.L  #4E714E71,F55C  ;
^2037A          BRA     2462A           ; $2037A + 42B0 = 2462A
^2037E          <RETURN>
```

Longueur de notre patch = $2037E-$20340=$**3E**
Et on s'attaque maintenant à la section '**boucle de copy de notre patch en $C0**'

Taper : `A 20300`

```
^20300          MOVEM.L D0/A0-A1,-(A7)  ; On sauvegarde les registres.
^20304          LEA     20340(PC),A0    ; A0=$20340     'Adr. relative du début de notre patch'
^20308          MOVEQ   #3D,D0          ; D0=$3D                Longueur à copier-1
^2030A          LEA     C0.S,A1         ; A1=$C0                Adr de destination
^2030E          MOVE.B  (A0)+,(A1)+     ; ➔ Boucle de copie de notre patch vers $C0
^20310          DBF     D0,2030E        ; ←
^20314          MOVEM.L (A7)+,D0/A0-A1  ; On restaure les registres.
^20318          RTS                     ; E.T Retour maison
^2031A          <RETURN>
```

Taper : `A 2000E`

```
^20010          BSR     20300           ; E.T Retour maison
^20014          <RETURN>
```

On recherche maintenant le fameux **JMP 100**

Taper : **FA 100 20000**

```
02000DE         LEA     100.S,A1                ; Nop, c'est pas celui la
02000EA         JMP     100.S                   ; Et voici celui que l'on recherche en 200EA
```

Taper : **A 200EA**

```
^0200EA         JMP     C0.S                    ; On le modifie pour sauter vers notre 'patch' copié en $C0
^0200EE         <RETURN>
```

Sans oublier de sauter aussi au début du code du *bootsecteur* vers notre routine :

Taper : **A 20010**

```
^020010         BSR     20300                   ; On Saute vers notre boucle de re-copie du code vers $C0
^020014         <RETURN>
```

On recalcul le checksum de notre code final

BOOTCHK 20000

Et on écrit le nouveau *bootblock*

WT 0 1 20000

Rebooter ensuite votre Amiga normalement et apprécier le jeu 😉