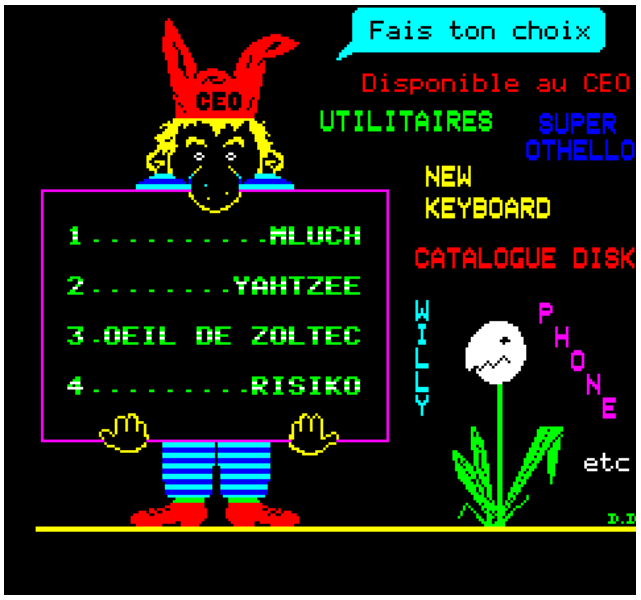
# La protection des disquettes selon Daniel Duffau (5e partie)

## Utilisation du debugger d'Euphoric pour récupérer Risiko

*Par André C.*

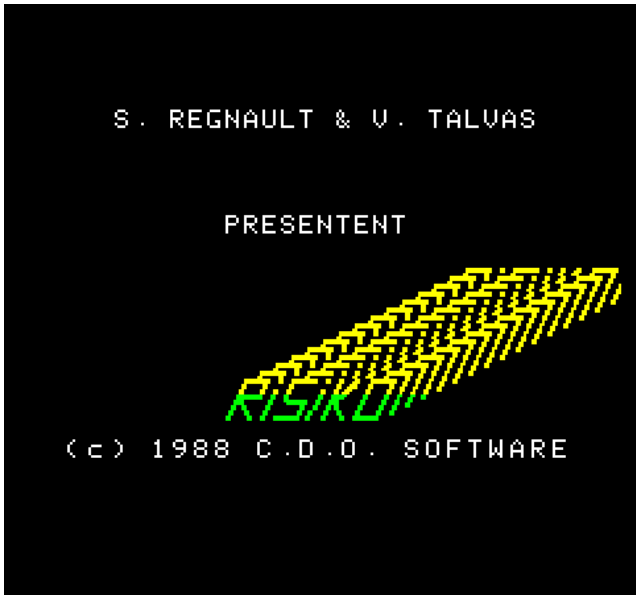



*Ecran-menu de la disquette CEO-soft 2*

### Récupération de Risiko.

Ouf, j'y suis arrivé ! Une semaine à temps plein, de la ténacité, des espoirs, de la logique, des erreurs de raisonnement, des déceptions ont pavés mon chemin. Combien de fois me suis-je dis : "Ce coup-là, c'est le bon, d'ailleurs je n'ai plus d'autre idée...", avant de constater que ce n'était pas encore le grand jour ! Mais à chaque fois, sur le point de laisser tomber, j'ai encore eu "une autre idée"... Bref, tout ça pour vous avouer que je ne suis pas qu'un peu fier d'avoir finalement réussi !

Via la disquette trimestrielle, je peux vous fournir une version propre de Risiko (et non protégée, œuf corse). Je peux même affirmer qu'il s'agit à l'octet près, du programme qui a servi pour fabriquer la disquette commercialisée par le CEO sous l'appellation "CEO-soft 2".





*Ecran-titre et écran-menu de Risiko*

### Etat du problème

Contrairement aux trois autres jeux (Mluch, Yahtzee et L'œil de Zoltec), Risiko n'est pas un programme Basic, mais un programme en langage machine (LM). Il est formé de deux parties distinctes (mais ça, ce n'est que maintenant que je le sais). Ces deux parties sont constituées de centaines de sous-programmes (évidemment dans le plus grand désordre) entrecoupés de zones de données bidon. Il s'articule autour d'une zone de données (cruciale celle-là) de #2000 à #2A9F sans cesse remodelée au cours de l'avancement de l'exécution.

Comme je vous l'ai expliqué précédemment, les quatre jeux sont "cachés" parmi les secteurs de la disquette. Ce n'est que lorsqu'on a exprimé son choix (de "1" à "4") que les secteurs ad hoc sont lus et copiés en mémoire. Enfin les octets copiés sont transcodés pour donner le vrai programme qui est alors lancé.

Simple non ? Oui sauf que le programme propre à la protection "DD" (Daniel Duffau) est hyper complexe et est lui-même camouflé en petits bouts au milieu de flopées de données sans intérêt.

Si la situation est simple avec un jeu en Basic, dans le cas de Risiko (en LM) on ne peut pas deviner ce qui appartient au jeu et ce qui appartient au programme de protection.

**Le clou de la protection est l'utilisation d'opcodes "non documentés".**

Petit encart : Il existe sur Internet de nombreuses informations sur ces opcodes non officiels. Je suppose qu'ils n'ont pas été développés à dessin par les concepteurs du 6502, mais qu'ils résultent d'une sorte d'état de fait électronique. Quand on fournit un code "hors nomenclature" au 6502, il ne le rejette pas forcement, mais dans certains cas, il en fait quelque chose d'inattendu (mais de reproductible). Des enragés (oups pardon, des passionnés) ont étudié tout ça, défini l'action générée et les flags modifiés. En général, un opcode "non documenté" est équivalent à la combinaison de plusieurs opcodes normaux. Par exemple, le code #2B est appelé "ANC #ab". Son action correspond à un AND entre l'Accumulateur et la valeur de l'argument "ac", suivit du transfert du bit 7 du résultat dans C (Carry). Si on n'utilise pas A et qu'on ne teste pas C, cet opcode et son octet "ab" associé sont sans effet sur le déroulement du programme.

Cerise sur le gâteau, les opcodes "non officiels" ne sont reconnus par aucun des désassembleurs 6502 (ni par celui du debugger d'Euphoric). Résultat, le désassembleur indique "??" dans le listing et passe à l'octet suivant (qui est le ou les octets associés "ab", voire "ab cd"). Pire, dans le registre "pas de bol", si l'octet "ab" correspond à un opcode normal, l'assembleur repart, sur un mauvais pied, dans une fausse direction.

Quand on bourre le programme LM avec des opcodes "non documentés", le listing de désassemblage ressemble alors comme deux gouttes d'eau à celui qu'on obtient en désassemblant une zone de données au lieu d'une zone de programme ! Si j'y pense je ferais un article là-dessus avec des exemples concrets tirés de la protection "DD".

**Stratégie (sic)**

Fidèle à ma devise "Pourquoi s'emm... quand on peut trouver un autre chemin plus simple ?", je me suis entêté à procéder comme avec les trois jeux précédents : Sous Euphoric, appuyer sur F9 au bon moment, examiner le fichier "Dump" obtenu, copier la zone du jeu décodé, la coller dans un fichier TAP, ajouter par devant un entête approprié, le CLOADer, le sauver sur une disquette et enfin le tester. Pas de problème, avec cette procédure elle-même : je suis maintenant bien rodé. Particulièrement rodé même avec Risiko, car j'ai procédé à une bonne trentaine de Dump... Avec, à chaque fois, une idée de génie... géniale jusqu'au moment du test !

**Un point très sensible propre à Risiko**

Il faut dire que, comme indiqué plus haut, parmi les nombreuses difficultés de l'opération, l'une des plus embêtantes est que le jeu s'articule autour d'une zone de données cruciale, sans cesse remodelées au cours de l'avancement de l'exécution. Si on reprend l'exécution au départ avec des données correspondant à un autre état d'avancement de l'exécution, le jeu plante.

Donc, il faut appuyer sur F9 juste à l'instant où le jeu est lancé. Ou prier pour qu'il n'ait pas encore eu le temps de modifier la zone de données "cruciales". J'ai chronométré. Entre le moment on l'on appuie sur la touche "4" et l'instant où le jeu est lancé, il s'écoule 4 secondes ! Bonjour la précision pour appuyer sur F9 !

Merci Fabrice, sans toi je n'y serais jamais arrivé. Non seulement tu as doté Euphoric de la touche de dump de la mémoire F9, mais tu nous as donné un émulateur à vitesse variable. En effet il est possible de remplacer la fréquence normale (1.0 MHz) par une plus lente ou une plus rapide. Dans les plus lentes, j'ai testé jusqu'à 0.1 MHz (on peut baisser encore, mais c'est difficilement utilisable). Du coup, mon délai pour appuyer sur F9 est passé de 4 secondes à 35 secondes. Un luxe !

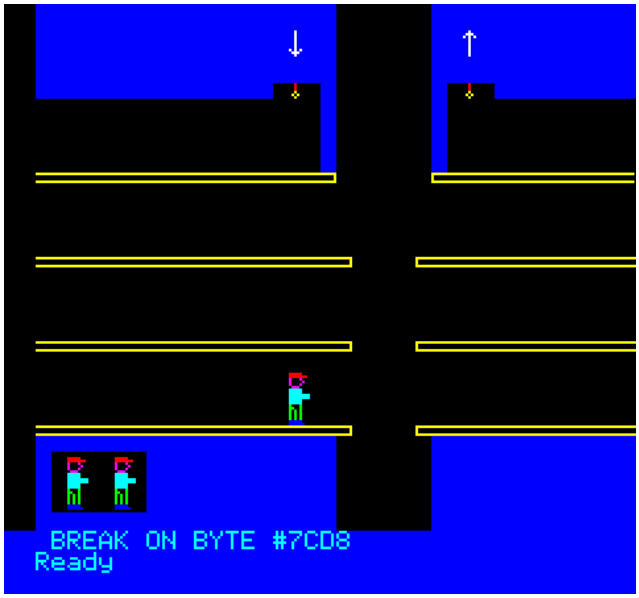
**Adresse de lancement de Risiko**

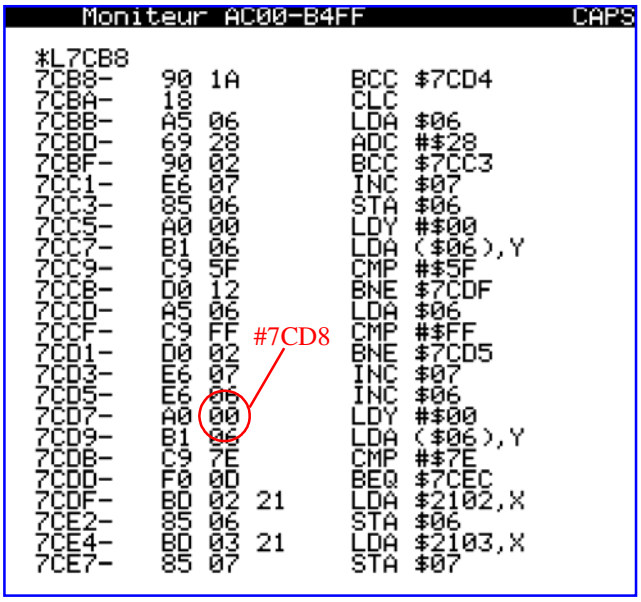
D'ailleurs pour en revenir à la question de lancement du jeu, où se situe l'adresse de lancement ? Au début du premier morceau ? Au début du second morceau ? Au début d'un autre morceau ? Ou bien ailleurs au milieu d'un morceau ? Jusqu'à la fin, je n'ai jamais connu la structure réelle du programme, ni quelle adresse de lancement utiliser.

**Essais divers et avariés**

Ma devise "Pourquoi s'emm… quand on peut trouver un autre chemin plus simple ?" ne m'a finalement pas réussi et j'avoue même qu'elle m'a fait perdre beaucoup de temps. A chacune de mes tentatives, je me suis heurté soit à un plantage muet, soit à un "BREAK ON BYTE #7CD8".

Heureusement, je n'ai trouvé que très tardivement le remède à ce "BREAK ON BYTE #7CD8", car j'aurais pu croire que tout était résolu et me contenter




```
*L7CB8
7CB8-   90 1A       BCC $7CD4
7CBA-   18          CLC
7CBB-   A5 06       LDA $06
7CBD-   69 28       ADC #$28
7CBF-   90 02       BCC $7CC3
7CC1-   E6 07       INC $07
7CC3-   85 06       STA $06
7CC5-   A0 00       LDY #$00
7CC7-   B1 06       LDA ($06),Y
7CC9-   C9 5F       CMP #$5F
7CCB-   D0 12       BNE $7CDF
7CCD-   A5 06       LDA $06
7CCF-   C9 FF       CMP #$FF
7CD1-   D0 02       BNE $7CD5
7CD3-   E6 07       INC $07
7CD5-   E6 06       INC $06
7CD7-   A0 00       LDY #$00
7CD9-   B1 06       LDA ($06),Y
7CDB-   C9 7E       CMP #$7E
7CDD-   F0 0D       BEQ $7CEC
7CDF-   BD 02 21    LDA $2102,X
7CE2-   85 06       STA $06
7CE4-   BD 03 21    LDA $2103,X
7CE7-   85 07       STA $07
```

*Quand on est plein d'espoir, que c'est désagréable de recevoir ça en pleine figure. Surtout quand il n'y aucune raison pour que l'exécution atterrisse en #7CD8 (où il y a effectivement un #00 !)*

d'un code incomplet ou faux. En effet, je suis bien incapable de tester un jeu de plateaux comme Risiko jusqu'au bout. Après bien des errances, il m'a fallu y renoncer et accepter de me pencher sur le programme de protection "DD" lui-même et le désosser.

**Le mécanisme de chargement-décodage de Daniel Duffeau**

Rassurez-vous, je ne vais pas vous asséner des pages et des pages de listing de désassemblage commenté, mais seulement les grandes lignes de l'opération.

Partant du menu de CEO-soft 2, pendant lequel le programme est en attente qu'une touche soit pressée, j'ai lancé le debugger avec un appui sur F11. J'ai constaté que le programme surveillait la mémoire #0208 ("motif ligne / colonne de la dernière touche pressée"), en copiant la valeur dans A et en analysant le contenu de A. Avec le debugger, je lui ai donc fourni la valeur A=#9A (qui correspond à la touche "4") et le programme, après quelques ajustements (et donc quelques appuis frénétiques sur F3), a commencé à lire le programme Risiko sur la disquette.

Petit encart : Dans le debogger d'Euphoric, F3 permet d'exécuter pas à pas le code au pointeur PC, un mnémonique à chaque appui sur F3. Sauf si le mnémonique est un JSR, auquel cas le sous-programme est exécuté tout d'un bloc et le pointeur PC passe à la ligne suivant le JSR.

En #9800, un sous-programme lit les secteurs correspondant au jeu "4" (Risiko) et les copie en Ram à partir de #1A36. Il s'agit des secteurs situés sur la disquette à partir du secteur #05 de la piste #19. Il reboucle jusqu'à ce que X atteigne la valeur #7A (en #9821) et donc après avoir copié #79 secteurs (121 en décimal), soit #7900 octets, qui se retrouvent maintenant de #1A36 à #9335.

Petit "détail" de l'histoire, comme je soupçonnais que le transcodage commencerait dès que la copie serait terminée, il me fallait surveiller celle-ci avec le debogger d'Euphoric. Indépendamment de la vitesse assignée au 6502, ce processus est long (des heures) avec le debugger, car celui-ci affiche à l'écran toutes les données du fonctionnement de l'Oric (code exécuté, pile, tous les registres, éventuellement le dump du résultat, pour surveiller l'avancement de la copie, etc.). Et ceci pour #7900 (30976 en décimal) octets. J'ai donc été amené à bloquer l'appui sur la touche F3 en posant dessus une gomme et mon appareil photo numérique sur la gomme, en guise de poids (vive Gaston-la-gaffe !).

Je suis donc arrivé au point névralgique, après avoir pratiqué manuellement les derniers appuis sur F3 pour arrêter pile-poil. J'ai donc pu tout savoir sur le transcodage.

En # 9300, un sous-programme lit, un à un, les #7900 octets, les "transcode" et les réécrit à la même place. L'opération est on ne peut plus simple : il suffit d'appliquer un EOR #1A à chaque octet codé pour qu'il redevienne l'octet original.

Petit encart : Le mnémonique EOR applique un "OU exclusif" entre chaque bit de l'accumulateur A et le bit homologue d'une valeur donnée en argument, selon une "table de vérité" très simple : si l'un des 2 bits (et un seul) est à 1, alors le bit résultant est mis à 1 et inscrit dans A, à la place du bit d'origine. Si l'on a 1 et 1, ça donne donc 0, de même si l'on a 0 et 0, ça donne 0.

Re-petit "détail", comme je soupçonnais que Risiko

serait lancé dès le transcodage terminé, il me fallait surveiller celui-ci. Comme expliqué plus haut, cela prend des heures, de quoi attraper une crampe au doigt qui est crispé sur F3. Comme précédemment, j'ai donc bloqué la touche F3 et surveillé de temps en temps l'état d'avancement du processus. Quand il a commencé à se rapprocher de la fin, j'ai débloqué F3 et terminé l'approche manuellement. Ceci avec d'autant plus d'attention que j'ignorais l'adresse réelle de fin de Risiko. Je savais seulement que cela se situait entre #9236 et #9335, selon le degré de remplissage du dernier secteur copié de la disquette (256 octets au maximum, soit #100 octets).

Et là, suprême jouissance, je suis tombé au bout du processus sur un JMP #1A36 ! Et oui, finalement, le point de départ de l'exécution est bel et bien au début du premier bloc de programme LM.

**C'est bon d'avoir la certitude que Risiko est transcodé en entier, implanté à la bonne place (de #1A36 à #9335) et qu'il n'attend plus qu'un JMP #1A36 (ou un CALL #1A36)…**

Impossible de sortir du debugger par un nouveau F11, car cela lancerait immédiatement l'exécution et mon précieux Risiko serait bousillé (au moins dans sa zone de données cruciales).

J'ai un peu tremblé en appuyant sur F9 pour récupérer le dump complet de la mémoire de l'Oric et donc aussi la zone #1A36 à #9335.

Petit encart : En effet la touche F9 n'est pas documentée dans le mode d'emploi du debugger. Ce n'est pas sensé fonctionner. Mais je savais, par mes essais antérieurs, que ça marche (un fichier Dump correct est créé dans le répertoire Euphoric), mais Euphoric plante ensuite. Le Dump ne peut servir qu'à récupérer la mémoire mais ne peut pas être utilisé pour relancer l'Oric en l'état précédent l'appui sur F9. Ou plutôt si, mais il relance une machine plantée !

Bref, j'ai récupéré le bloc #1A36-#9335, ajouté par devant un entête cassette adapté et enfin sauvé Risiko.tap. Ouf ! Mais mes aventures n'étaient pas encore terminées pour autant. Je récupère bien ce TAP sur une disquette selon la procédure habituelle, CLOAD"Risiko" & SAVE"Risiko.bin",A#1A36,E#9335.

Petit encart : J'utilise toujours Sedoric 3.0, c'est pourquoi j'ai oublié, dans les articles précédents, de vous rappeler qu'il faut absolument utiliser Sedoric 3.0 et non une des versions précédentes afin que l'incompatibilité entre CLOAD et SAVE soit corrigée.

Je lance le jeu avec un CALL#1A36 et patatras, douche froide je retrouve mon fidèle "BREAK ON BYTE #7CD8 ! Pourtant, je ne me laisse pas faire. Le bon programme est en place et il doit démarrer en #1A36. Alors, que je passe-t-il ? Pour le savoir, j'appuie sur F11 (merci Fabrice) et découvre que PC=0486. En page 4 donc ! Merde, il faut tout simplement faire un QUIT ! Après plusieurs essais, je confectionne le lanceur suivant :

```
100 HIRES:QUIT
110 !LOAD"RISIKO.BIN"
120 CALL#1A36
```

Ce coup-là ça marche et je peux me mesurer à Risiko. Je constate que je suis toujours aussi doué pour les jeux de plateformes. Mais le but de tout ça n'était pas de pouvoir jouer à Risiko, ça je pouvais le faire à partir de la disquette CEO-soft 2. Non, mon problème était de résoudre une énigme et là, j'ai finalement réussi ! Fin !