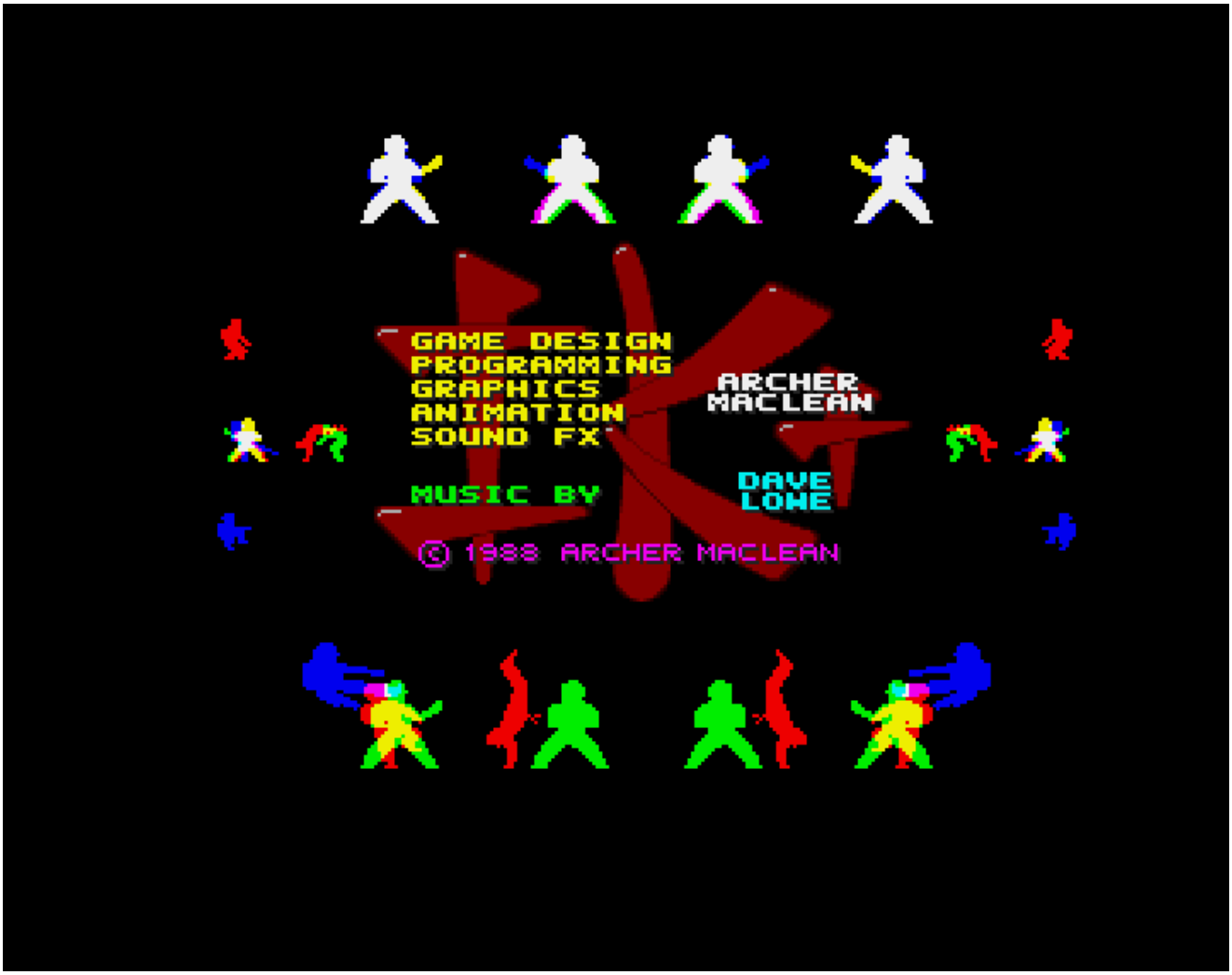


Tutoriel	**AmigaCracking : MFM : IK +**
Protection	**MFM (multi file)**
Auteur Original	**Rob**
Soumit par (sur flashtro.com)	**Musashi9 [2015-09-08]**
Version	**17/02/2017 Gi@nts**
Vérification/Correction	**V2, Testé et fonctionnel de A à Z**

* IK+CRACK TUTORIEL *

Table des matières

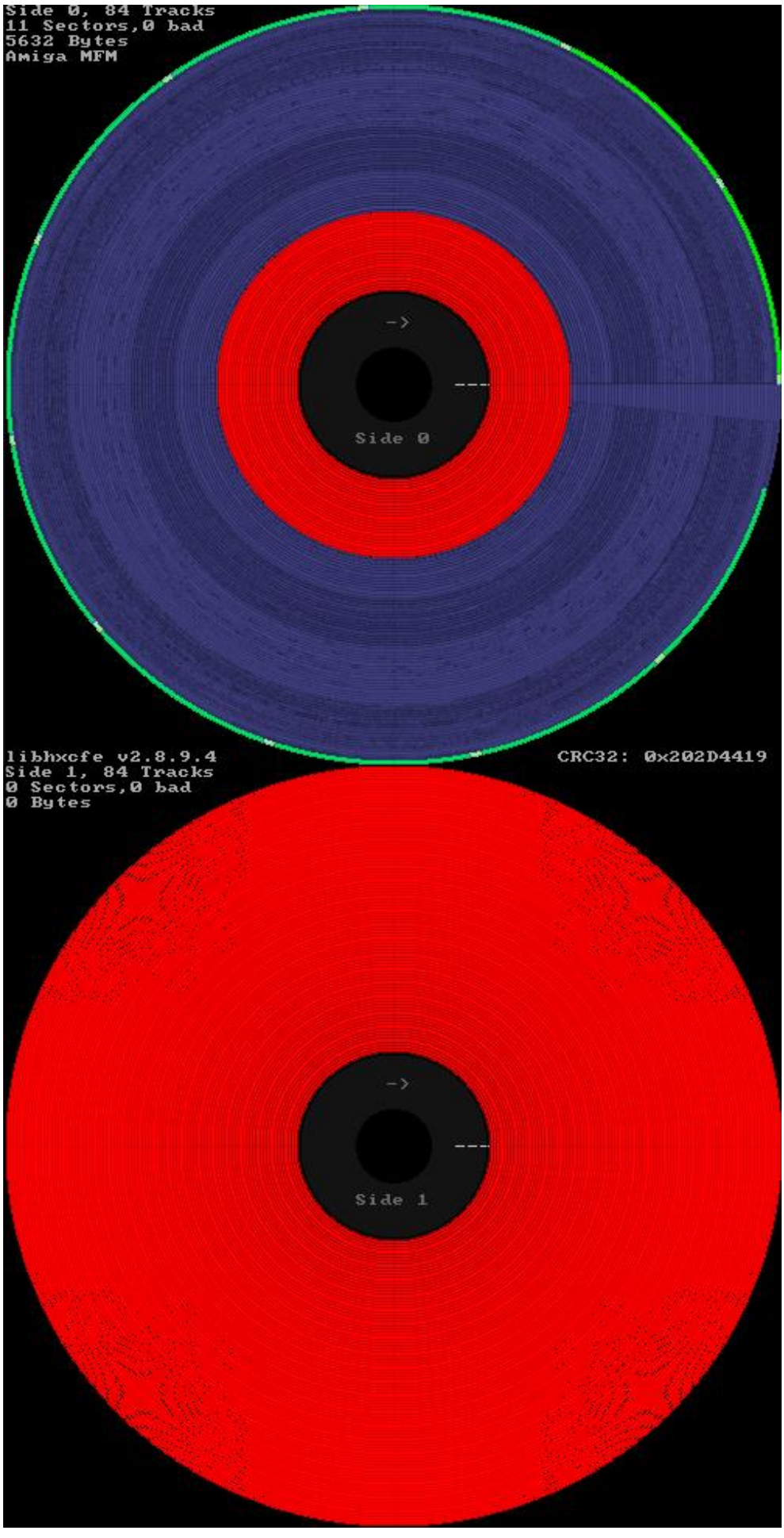
Matériels nécessaire

1) Un AMiGA avec 512K (ou plus) ou l'émulateur WINUAE
2) Une Carte ACTION REPLAY MKIII (ou sa ROM Image)
3) Le jeu Original IK+ ou son image CAPS (SPS 0339)
4) Le logiciel de compression 'Double Action' de VINCE of TRISTAR (exemple compilation : Anarchy-ScratchPack29)

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.

Bon Tuto. **Gi@nts**


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

Side 0
libhxcfe v2.8.9.4
Side 1, 84 Tracks
0 Sectors,0 bad
0 Bytes
CRC32: 0x202D4419
->

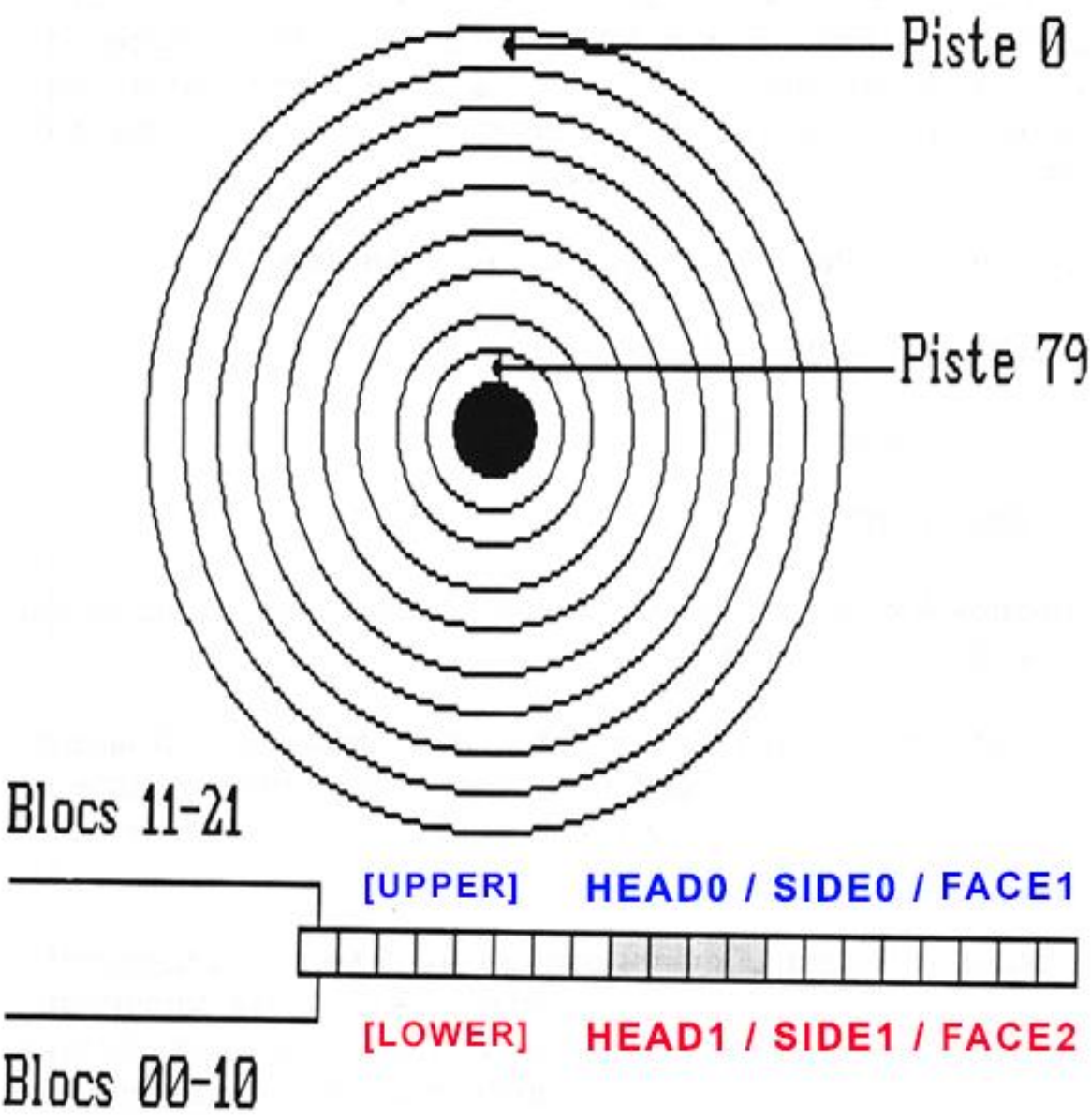
Side 1

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).

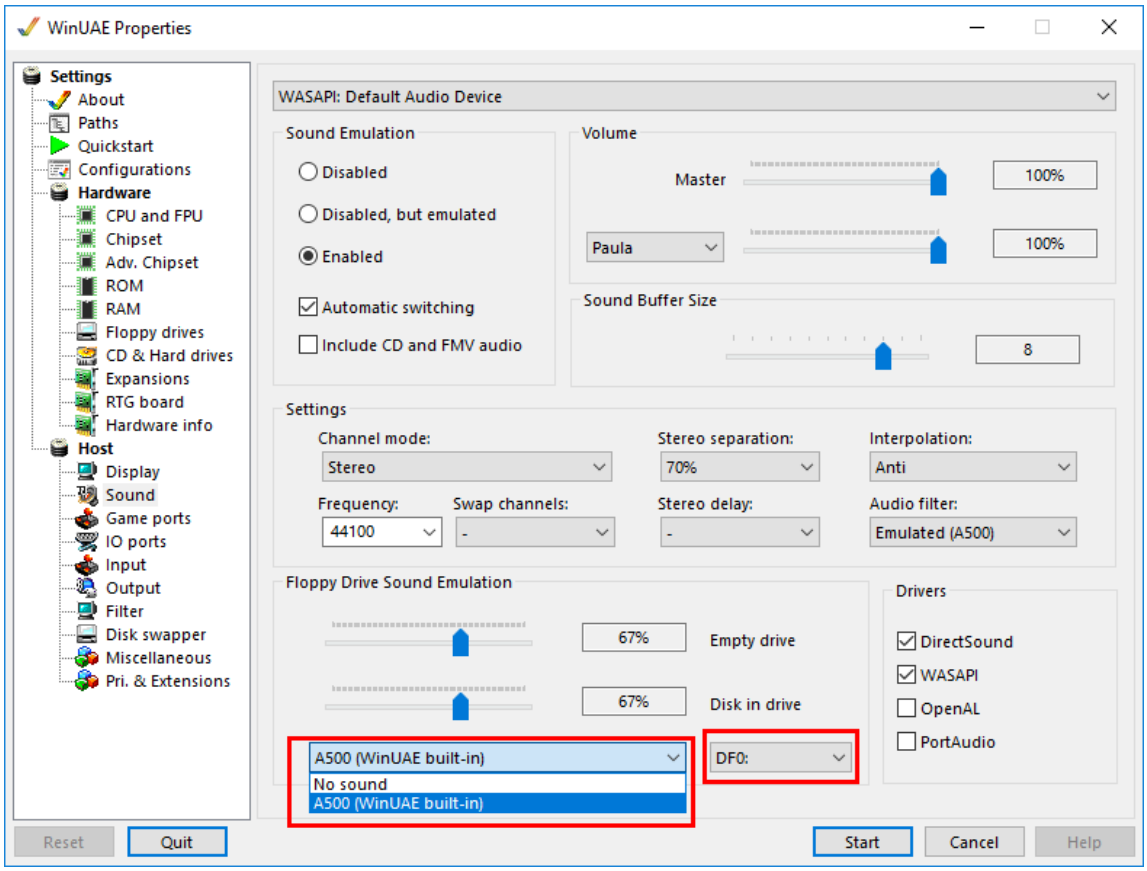
$20/2=$10 = !16 (en décimal donc) et comme il n'y a pas de retenu on est sur la face0.

```
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

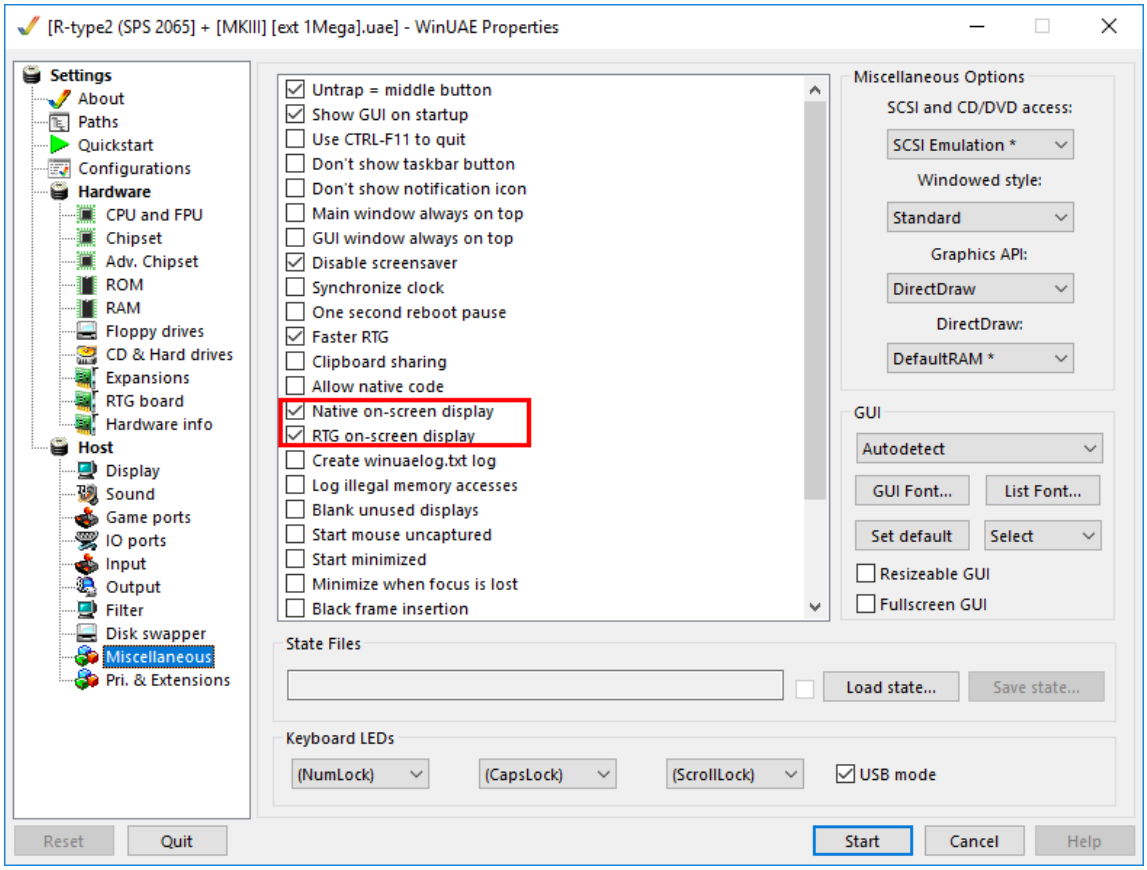
21 est donné en hexa **donc $21 = !33** en décimal.
33/2 = 16.5 ,donc piste 16 side 1 et comme on continue à lire/copier les donnes (**taille à lire =2**), on continue la copie.
On change donc de track car on est déjà sur la face 1 (il existe que 2 faces sur une disquette)
On arrive donc sur la prochaine 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.
Entrer dans votre **AR**
et **Taper :** `burst`

Choisissez une unité de destination contenant une disquette vierge et **lancer** la copie.

Ou si vous préférez passer par X-Copy

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*

Devinez quoi... Ce n'est pas vraiment copiable en l'état.

Part 2 Analyse de l'image IPF

FILENAME	0339_IKplus_AMIGA.ipf
TYPE	Floppy_Disk
ENCODER	CAPS(V1)
FILE	339(V1)
DISK	0
TRACK	00-83
SIDE	0-1
PLATFORM	Amiga
REVOLUTION	4

Au premier coup d'œil sur l'image disque ainsi que sous X-Copy et burstnibbler, on peut voir que la 1er piste est standard et qu'ensuite vient un format qui lui ne l'est pas.
On peut aussi souligner, car ce n'est pas courant sur Amiga, que les données sont présentes uniquement sur une seule face.

Maintenant si on regarde en détail cette image ipf, on peut voir qu'effectivement, de la piste 1 à 60, on est bien sur un format propriétaire, non **AmigaDOS**

```
unknown MFM address mark (AC) at offset 75185 on cyl 2 head 0
unknown MFM address mark (A4) at offset 83305 on cyl 1 head 0
unknown MFM address mark (A0) at offset 69122 on cyl 3 head 0
unknown MFM address mark (A0) at offset 62092 on cyl 4 head 0
unknown MFM address mark (A0) at offset 54073 on cyl 5 head 0
unknown MFM address mark (A0) at offset 55036 on cyl 6 head 0
unknown MFM address mark (A1) at offset 48022 on cyl 7 head 0
unknown MFM address mark (A0) at offset 41005 on cyl 8 head 0
unknown MFM address mark (A0) at offset 25004 on cyl 9 head 0
unknown MFM address mark (AF) at offset 17993 on cyl 10 head 0
unknown MFM address mark (A0) at offset 9084 on cyl 11 head 0
unknown MFM address mark (AF) at offset 968 on cyl 12 head 0
unknown MFM address mark (A0) at offset 94039 on cyl 13 head 0
unknown MFM address mark (A8) at offset 79046 on cyl 14 head 0
unknown MFM address mark (A4) at offset 70037 on cyl 15 head 0
unknown MFM address mark (A1) at offset 63028 on cyl 16 head 0
unknown MFM address mark (A8) at offset 56010 on cyl 17 head 0
unknown MFM address mark (A0) at offset 41018 on cyl 18 head 0
unknown MFM address mark (A5) at offset 9193 on cyl 19 head 0
unknown MFM address mark (A0) at offset 6172 on cyl 20 head 0
unknown MFM address mark (A0) at offset 97243 on cyl 21 head 0
unknown MFM address mark (A0) at offset 82249 on cyl 22 head 0
unknown MFM address mark (A0) at offset 81237 on cyl 23 head 0
unknown MFM address mark (A0) at offset 70358 on cyl 24 head 0
unknown MFM address mark (A0) at offset 70336 on cyl 25 head 0
```

En **bleu foncé** sur l'image ci-dessous on peut voir qu'effectivement, aucune donnée n'est présente sur la face 1 'Upper' (champs à zéro)
En **violet**, aucune donnée n'est présente après la **piste T60.0**

T55.0	80	2091B6B0	Good	12510	100079 bits = Data=98416 + Gap=1663	698	5586	111	1	Auto	cell_2us	0	None
T55.1	80	4B123ECD	Good	0	0 bits = Data=0 + Gap=0	0	0	112	0	Noise	cell_2us	0	None
T56.0	80	C29FBBC9	Good	12510	100074 bits = Data=98416 + Gap=1658	12125	97004	113	1	Auto	cell_2us	0	None
T56.1	80	C34FB900	Good	0	0 bits = Data=0 + Gap=0	0	0	114	0	Noise	cell_2us	0	None
T57.0	80	14B7301A	Good	12510	100075 bits = Data=98416 + Gap=1659	12247	97979	115	1	Auto	cell_2us	0	None
T57.1	80	E340343B	Good	0	0 bits = Data=0 + Gap=0	0	0	116	0	Noise	cell_2us	0	None
T58.0	80	BDA7169C	Good	12510	100075 bits = Data=98416 + Gap=1659	12370	98960	117	1	Auto	cell_2us	0	None
T58.1	80	D953701B	Good	0	0 bits = Data=0 + Gap=0	0	0	118	0	Noise	cell_2us	0	None
T59.0	80	52C3869B	Good	12510	100076 bits = Data=98416 + Gap=1660	11371	90968	119	1	Auto	cell_2us	0	None
T59.1	80	A35F2E4D	Good	0	0 bits = Data=0 + Gap=0	0	0	120	0	Noise	cell_2us	0	None
T60.0	80	CF8248B5	Good	12510	100076 bits = Data=98416 + Gap=1660	12368	98945	121	1	Auto	cell_2us	0	None
T60.1	80	F7762B36	Good	0	0 bits = Data=0 + Gap=0	0	0	122	0	Noise	cell_2us	0	None
T61.0	80	D7C7ED6A	Good	0	0 bits = Data=0 + Gap=0	0	0	123	0	Noise	cell_2us	0	None
T61.1	80	D779A60D	Good	0	0 bits = Data=0 + Gap=0	0	0	124	0	Noise	cell_2us	0	None
T62.0	80	2D6DC3DC	Good	0	0 bits = Data=0 + Gap=0	0	0	125	0	Noise	cell_2us	0	None
T62.1	80	ED6AE22D	Good	0	0 bits = Data=0 + Gap=0	0	0	126	0	Noise	cell_2us	0	None
T63.0	80	CDDB2471	Good	0	0 bits = Data=0 + Gap=0	0	0	127	0	Noise	cell_2us	0	None
T63.1	80	8DF8E5E8	Good	0	0 bits = Data=0 + Gap=0	0	0	128	0	Noise	cell_2us	0	None
T64.0	80	341B4C56	Good	0	0 bits = Data=0 + Gap=0	0	0	129	0	Noise	cell_2us	0	None
T64.1	80	F41C6DA7	Good	0	0 bits = Data=0 + Gap=0	0	0	130	0	Noise	cell_2us	0	None

Sur l'image ci-dessous, on peut voir en **T00.0** un format standard **AmigaDOS** comme prévu.
Qu'effectivement, aucune donnée n'est présente sur toute les pistes **TXX.1**
Et que l'on a une taille de piste de 6194 sur toutes les autres pistes qui suivent T00.0

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				3BA1DB70			13576-20021
#0		8704	545		0	1	MFM	0352	0352	15	13576-13607
#1	1	8704	545	512	0	1	MFM	0906	0906	57	13608-13639
#2	2	8704	545	512	0	1	MFM	1460	1460	92	13640-13671
#3	3	8704	545	512	0	1	MFM	2014	2014	126	13672-13703
#4	4	8704	545	512	0	1	MFM	2568	2568	161	13704-13735
#5	5	8704	545	512	0	1	MFM	3122	3122	196	13736-13767
#6	6	8704	545	512	0	1	MFM	3676	3676	230	13768-13799
#7	7	8704	545	512	0	1	MFM	4230	4230	265	13800-13831
#8	8	8704	545	512	0	1	MFM	4784	4784	300	13832-13863
#9	9	8704	545	512	0	1	MFM	5338	5338	334	13864-13895
#10	10	8704	545	512	4468	280	MFM	5892	5892	369	13896-13927
[T00.1]		0		0				00000000			20050-20049
T00.1	EMPTY EMPTY EMPTY EMPTY EMPTY EMPTY EMPTY EMPTY EMPTY EMPTY EMPTY EMPT										
[T01.0]		6194		49552				2D0E536A			20078-26271
#0	N/A	98416	6152	N/A	1693	106	MFM	0032	0032	2	20078-20109
[T01.1]		0		0				00000000			26300-26299
T01.1	EMPTY EMPTY EMPTY EMPTY EMPTY EMPTY EMPTY EMPTY EMPTY EMPTY EMPTY EMPT										
[T02.0]		6194		49552				1AAC3A74			26328-32521
#0	N/A	98416	6152	N/A	1676	105	MFM	0032	0032	2	26328-26359
[T02.1]		0		0				00000000			32550-32549
T02.1	EMPTY EMPTY EMPTY EMPTY EMPTY EMPTY EMPTY EMPTY EMPTY EMPTY EMPTY EMPT										
[T03.0]		6194		49552				24B9CF19			32578-38771
#0	N/A	98416	6152	N/A	1668	105	MFM	0032	0032	2	32578-32609
[T03.1]		0		0				00000000			38800-38799
T03.1	EMPTY EMPTY EMPTY EMPTY EMPTY EMPTY EMPTY EMPTY EMPTY EMPTY EMPTY EMPT										

Donc on est bien sur un format propriétaire sur l'ensemble du disque.

Part 3 Let's do it

Insérez la disquette originale du jeu dans le lecteur de l'Amiga et chargeons la premier piste (et seulement **dos-lisible**) dans la mémoire pour voir ce qui se passe.

#RT alias Read Track, permet le chargement de la track 0 à 1 (1ère piste de la face 0)
#M, alias Visualisation mémoire HEXA/ASCII

Entrez dans votre AR et tapez le texte suivant : `rt 0 1 70000` puis `m 70000`

Comme le montre clairement l'image ci-dessus, la 1er piste est standard avec un *header* '**DOS**'
Aller donc désassembler ce bootcode, juste après ce header :
#Désassemblage à partir de l'adresse mémoire 7000C
Taper `d 7000c`

Nous avons ici une petite routine de copie.
Elle copie le reste du bloc en **$7FA00** et l'exécute.

Effectuer un reboot de votre Amiga et laisser le jeu se charger, il semblerait qu'il charge toutes les données d'un seul coup et démarre le jeu.

Ce qui serait intéressant pour nous, serait de trouver l'adresse du début du jeu en mémoire et de la sauver sur disquette.

Nous savons que le *bootcode* est déplacé en **$7FA00.**
Allons donc désassembler cette zone mémoire et chercher le code qui, après chargement, démarrer le jeu.

Rebooter encore une fois votre Amiga, laissez le jeu commencer à charger les données et **entrer** dans votre **AR**
#D alias désassemblage à partir de l'adresse 7fa00
Taper `d 7fa00`

```
********************************************************************************
                        ACTION REPLAY AMIGA MK III
                (c) 1990/1991 by Olaf Boehm & Jörg Zanger
                      (p) by Datel Electronics Ltd
********************************************************************************
No known virus in memory!
Ready.
d 7fa00
~07FA00 MOVEA.L 00000004.S,A6
~07FA04 ADDQ.B  #1,127(A6)
~07FA08 JSR     -96(A6)
~07FA0C LEA     000005E0.S,A7
~07FA10 LEA     00DFF000,A6

d 7fc32
~07FC32 MOVE.B  59(PC,D1.W),1E1(A0)
~07FC38 MOVE.B  54(PC,D0.W),280(A0)
~07FC3E MOVE.B  4E(PC,D1.W),281(A0)
~07FC44 MOVE.W  7FD40(PC),D0
~07FC48 BNE     0007FAFA
~07FC4C JMP     00000600.S
;-------------------------------------

-
```

Après quelques sous-routines, il semblerait qu'il se branche à l'adresse **$600**
Effectuez encore un reset de votre Amiga, laissez le jeu se charger un petit peu comme précédemment.
Nous allons insérez un *'breakpoint'* à l'adresse **$600**

Entrer dans votre **AR**
#BS permet, dès que l'adresse mémoire $600 est atteinte, d'effectuer un arrêt du code.
#X permet le retour au code Amiga en court.
Taper `bs 600` puis `x`

Une fois le breakpoint inséré et le retour au code normal, le jeu continue son chargement.
Comme prévus, dès l'adresse **$600** atteinte, notre **AR** prends la main.

```
bs 600
Breakpoint inserted
Ready.
x
No known virus in memory!
Ready.
Breakpoint raised at address: 00000600
```

Profitons-en pour sauver tout ça sur disquette, insérez une disquette vierge dans df0
#SM, alias SaveMemory permet donc de sauver la zone mémoire de 600 à 7D000
Taper : `SM a,6000 7D000`

Puis, rechargez le tout à l'adresse **$600** et exécutons ce code
#LM, alias LoqdMemory permet donc de charger un fichier à l'adresse mémoire $600
#G comme GO, démarre le code en $600 comme demandé.
Taper : `LM a,6000` puis `G 600`

```
lm a, 600
Loading from 000600 to 07D000
Disk ok
g 600_
```

Le jeu se lance.
Commencez une partie afin de tester si tous se passe bien.
Bien sûr, cela aurait été trop beau, très rapidement le message **BUM COPY** apparaît à l'écran :

Suivi d'un crash complet du jeu.

Au bout de quelques essaies, il semblerait que ce message apparaisse tout le temps à la 26eme seconde du temps.

Il serait sympa de trouver ce compteur et voir si on peut faire des choses sympathiques avec.
Effectuez encore une fois un redémarrage de votre Amiga, **entrez** dans votre **AR**, recharger le jeu et exécuter le comme précédemment.

```
lm a, 600
Loading from 000600 to 07D000
Disk ok
g 600_
```

Comme précédemment, commencez une partie.
Quand le compteur **TIME** en haut à droite de l'écran, affiche **29**, **entrer** dans votre **AR**
#TS, alias Training Search, permet de commencer une recherche de 'trainer', pratique pour les compteurs.
Taper : `TS 29` puis `X`


```
Ready.
ts 29
First trainpass!
Change the countvalue next time!
Searched up to: C80000
Trainmode active!
Ready.
X_
```

Quand le compteur **TIME** en haut à droite de l'écran, affiche **28**, **entrez** dans votre **AR**
#T, alias Training, permet de continuer une recherche de 'trainer'.
Taper : `T 28` puis `X`

```
Ready.
t 28
Possible addresses:
0007DF
Searched up to: C80000
Trainmode active!
Ready.
```

Comme le montre l'image ci-dessus, une adresse mémoire est rapidement trouvée, à savoir : **$7DF**

Allons chercher quelle adresse d'opcode fait appel à cette zone mémoire.
#FA, alias Find addressing opcode, permet de chercher une adresse de code modifiant une zone mémoire.
Taper : `FA 7DF`

```
FA 7df
Search from: 000000 to: C80000
001172 CMPI.B   #26,000007DF.S
0011E4 MOVE.B   000007DF.S,D0
0011F2 MOVE.B   D0,000007DF.S
006012 MOVE.B   000007DF.S,D0
00606A MOVE.B   000007DF.S,D0
006078 MOVE.B   D0,000007DF.S
006200 MOVE.B   000007DF.S,D0
006310 MOVE.B   #30,000007DF.S
006BCA CMPI.B   #6,000007DF.S
006BEC MOVE.B   000007DF.S,D0
006BFC MOVE.B   000007DF.S,D0
_
```

La première ligne à tendance à nous sauter aux yeux :
001172 CMPI.B #26,000007DF.S

Allons désassembler à partir de cette adresse mémoire.
Taper : `D 1172`

```
d 1172
~001172 CMPI.B  #26,000007DF.S
~001178 BNE     00001194
~00117C LEA     FFFFFE4A.S,A0
~001180 TST.W   7D0(A0)
~001184 BEQ     00001194
~001188 MOVE.W  #C8,00000952.S
~00118E MOVE.W  #18,00000954.S
~001194 NOP
~001196 MOVE.B  00000608.S,D1
~00119A ANDI.W  #3,D1
~00119E BNE     000011A8
~0011A2 JSR     00006C8C
~0011A8 NOP
~0011AA NOP

?fffffe4a+7d0
%00000000000000000000011000011010 = $0000061A = !0000001562

m 61a
:00061A FF FF 03 33 0E EE 0E 6E 0C C0 00 0F 00 0F 00 00 ...3...n........
```

Voyons voir...

```
001172 CMPI.B   #26,000007DF.S   Compare si $7DF 'compteur TIME' = 26, si c'est le cas Z=1
001178 BNE      00001194         Est ce que Z=0 ? Si Z=0 alors on branche en 1194 => Boucle normal
00117C LEA      FFFFFE4A.S,A0    Sinon on continue en positionnant FFFFFE4A dans A0
001800 TST.W    7D0(A0)          On test si l'adresse '7D0+A0' est égale à 0
                                 FFFFFE4A.S + 7D0 = $61A
                                 Donc est testé que l'adresse en $61A contient des 00, si c'est le cas Z=1

001184 BEQ      00001194         Si Z=1, alors on branche en $1194
001188 MOVE.W   #C8,00000952.S   Si c'est pas le cas.... On positionne C8 à l'adresse $952
00118E MOVE.W   #C18,00000954.S  et C18 à l'adresse $954

001194 NOP                       Début mode 'normal', TIME ≠ 26
001196 MOVE.B   00000708.S,D1    Début de routine pour décompte compteur, aucune importance dans notre cas.
00119A ANDI.W   #3,D1
00119E BNE      000011A8
0011A2 JSR      00006C8C
```

Allons jetez un coup d'œil en **$61A**
#M alias memory read, permet de voir les données en mémoire.
Taper : `M 61A`

Comme le montre l'image ci-dessus, l'adresse mémoire **$61A** contient les valeurs **FF FF**

Voyons voir, si en modifiant les valeurs en **$61A** on arrive à éviter le crash
Taper : `M 61A` et **modifiez**

```
:0061A FF FF 03 33 0E EE 0E 6E 0C C0 00 0F 00 0F 00 00 ...3...n........
```

en

```
:0061A 00 00 03 33 0E EE 0E 6E 0C C0 00 0F 00 0F 00 00 ...3...n........
```

Puis **Taper :** `x`

Le compteur *TIME* dépasse le 26 et le jeu semble ne pas planter.
On pourrai se contenter de positionner ces valeurs et sauver le tout mais peut-être existe-t-il une sécurité autre part !
On préféra donc rechercher l'adresse du code qui modifie cette valeur.

Redémarrez encore votre Amiga, **recharger** et **exécuter** le fichier comme précédemment,
Mais **1s** juste après avoir taper le `G 600`
Ré-entrer dans votre **AR**

Vérifiez qu'il y a bien des **00** à l'adresse **$61A**
Taper : `M 61A`

Puis, positionnez un 'Match Point'.
#MS, alias Set Match Point, Active l'AR dès que la zone mémoire en question est modifiée
Taper : `MS 61A` puis `MW`
Puis : `X`

```
**************************************************************************
                    ACTION REPLAY AMIGA MK III
              (c) 1990/1991 by Olaf Boehm & Jörg Zanger
                   (p) by Datel Electronics Ltd
**************************************************************************
No known virus in memory!
Ready.
lm a,600
Loading from 000600 to 07D000
Disk ok
g 600
No known virus in memory!
Ready.
m 61a
:00061A 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................

ms 61a
Memwatchpoint set
Ready.
mw
List of memwatchpoints: 00061A
Ready.
x_
```

Attendez quelques secondes que votre **AR** reprenne la main.

```
Ready.
Memorybyte 00061A has changed from $00 to $FF
```

Désassemblons ce bout de code en prenant le peine de remonter un peu en adresse mémoire afin de mieux comprendre ce qui se passe.
Taper : `D`

```
~024806 ADDQ.W  #2,A0
~024808 AND.L   D1,D4
~02480A CMP.L   D0,D4
~02480C DBEQ    D2,00024804
~024810 BEQ     0002482E
~024812 LSR.L   #1,D0
~024814 ROR.L   #1,D1
~024816 ADDQ.W  #1,D3
~024818 CMP.W   #10,D3
~02481C BNE     000247FA
~02481E LEA     00000962.S,A0
~024822 MOVE.W  #FFFF,-348(A0)
~024828 BSR     00024878
~02482C RTS
;=====================================


?962-348
%00000000000000000000011000011010 = $0000061A = !0000001562
_
```

Encore une fois, la commande située en **$24822** saute aux yeux.
En effet, l'instruction en **$2481E** place la valeur 962 en **A0**
Est placé ensuite en **$24822** les valeurs **#FFFF** à l'adresse mémoire de **A0**-348 (donc 962-348)
À savoir : **$61A**

BINGO, il ne reste plus qu'à changer ces deux valeurs par **00 00**
Taper : `M 24822` et modifier

```
:024822 31 7C FF FF FC B8 61 00 00 4E 4E 75 59 48 28 10      1|..ü.a..NNuYH(.
```

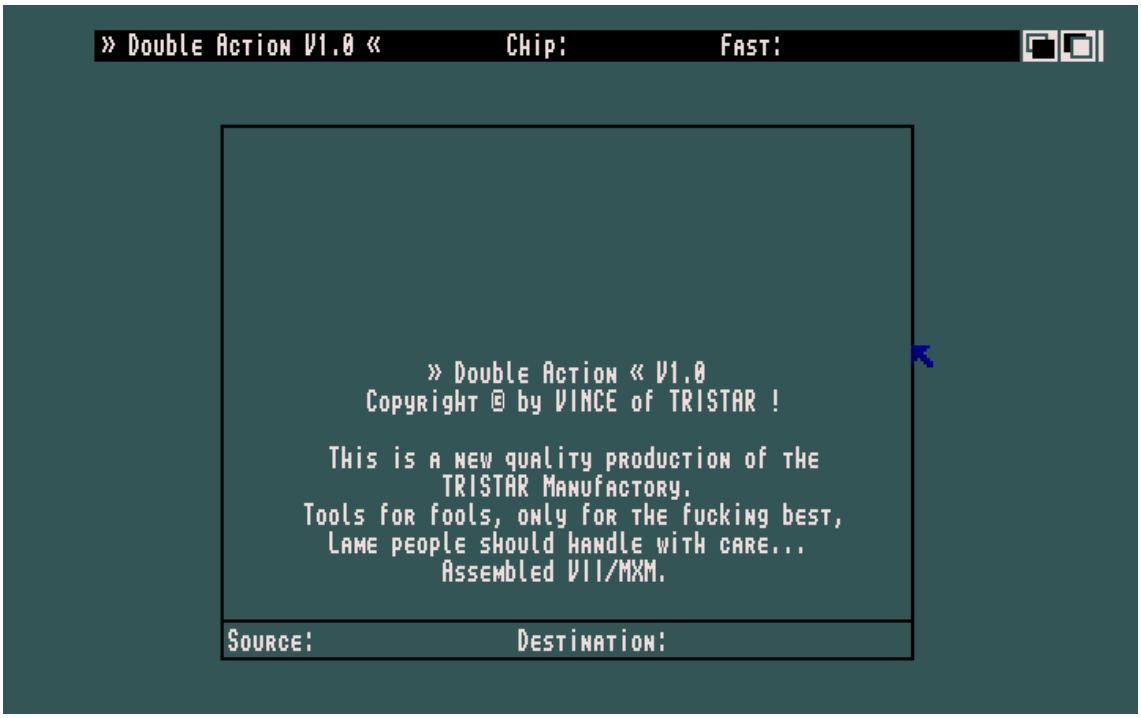
en

```
:024822 31 7C 00 00 FC B8 61 00 00 4E 4E 75 59 48 28 10      1|..ü.a..NNuYH(.
:024832 C8 81 20 3C 24 44 00 00 E6 A8 B8 00 66 DE 41 F8      .. <$D......f.A.
```

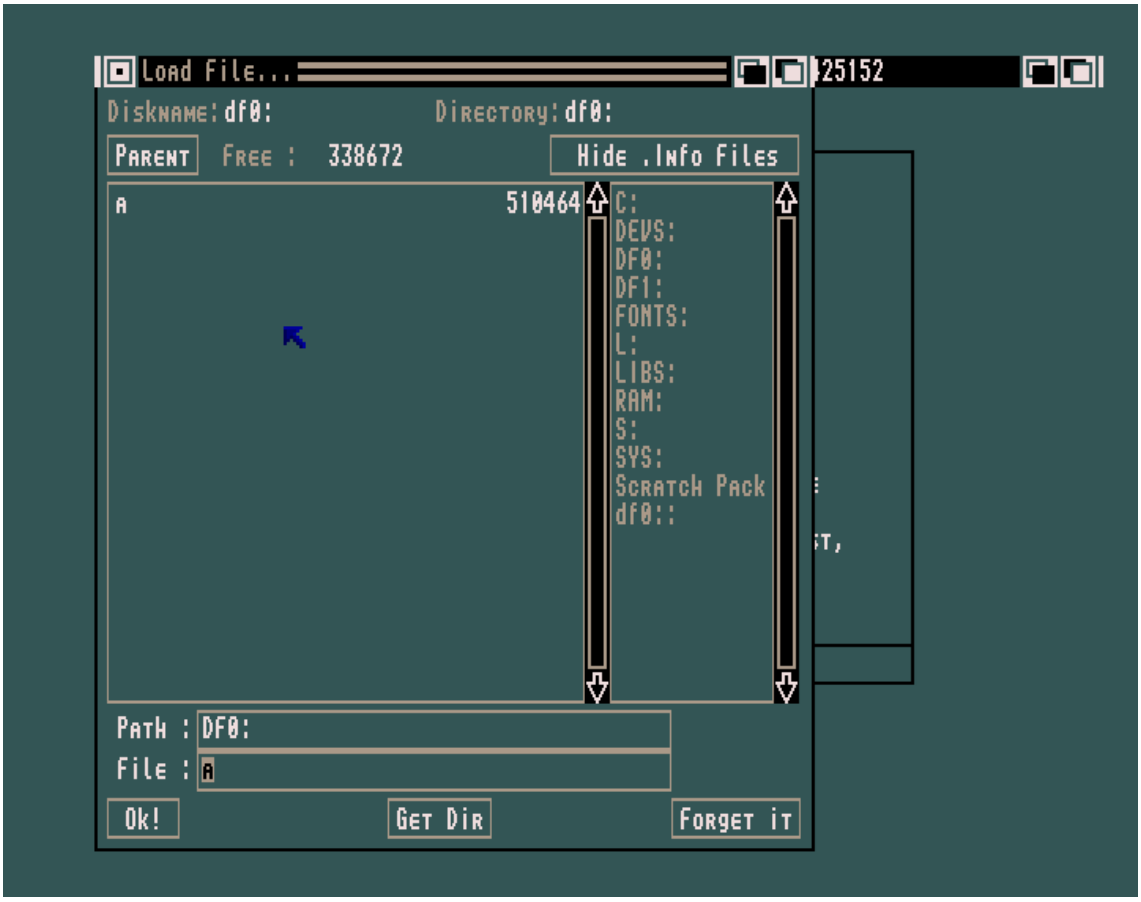
Sauvons maintenant le tout
Taper : `SM A,600 7D000`

```
M 24822
:024822 31 7C FF FF FC B8 61 00 00 4E 4E 75 59 48 28 10 1|..ü.a..NNuYH(.

M 24822
:024822 31 7C 00 00 FC B8 61 00 00 4E 4E 75 59 48 28 10 1|..ü.a..NNuYH(.
:024832 C8 81 20 3C 24 44 00 00 E6 A8 B8 00 66 DE 41 F8 .. <$D......f.A.

SM a, 600 7d000
```

Il ne reste maintenant plus qu'à compresser le fichier sauvé et le rendre exécutable.
Insérez dans l'Amiga la disquette contenant le logiciel de compression *'Double Action'* de VINCE of TRISTAR
Redémarrez et lancer cet outil.

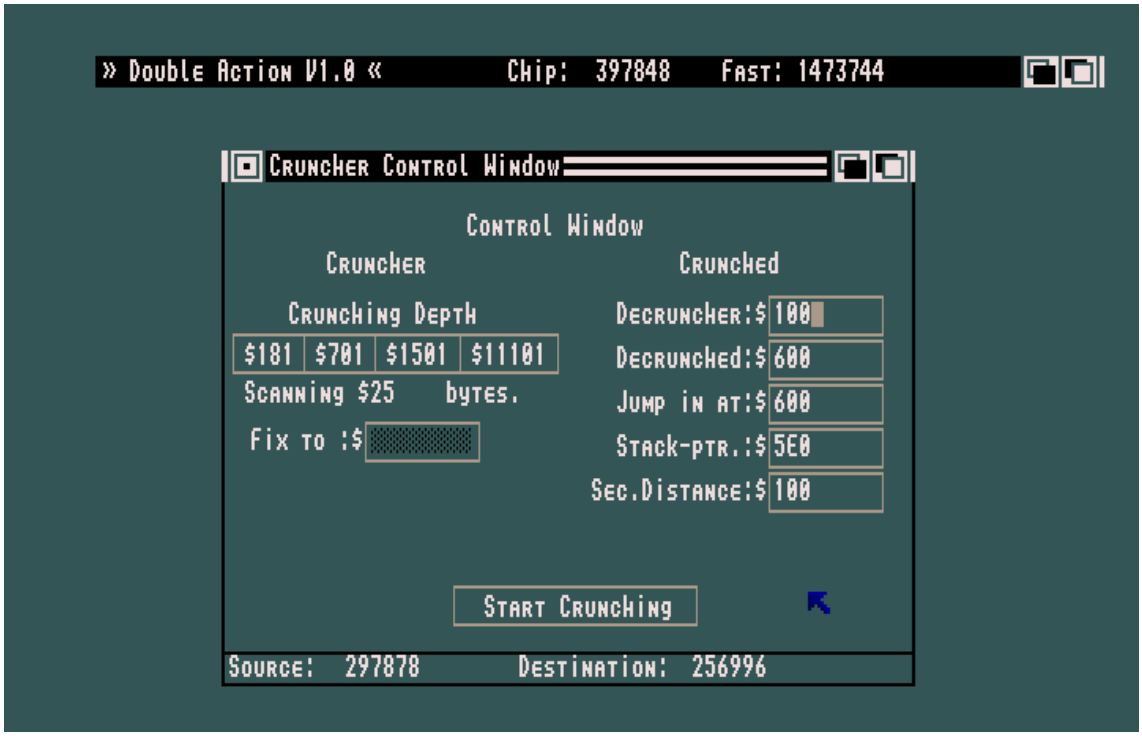


Comme indiqué sur les images prises, chargez votre fichier préalablement sauvé :
PROJEKT => LOAD FILE => a



Concernant les valeurs de compression :

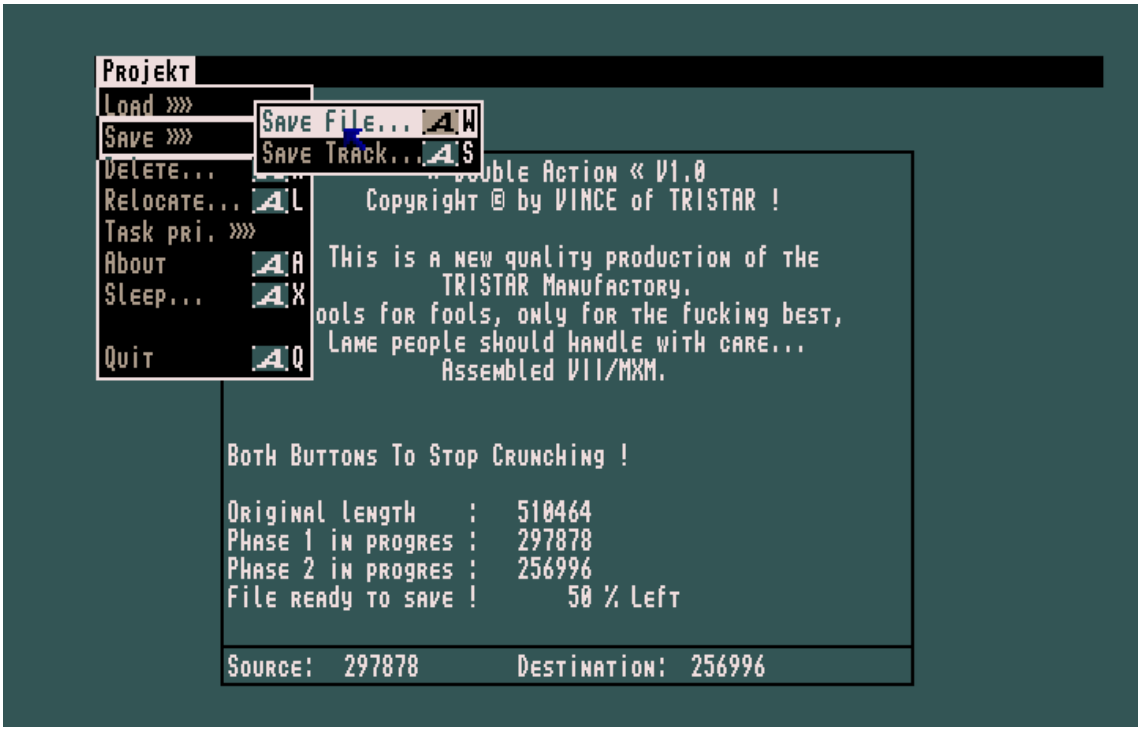
```
Crunching depth  : $25    # meilleur compromis
Decruncher       : $100   # Le jeu n'utilise pas cette zone mémoire
Decrunched       : $600   # exactement comme le jeu original
Jump in AT       : $600   # Adresse de départ du jeu
Stack            : $5E0   # Juste au-dessus du code, le jeu original utilise d'ailleurs cette valeur, voir 3eme image de ce tuto
Sec distance     : $100
```



Puis **CLIQUER** sur **START CRUNCHING**

La compression est en court …

Une fois terminée, sauvez le tout sur une nouvelle disquette vierge par exemple :
PROJEKT => SAVE FILE => ik+

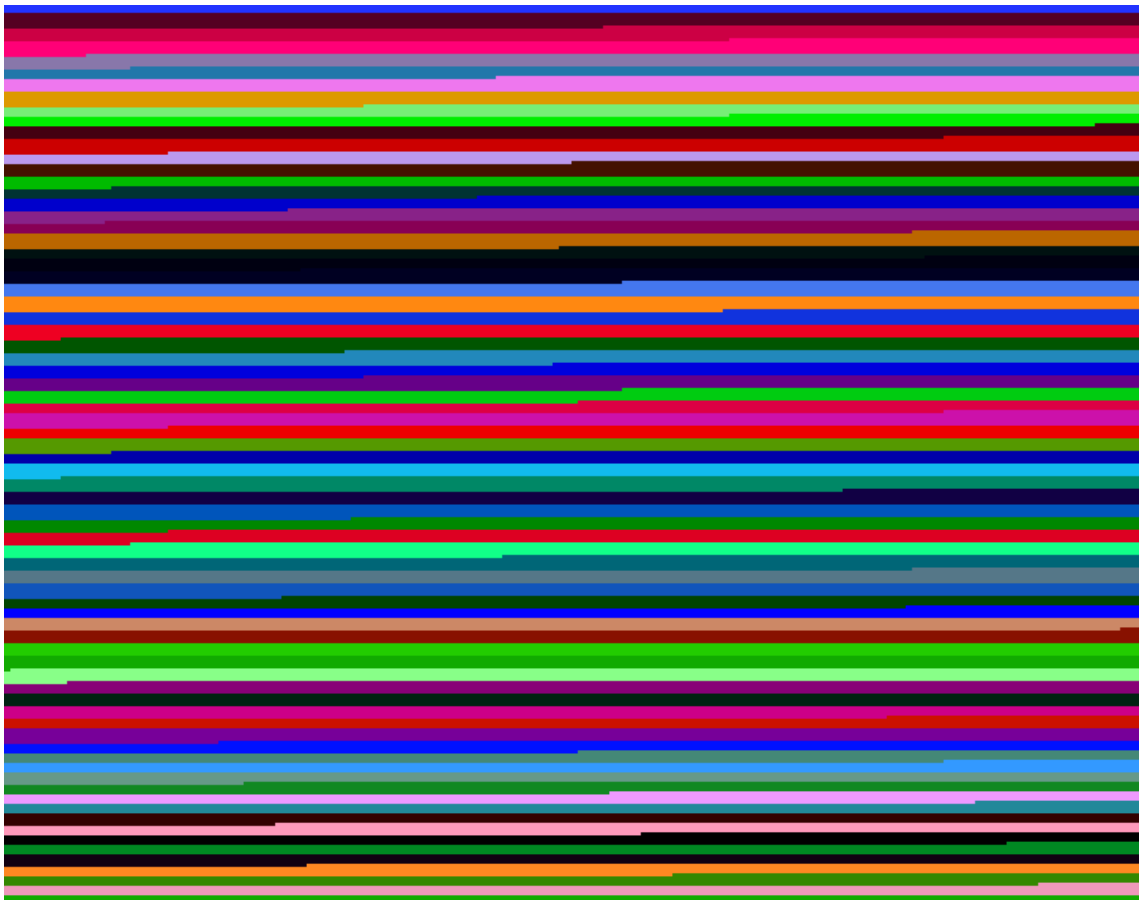


Il ne reste plus qu'a rendre la disquette bootable, **entrez** dans votre **AR**
#INSTALL, alias BOOTBLOCK INSTALL, permet d'installer un secteur de boot sur l'unité indiquée, 0=DF0, 1=DF1
et **Taper :** `INSTALL 0`

```
dir
Directory of (df0:)
      257624  ik+
1220 blocks free, 30.6 % of disk used
Disk ok
install 0
Ready to install disk in drive df0:? (y/n)
y
Disk ok
```

Effectuer un **reset** de votre Amiga et à l'invite de commande, **tapez** le **nom du fichier** préalablement sauvé, à savoir **ik+**

Après une phase de décompression, le jeu se lance.



Bien sûr, vous pouvez bien sur créer un fichier *startup-sequence* qui automatise tout ça, créer une cracktro, etc.