Tutoriel	**AmigaCracking : Arkanoid – Revenge of Doh**
Protection	**Single Track Protection - Copylock**
Auteur Original	**Rob**
Soumit par (sur flashtro.com)	**Rob [2005-01-16]**
Version	**01/02/2018 Gi@nts**
Vérification/Correction	**V2, Testé et fonctionnel de A à Z**

ARKANOID – REVENGE OF DOH
* CRACK TUTORIEL *

Table des matières

Matériels nécessaire

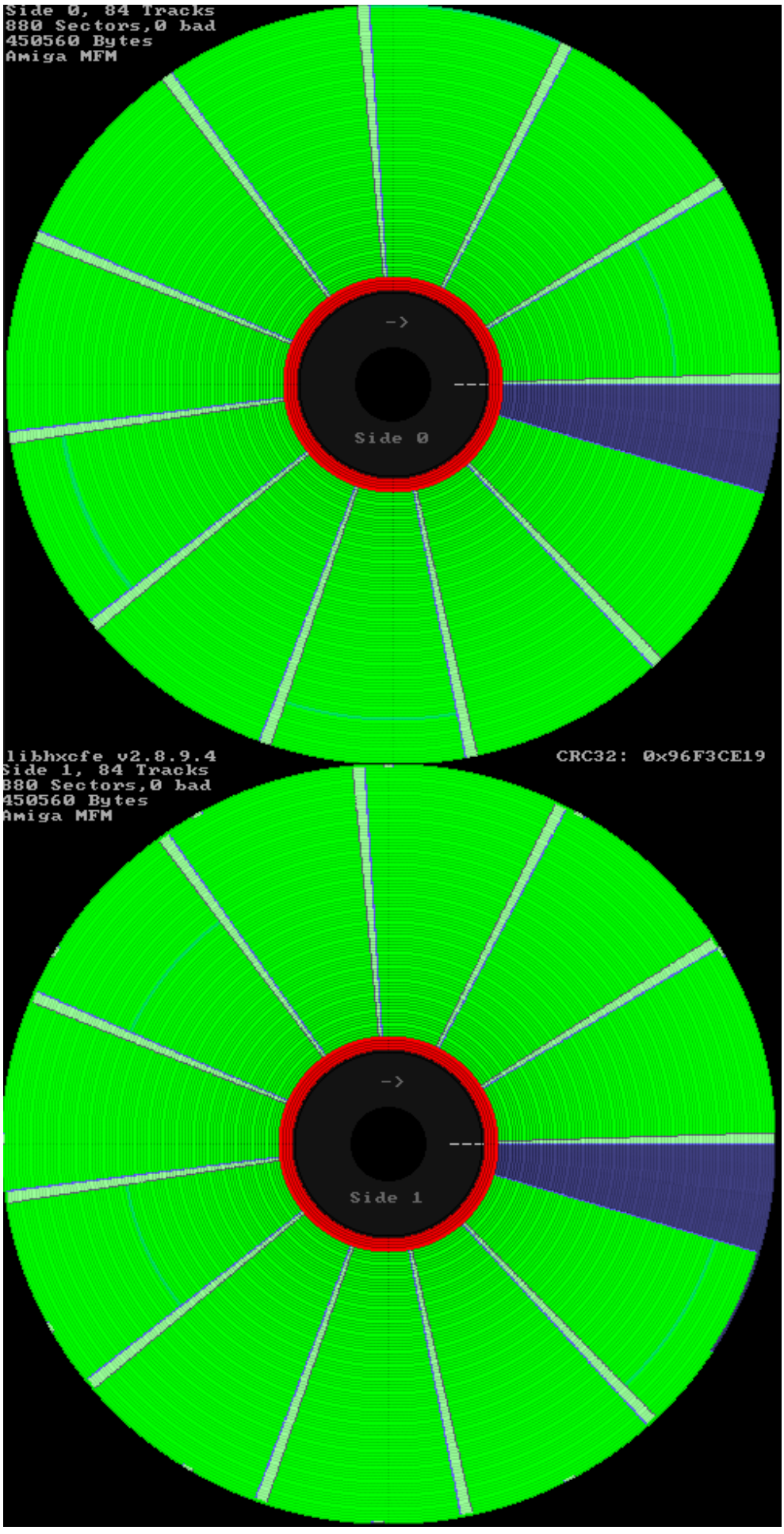
1) Un Amiga avec 512K (ou plus) ou l'émulateur WINUAE
2) Une Carte ACTION REPLAY MKIII (ou ça ROM Image)
3) Le jeu Original Arkanoid – Revenge Of Doh ou son image CAPS (SPS 0765)
4) Le logiciel Xcopy Pro en disquette ou image disk.

Général Info

Ce tutoriel Français est basé sur le tutoriel original de Rob.
Ce document n'est pas une traduction mot par mot de celui-ci mais plus une nouvelle version.
Suivit pas à pas avec des nouvelles informations.
Noter que les adresses mémoire indiquée peuvent changer en fonction de votre configuration H/W

Un grand merci à **WayneK** sur le forum de **flashtro.com** qui m'a bien aidé sur les questions **que je me posais et du coup** a permis la création de ce tuto.

Bon Tuto. **Gi@nts**

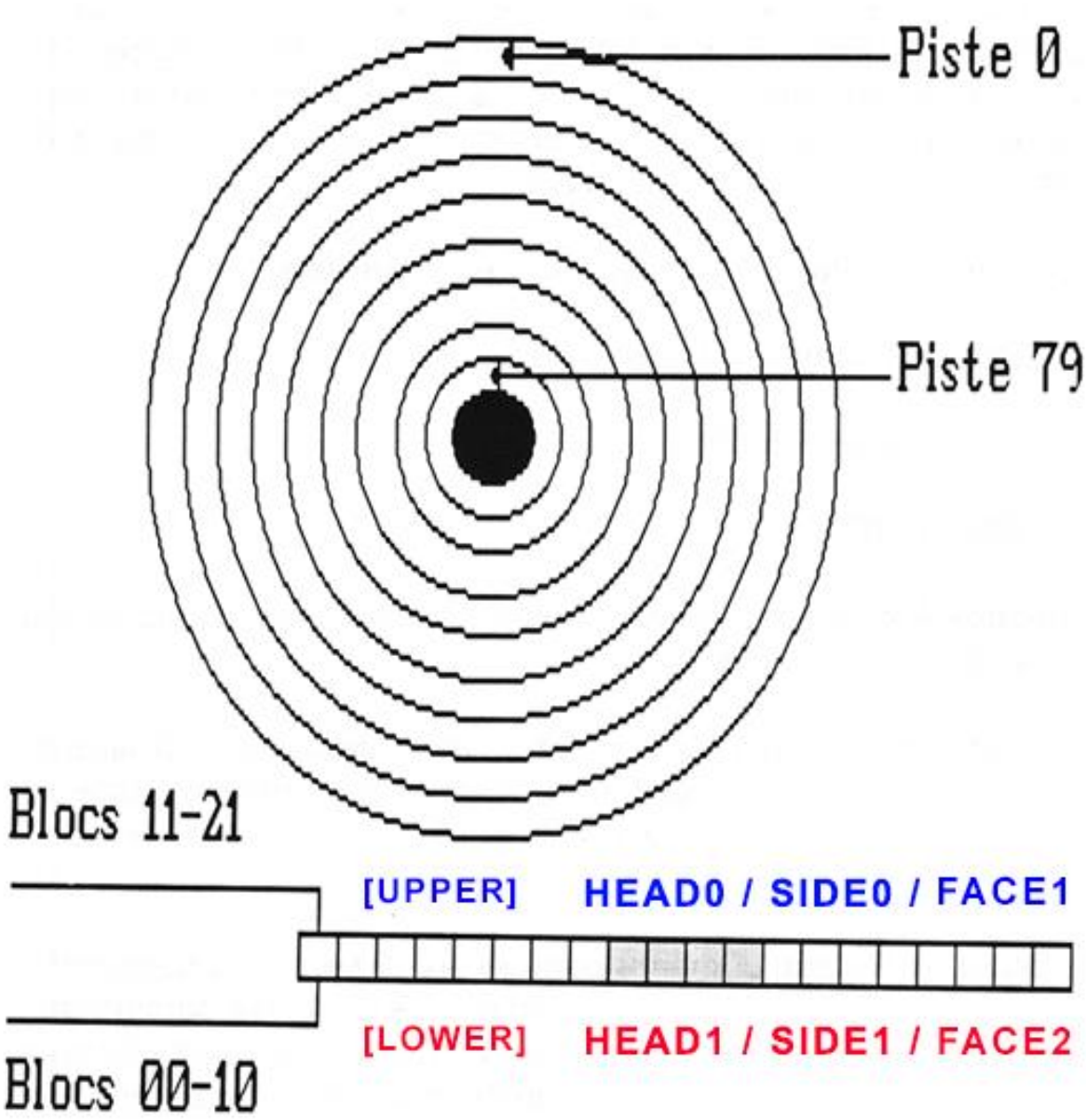

Side 0, 84 Tracks
880 Sectors,0 bad
450560 Bytes
Amiga MFM
->
Side 0
libhxcfe v2.8.9.4
Side 1, 84 Tracks
880 Sectors,0 bad
450560 Bytes
Amiga MFM
CRC32: 0x96F3CE19
->
Side 1

Agencement des disquettes Amiga v1.1

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).

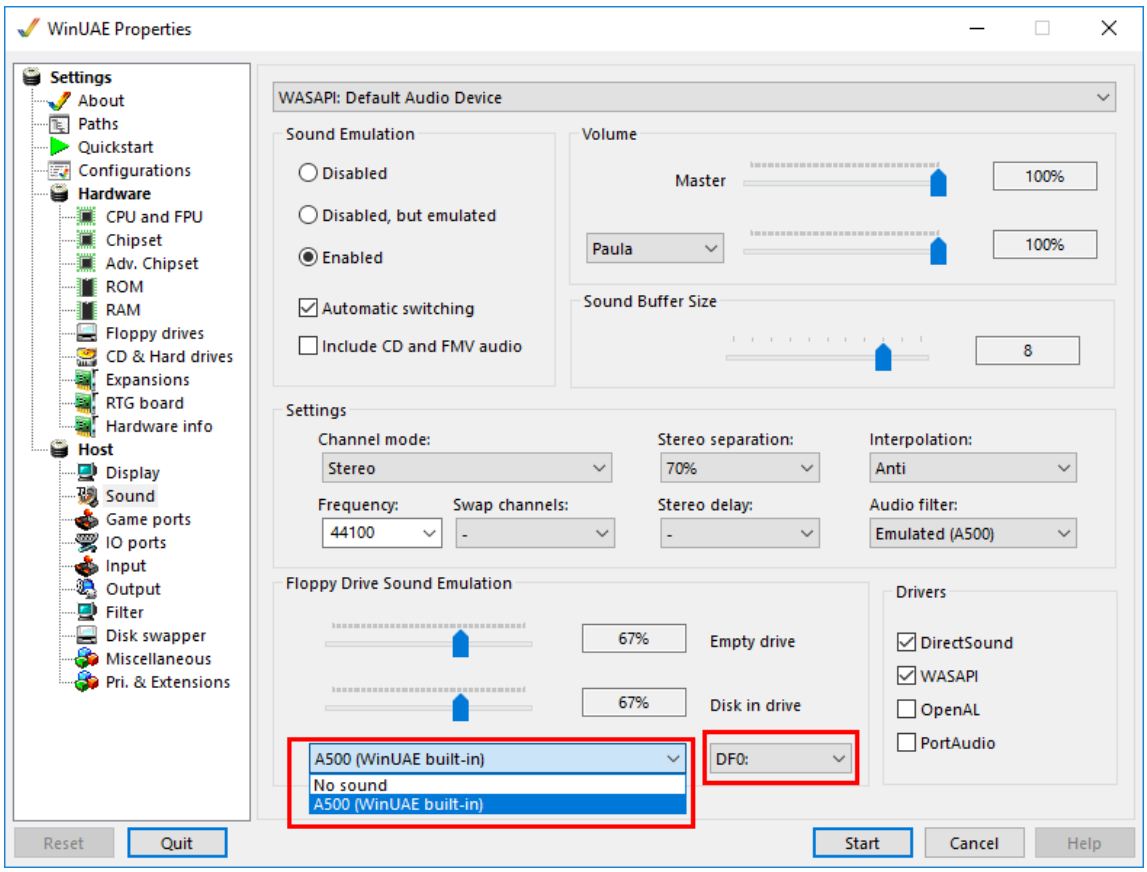
$20/2=$10 = !16 (en décimal donc) et comme il n'y a pas de retenu on est sur la face0.

```
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

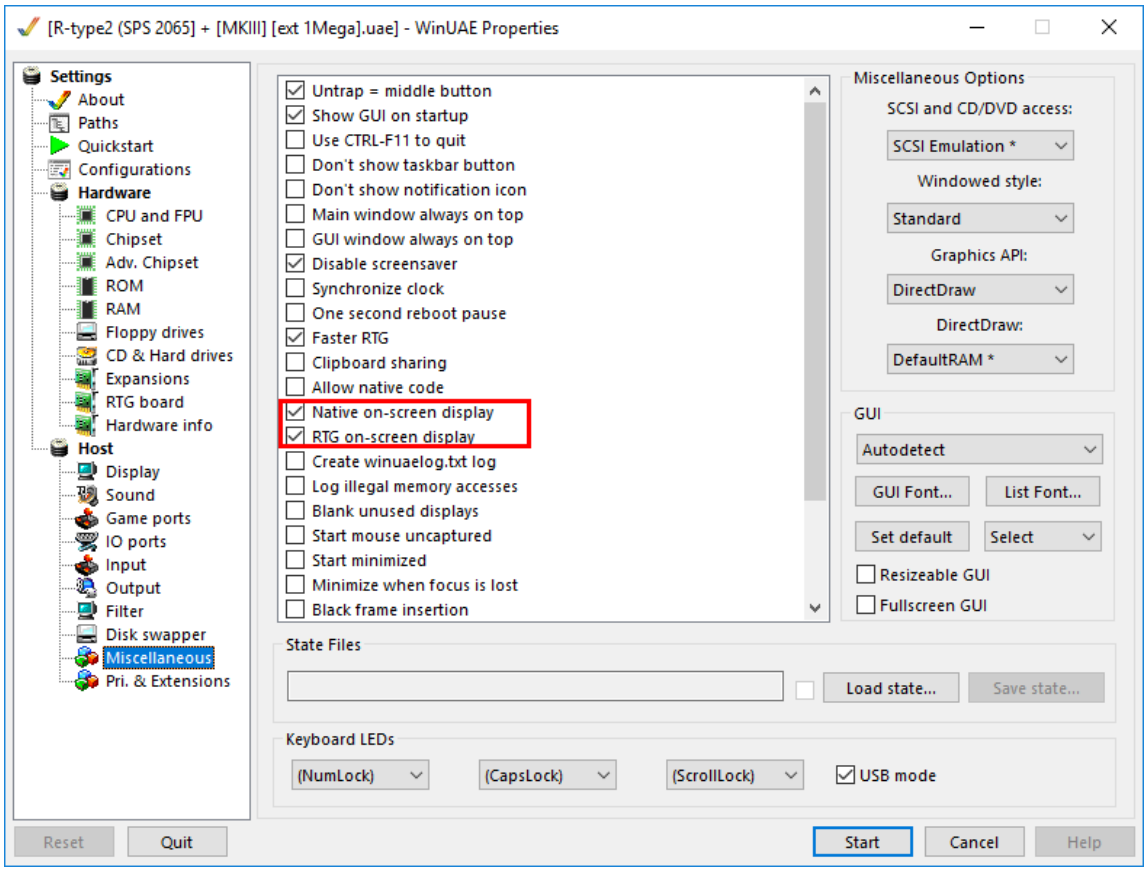
21 est donné en hexa **donc $21 = !33** en décimal.
33/2 = 16.5 ,donc piste 16 side 1 et comme on continue à lire/copier les donnes (**taille à lire =2**), on continue la copie.
On change donc de track car on est déjà sur la face 1 (il existe que 2 faces sur une disquette)
On arrive donc sur la prochaine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

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

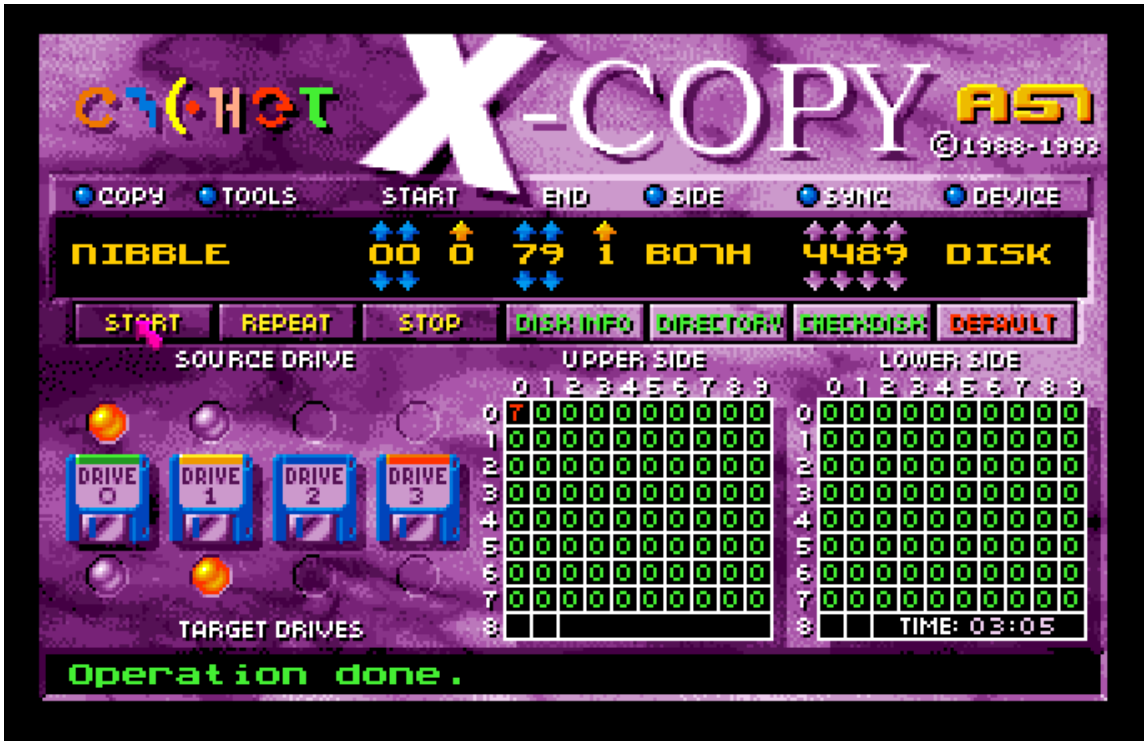


Voir même, pour plus d'information. Par exemple afficher sur qu'elle face l'on se trouve, d'activer :
Host -> Miscellaneous -> Native on-screen display AND RTG on-screen display



Part 1 X-Copy

Comme dans Tous bon hack qui se respecte, on va commencer par essayer de copier le disk original.
Démarrer votre logiciel de copie préféré, à savoir **Xcopy Pro**
Choisissez le mode **NIBBLE, insérer** une disquette **vierge** en **DF1** et la disquette du **jeu original** en **DF0**
Lancer la copie



Hummm, cela ressemble à une protection '**copylock**'

Bien sûr, cette copie ne fonctionnera pas et/ou fera très vite crasher votre Amiga. (Vous pouvez tester)
Mais gardons quand même ce ***backup.***

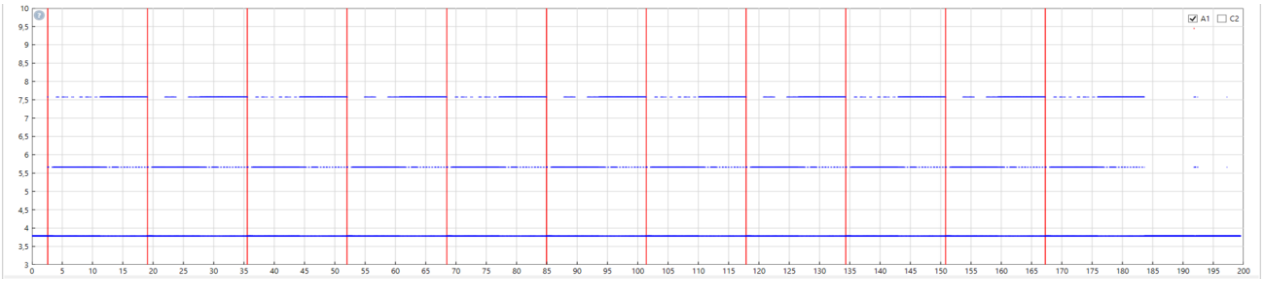
Rappel des codes d'erreur de Xcopy :

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	0765_ArkanoidRevengeOfDoh_AMIGA.ipf
TYPE	Floppy_Disk
ENCODER	CAPS(V1)
FILE	765(V1)
DISK	0
TRACK	00-83
SIDE	0-1
PLATFORM	Amiga
REVOLUTION	4
PROTECTION	Copylock [T00.1]

Mon script d'analyse nous indique la présence d'une protection **Copylock**
D'ailleurs si on regarde la piste 00.1 de plus prêt.



TrackNumber	Size Record (bytes)	Crc	Status	Track Size	Detail Tr. Size	Start Byte	Bit	DataKey	Block	Density	Signal	Encoder	Flag
T00.0	80	10A75675	Good	12551	100408 bits = Data=95744 + Gap=4664	274	2198	001	11	Auto	cell_2us	0	None
T00.1	80	BF87596E	Good	13182	105450 bits = Data=96144 + Gap=9306	672	5378	002	12	Auto	cell_2us	0	None
T01.0	80	1358067A	Good	12551	100408 bits = Data=95744 + Gap=4664	274	2197	003	11	Auto	cell_2us	0	None
T01.1	80	7B62FF16	Good	12550	100400 bits = Data=95744 + Gap=4656	272	2182	004	11	Auto	cell_2us	0	None
T02.0	80	01E5B86D	Good	12551	100408 bits = Data=95744 + Gap=4664	274	2196	005	11	Auto	cell_2us	0	None
T02.1	80	4171BB36	Good	12550	100400 bits = Data=95744 + Gap=4656	272	2182	006	11	Auto	cell_2us	0	None
T03.0	80	0944CF61	Good	12551	100408 bits = Data=95744 + Gap=4664	274	2197	007	11	Auto	cell_2us	0	None
T03.1	80	0659E4EF	Good	12551	100408 bits = Data=95744 + Gap=4664	273	2187	008	11	Auto	cell_2us	0	None
T04.0	80	9C0A568F	Good	12550	100400 bits = Data=95744 + Gap=4656	274	2193	009	11	Auto	cell_2us	0	None
T04.1	80	B1395636	Good	12551	100408 bits = Data=95744 + Gap=4664	273	2184	010	11	Auto	cell_2us	0	None
T05.0	80	CF7604ED	Good	12551	100408 bits = Data=95744 + Gap=4664	274	2196	011	11	Auto	cell_2us	0	None
T05.1	80	8CD4B208	Good	12551	100408 bits = Data=95744 + Gap=4664	273	2190	012	11	Auto	cell_2us	0	None
T06.0	80	B06CB918	Good	12551	100408 bits = Data=95744 + Gap=4664	275	2206	013	11	Auto	cell_2us	0	None
T06.1	80	96019EA2	Good	12550	100400 bits = Data=95744 + Gap=4656	272	2181	014	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				1FF196E4			13576-20021
#0	10	8704	545	512	0	1	MFM	0352	0352	15	13576-13607
#1	0	8704	545	512	0	1	MFM	0906	0906	57	13608-13639
#2	1	8704	545	512	0	1	MFM	1460	1460	92	13640-13671
#3	2	8704	545	512	0	1	MFM	2014	2014	126	13672-13703
#4	3	8704	545	512	0	1	MFM	2568	2568	161	13704-13735
#5	4	8704	545	512	0	1	MFM	3122	3122	196	13736-13767
#6	5	8704	545	512	0	1	MFM	3676	3676	230	13768-13799
#7	6	8704	545	512	0	1	MFM	4230	4230	265	13800-13831
#8	7	8704	545	512	0	1	MFM	4784	4784	300	13832-13863
#9	8	8704	545	512	0	1	MFM	5338	5338	334	13864-13895
#10	9	8704	545	512	4664	292	MFM	5892	5892	369	13896-13927
[T00.1]		6516		52128				9DC429DC			20050-26565
#0		8704	545		0	1	MFM	0384	0384	25	20050-20081
#1	0	8704	545	512	0	1	MFM	0938	0938	59	20082-20113
#2	1	8704	545	512	0	1	MFM	1492	1492	94	20114-20145
#3	2	8704	545	512	0	1	MFM	2046	2046	128	20146-20177
#4	3	8704	545	512	0	1	MFM	2600	2600	163	20178-20209
#5	4	8704	545	512	0	1	MFM	3154	3154	198	20210-20241
#6	5	8704	545	512	0	1	MFM	3708	3708	232	20242-20273
#7	6	8704	545	512	0	1	MFM	4262	4262	267	20274-20305
#8	7	8704	545	512	0	1	MFM	4816	4816	302	20306-20337
#9	8	8704	545	512	0	1	MFM	5370	5370	336	20338-20369
#10	9	8704	545	512	4282	268	MFM	5924	5924	371	20370-20401
#11	10	400	26	512	5024	315	MFM	6478	6478	405	20402-20433
[T01.0]		6446		51568				9369DD3C			26594-33039
#0	4	8704	545	512	0	1	MFM	0352	0352	15	26594-26625
#1	5	8704	545	512	0	1	MFM	0906	0906	57	26626-26657
#2	6	8704	545	512	0	1	MFM	1460	1460	92	26658-26689

On peut voir que 12 'block' sont détectées en T00.1 alors que normalement on a 11 block sur un disk AmigaDOS.
Que la taille de la piste est plus importante que les autres à savoir **6516 bytes** (logique vue que l'on est sur une piste plus longue), alors que l'on est sur **6446 bytes** sur toutes les autres pistes.

Part 3 Let's do it

Insérez la disquette originale du jeu dans le lecteur de l'Amiga et démarrer dessus.

Attendez le chargement de quelques pistes (2s) puis,
Entrer dans votre **AR** et allons regarder si on trouve notre **'copylock'** en mémoire :
#F, alias Find, permet de chercher une valeur hexa ou ascii en mémoire
#Ici on cherche les variables $48 $7A qui correspondent en général au début d'une signature de copylock
Taper `F 48 7A`

Aucun résultat.... Humm

On va faire autrement, à savoir : charger le **bootblock** en mémoire et regarder ça de plus près :

#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`

```
d 10000
~010000 NEG.W   A7
~010002 SUBQ.B  #1,D0
~010004 CHK     A5,D4
~010006 MOVE.W  USP,-(A6)
~010008 ORI.B   #70,D0
~01000C MOVEM.L D0-D1/A0-A2,-(A7)
~010010 MOVE.L  #E00,D0
~010016 MOVE.L  #10002,D1
~01001C MOVEA.L 00000004.S,A6
~010020 JSR     -C6(A6)
~010024 TST.L   D0
~010026 BEQ     00010010
~010028 MOVE.L  D0,10(A7)
~01002C MOVEA.L C(A7),A1
~010030 MOVE.L  10(A7),D0
~010034 MOVE.L  D0,28(A1)
~010038 MOVE.L  #E00,24(A1)
~010040 MOVE.L  #400,2C(A1)
~010048 MOVE.W  #2,1C(A1)
~01004E MOVEA.L 00000004.S,A6
~010052 JSR     -1C8(A6)
~010056 MOVEA.L C(A7),A1
~01005A MOVE.B  1F(A1),D0
~01005E BNE     00010030
~010060 MOVEM.L (A7)+,D0-D1/A0-A1
~010064 RTS
;======================================
```

Il semblerait qu'on ait notre petite routine de déplacement de donnée ici :

010034 MOVE.L D0,28(A1) <== Adresse de destination en **A1**
010038 MOVE.L #E00,24(A1) <== Nbr de donnée à copier : **$E00**
010048 MOVE.L #400.2C(A1) <== Adresse source de lecture : **$400**

Cela va donc copier **$E00** de donnée à partir de **$400** vers l'adresse contenue dans **D0**

On va modifier notre petit **'bootblock'** pour éviter le **RTS** en **$10064** histoire d'en savoir plus.
Taper :
#A, alias Assemble, Instruction qui va permettre de taper du code assembleur.
#BRA, Instruction du 68000 qui permet de se brancher à l'adresse indiqué, ici une boucle sur nous-même.
#BOOTCHK, permet de calculer le checksum d'un bootblock en mémoire
#WT, alias Write Track, permet d'écrire une zone mémoire sur la disquette à l'adresse indiqué en cylindre.

 # On modifie le code à partir de 10064
`A 10064`

#On change le RTS par un BRA sur soit même, la fameuse boucle infini.
`BRA    10064` [**ENTRER**] puis [**ESCAPE**]
#On calcule le nouveau checksum de ce bootblock
`BOOTCHK 10000`

#On sauve le tout sur disquette
/!\ Insérez la disquette de **backup** préalablement crée et taper :
`WT 0 1 10000`

Effectuer un **reboot** de votre Amiga **tout en laissant la disquette de backup** dans celui-ci.

Après un court chargement (celui du **bootblock**), l'Amiga semble ne plus rien faire.
Il est entré dans notre boucle sans fin.

Entrer dans votre **AR**
#D, alias Désassemble, Instruction qui va permettre de désassembler du code assembleur.
Taper : `D`

~005CA4 BRA 00005CA4

On est bien dans notre boucle sans fin, impeccable.
On va voir si maintenant on arrive à trouver notre petite signature de '***copylock***' en mémoire.
Taper `F 48 7A`

```
d
~005CA4 BRA     00005CA4

f 48 7A
Search from: 000000 to: 080000
00A520  _
```

Bingo, trouvé !

On jette un petit coup d'œil en mémoire à l'adresse indiquer :
#n alias memory read, permet de voir les données en mémoire en ASCII.
Taper `n A520`

```
n A520
.00A520 Hz..#ß... @..l .].\.#.... ?.NhC..X".L.......A...H...A..d#.....Jü
.00A560 ü..[...[...[...$..'=.Ä'..Ä.+...=..'.... '.....4...0'...'/'../.\.§
.00A5A0 .X.q.....F.............Ö..H...A..4#....$A...#.... ........../....
.00A5E0 .o....C..&g. Q ...'&.I..H...C... Q ... o.."#P.. (.üF.H@..Lß..Ns
.00A620 ........................D.ß.o...t!..\j.....$..;...Ye...l..r.ü.
.00A660 ..ü.ü....E....L.....\...)Ö..ovÖ..ö..Ö..?Fc..f.Cw.Bf.Ö....n(o...=
.00A6A0 8..*.$.....6.pK..K".cK8...ö/.6..... ....}{.b.....ü....J.bä.. cü_
.00A6E0 .LC..Ö... ).A......5....ök.b.Yö..%Ue....eemeQe.E..ü(.%.e...Ä....
```

Bon…Rien de bien percutant.
On va simplement continuer l'exécution du **bootblock**
Cela passe bien sûr par la suppression de notre modification.

#RTS est une fonction assembleur qui permet le retour d'un sous-programme.
Taper `A 5CA4` puis `RTS` [**ENTRER**] puis [**ESCAPE**]

Et **avant** de retourner à l'exécution de notre code, on **remplace** la **disquette de backup** par la **disquette originale** de notre **jeu** dans le lecteur Amiga, puis **taper** la commande : `X`
#X permet le retour au code Amiga en court.

```
a 5CA4
^005CA4 rts
^005CA6
```

Le code continue son chemin et charge les données du jeu.
Assez rapidement, une image du jeu est affichée.

/!\ ➔ Entrer sans attendre dans votre AR

On va aller voir si les données ont changé à l'adresse **$5A20**
Taper `n A520`

```
n A520  [[ Avant ]]
.00A520 Hz..#ß... @..I .].\.#....?.NhC..X".L.......A...H...A..d#.....Jü
.00A560 ü..[...[...[...S..'=.A'..A.+...=..'...'......4...0'...'/'../.\.§
.00A5A0 .X.q.....F..............Ó..H...A..4#....$A...#.... ........./....
.00A5E0 .o....C..&g. Q ...'&.I..H...C... Q .... o.."#P.. (.üF.H@..LB..Ns
.00A620 ..........................D.ß.o...t!..\j.....$..;...Ye...l..r.ü.
.00A660 ..ü.ü....E....L....\...)O..ovÖ..ö..Ö..?Fc..f.Cw.Bf.Ö....n(o...=
.00A6A0 8..*.$.....6.pK..K".cK8...Ö/.6..... ....}{.b.....ü....J.bä.. cü_
.00A6E0 .LC..Ö... ).A......5....ök.b.Yö..%Ue....eemeQe.E..ü(.%.e...Ä....



n A520  [[ Maintenant ]]
.00A520 ..3ü...ß..By.ß..3üB..ß.. |..▒.0(.."I.ß..2.Q..ü"M#I.....(#I.....$
.00A560 #I.....,3I....,x..N..8"M.)..f.3I....#I.....$,x..N..83ü...ß..3ü.
.00A5A0 .ß..#..X...13ü. .ß..0(.. |.ß..BXQ..üN......9...ß..g(#ü.....ß..#ü
.00A5E0 ...@.ß.ä#ü.....ß..#ü.....ß..N..ü.....................................
.00A620 .........................................................................
.00A660 .........................................................................
.00A6A0 .........................................................................
```

Il semblerait qu'il y ait eu du changement.
Le '***copylock***' a sûrement décodé les données.

Taper `D 5A20`
Remonter au début de cette 'partie' de code, à savoir : **$A49E**

Le code juste au-dessus nous envoie en **$A490**, qui nous renvoie rapidement en **$A4A4**

```
~00A48E MOVE.L  -5556(A1),(A5)
~00A492 LINEA
~00A494 ORI.B   #0,D0
~00A498 BRA     0000A4A4
;=====================================
~00A49A SUBQ.B  #1,D0
~00A49C BRA     0000A490
;=====================================
~00A49E LINEF
~00A4A0 ORI.B   #70,D0
~00A4A4 MOVEA.L A1,A5
~00A4A6 MOVE.L  #77FDE,28(A1)
~00A4AE MOVE.L  #7E00,24(A1)
~00A4B6 MOVE.L  #2C00,2C(A1)
~00A4BE MOVE.W  #2,1C(A1)
~00A4C4 MOVEA.L 00000004.S,A6
~00A4C8 JSR     -1C8(A6)
~00A4CC MOVEA.L A5,A1
~00A4CE MOVE.B  1F(A1),D0
~00A4D2 BNE     0000A4A6
~00A4D4 MOVE.W  #A0,00DFF096
~00A4DC MOVE.W  #20,00DFF09A
~00A4E4 MOVE.W  #8100,00DFF096
~00A4EC LEA     A5FC(PC),A0
~00A4F0 MOVE.L  0000006C,2(A0)
```

$A498 semble être une bonne adresse de départ, beaucoup de chose se passe tout de suite après cette adresse.
Allons regarder ce qu'il y a exactement en mémoire à cet endroit.
Taper `N A498`

```
n A498
.00A498 '.S.'..C...p*I#I..▒..(#I..~..$#I.....,3I....,x..N..8"M.)..f.3ü..
.00A4D8 .ß..3ü. .ß..3ü...ß..A...!y...l..A...#....13ü. .ß..3ü(..ß..3ü...ß
.00A518 ..3ü.0.ß..3ü...ß..By.ß..3üB..ß.. |..▒.0(.."I.ß..2.Q..ü"M#I.....(
.00A558 #I.....$#I.....,3I....,x..N..8"M.)..f.3I....#I.....$,x..N..83ü..
.00A598 .ß..3ü. .ß..#..X...13ü. .ß..0(.. |.ß..BXQ..üN......9...ß..g(#ü..
.00A5D8 ...ß..#ü...@.ß.ä#ü.....ß..#ü.....ß..N..ü.....................
.00A618 .........................................................
```

Le code semble se terminer vers **$A618**

Chose confirmé avec la commande `M A498`

```
:00A5D8 80 00 00 DF F0 E0 23 FC 00 07 9F 40 00 DF F0 E4 ...ß..#ü...@.ß.ä
:00A5E8 23 FC 00 07 BE 80 00 DF F0 E8 23 FC 00 07 DD C0 #ü.....ß..#ü....
:00A5F8 00 DF F0 EC 4E F9 00 FC 0D 14 00 00 00 00 00 00 .ß..N..ü........
:00A608 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
:00A618 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
:00A628 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
:00A638 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
:00A648 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
:00A658 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
:00A668 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

Le code semble même se terminer quelque part vers **$A601** pour être plus précis.

Comme on a pu le remarquer plus haut, dans le code du '***bootblock***', des données étaient accédé à l'adresse **$400**

En fait, dans le cas d'une protection '***copylock***', cet adresse mémoire sert de tampon a ce qui est décodé.
En l'occurrence ici, la track 0

Maintenant qu'on a ses données décodées en mémoire, on va tout simplement les déplacer ses données décryptées vers cette adresse et réécrire les données sur disque.

Taper :
#On lie et charge en mémoire notre bootblock
`RT 0 2 70000`

#TRANS, alias transfert. Permet le transfert d'une zone mémoire vers une autre.
#On transfère nos datas décryptée vers la zone tampon en question (70000+400)
`TRANS A498 A618 70400`

Remplace la **disquette du jeu original** par la **disquette de BACKUP** dans le lecteur Amiga, puis :

Taper :
#On sauve le tout sur disquette,
#WT alias Write Track, permet l'écriture d'une piste à partir d'une zone mémoire.
`WT 0 2 70000`

Maintenant je le jeu devrait être fonctionnel et surtout copiable simplement avec xcopy
Redémarrer votre Amiga et apprécier ce superbe jeu 😉