

# Fugitif 31 : *Making Of*

— Crocofest 2026 —

## Introduction

Fugitif est un jeu d'aventures développé à la fin des années 80 sur Amstrad CPC. Sa particularité est d'utiliser des graphes en mode 1 sous *rasters*, permettant de modifier la palette de couleurs d'une ligne à l'autre (en fait, seules 3 couleurs sur 4 sont changées, le noir étant toujours utilisé).

Il y a quelques années, m'a pris l'idée saugrenue d'en faire un *reboot*. Ce document en donne quelques explications techniques, qui pourront être présentées au Crocofest 2026. Ne pouvant y participer, cela permettra, j'espère, de donner quelques infos intéressantes pour les plus curieux. Je n'ai à ce jour aucune date de sortie à communiquer, j'avance très doucement ;o)

## Des icônes au *point'n'click*, en passant par la console

### Fugitif – version originale (1991)

À l'époque, mon cousin et moi n'avions pas les compétences pour gérer autre chose pendant les *rasters*, c'est pourquoi nous avons choisi de basculer sur un autre écran pour la gestion du jeu. Du coup, il y avait beaucoup de place à meubler, d'où le système d'icônes (de mémoire, suggéré par **Bruno Gourrier**, un des fondateurs de Lankhor, chez qui le jeu allait être édité).



Gestion par icônes, adulée par les uns, détestée par les autres !

## Fugitif 31 – version console + interpréteur de commandes

La première approche pour ce *reboot* a été d'implémenter une console genre *shell* **Linux**, avec *complétion*, rappel des commandes, raccourcis clavier, etc. Et bien sûr un interpréteur de commandes. Le but était de rendre l'utilisation de ce système la plus conviviale possible.

Côté présentation, afin de conserver les graphes dans leur taille originale, l'écran est donc décalé vers le haut pour permettre d'afficher un *hud* en dessous, lequel contient la console et diverses autres informations. Là, un grand merci aux *démomakers* du "Z80 Corner" (salon *Discord* géré par **Siko**) qui m'ont aidés à coder correctement cette partie, en particulier à **Longshot**, maître incontesté du CRTC !



La console d'entrée/sortie, et les icônes de gestion

Le code correspondant à cette console pourra être publié si cela peut servir à d'autres, car elle fonctionnait relativement bien.

Mais avec un interpréteur de commandes, il est très difficile d'anticiper toutes les actions potentielles qu'un joueur veut faire. Implémenter le minimum d'actions nécessaires au jeu peut être très rapidement frustrant, alors que tout prévoir devient un vrai cauchemar pour le développeur, sans parler de la place nécessaire.



## Fugitif 31 – version *point'n'click*

Avec le *point'n'click*, c'est au contraire le développeur qui décide quelles interactions le joueur est autorisé à faire. Ce dernier est donc moins frustré, puisque l'interface *point'n'click* lui montre directement le “vocabulaire” compris par le jeu. Il n'a alors qu'à se concentrer sur le *gameplay*.

La solution *point'n'click* avait initialement été mise de côté pour **Fugitif 31** à cause de la dégradation du pointeur, du fait des *rasters* : un *sprite* change en effet de couleur au gré des lignes, et selon le graphe affiché ; l'effet “arc-en-ciel” est, disons-le, très moche !

D'où l'idée géniale de **Sylvain Guehl** : faire une ligne blanche sur toute la largeur de l'écran (facile à gérer via la table des *rasters*) pour gérer la position verticale, et une zone sombre de quelques pixels de large pour gérer la position horizontale. De plus, la ligne blanche devient rouge lorsqu'on survole une zone sensible, facilitant ainsi la recherche d'éléments interactifs. Le résultat peut paraître surprenant, mais fonctionne vraiment bien. Les couleurs de la ligne seront dynamiquement ajustées en fonction des images, pour conserver un bon contraste.



Ce n'est pas forcément évident, mais il y a une rupture après le graphe, car le *hud* est lui en mode 2. En effet, souhaitant créer une police à chasse variable, ce mode permet une plus grande finesse dans la définition et le positionnement des caractères (au *pixel* près). Toujours au sujet du texte, celui-ci est justifié automatiquement lors de la phase d'assemblage, via du code Lua. De même que la conversion UTF-8 → CPC. La fonte est dérivée de [Twobit](#).

Les graphes ne font pas la largeur totale de l'écran, car à l'origine, la routine de *rasters* n'était pas très bien écrite, et souffrait d'un *jitter* important. Du coup, pour éviter de voir clignoter des *pixels* sur les bords, j'avais réduit leur largeur.

À noter que le *hud* sur la vue ci-dessus n'est pas encore finalisé.

## Moteur de jeu *point'n'click*

La version *point'n'click* a nécessité de pouvoir décrire le GUI et définir le *gameplay* de manière conviviale, afin de ne pas se focaliser sur la partie technique pendant la création du scénario. Cela a pu se faire grâce à des macros [SjASMPlus](#), écrites en [Lua](#), pour *parser* la description des lieux. Ces macros convertissent un pseudo-langage en tables, interprétées par le moteur écrit en assembleur Z80.

Les deux macros principales permettant cela sont **M\_GUI**, pour décrire l'interface et le *gameplay* de manière synchrone (en réaction aux actions du joueur), et **M\_CODE**, pour exécuter des commandes de manière asynchrone (gestion de compteurs...).

Exemple :

```
M_GUI zone(2) object(TRUC)
M_GUI   action(LOOK) cmd(PRINT TRUC_LOOK)
M_GUI   action(OPEN)
M_GUI     if(TRUE var1)
M_GUI       cmd(SET var2 PRINT TRUC_OPEN)
M_GUI     else()
M_GUI       cmd(PRINT IMPOSSIBLE)
M_GUI   endif()
```

Ici, on réagit lorsque le pointeur est dans la zone 2. Cette zone correspond à l'objet TRUC.

On définit pour cet objet deux actions possibles pour l'utilisateur : LOOK et OPEN. Lorsqu'il choisit la première, on affiche simplement un message. Pour la seconde, le message affiché va dépendre de l'état de la variable var1.

Il peut y avoir plusieurs conditions dans un `if()` ou `elseif()` : elle sont combinées en un ET logique. De même qu'il peut y avoir plusieurs commandes en réaction à une demande du joueur.

Les conditions peuvent être imbriquées, et peuvent s'appliquer à n'importe quelle règle. Par exemple, une même zone peut changer d'objet selon le contexte :

```
M_GUI zone(1)
M_GUI   if(TRUE var1)
M_GUI     object(TRUC) action(LOOK) cmd(PRINT TRUC_LOOK)
M_GUI   else()
M_GUI     object(BIDULE)
M_GUI       action(LOOK) cmd(PRINT BIDULE_LOOK)
M_GUI       action(OPEN) cmd(SET var3 PRINT BIDULE_OPEN)
M_GUI   endif()
```

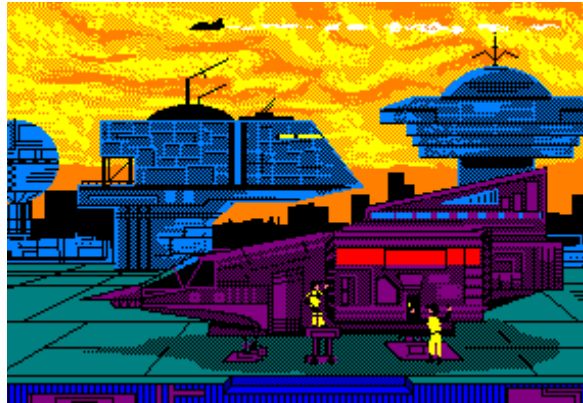
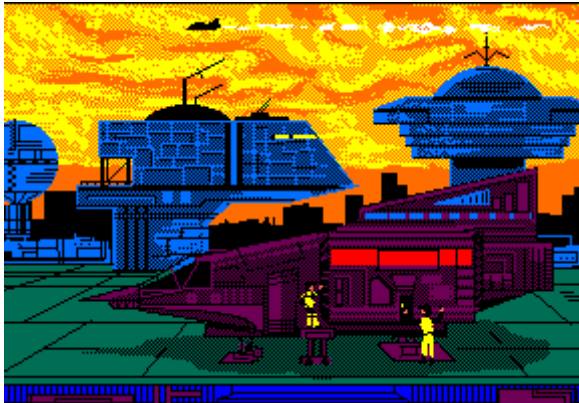
À noter que l'indentation et les retours à la ligne ne sont là que pour rendre le code plus lisible. Le *parser* ne tient compte que de l'ordre d'apparitions des règles `zone()`, `object()`, `tool()`, `action()`, `cmd()`, `if()`, `elseif()`, `else()`, `endif()`.

La macro **M\_CODE** fonctionne sur le même principe, mais ne correspond à aucune zone ; elle est exécutée après chaque action du joueur. Seules les règles `if()`, `elseif()`, `else()`, `endif()`, et `cmd()` sont donc implémentées.

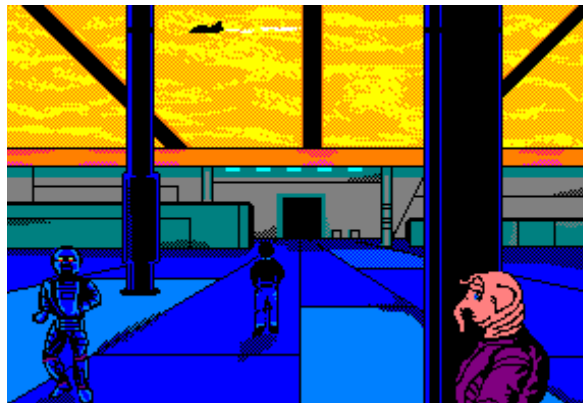
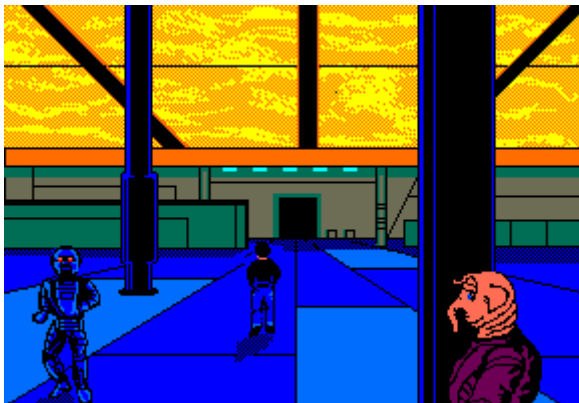
# Graphes

Les graphes originaux sont de **Jean-Paul Renault**, qui a fait un très beau travail, exploitant au mieux les *rasters* en mode 1 et la palette du CPC. Comme il n'existait aucun logiciel permettant de faire cela sur CPC, il avait travaillé sur Amiga.

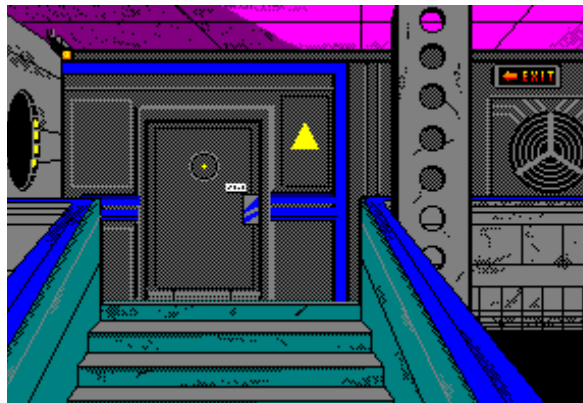
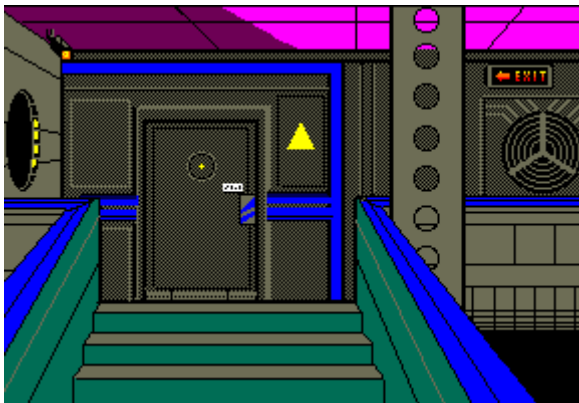
**Tero Heikkinen**, le développeur de [multipaint](#), a ajouté à ma demande un nouveau mode pour **Fugitif 31**, afin de pouvoir éditer des images en mode 1 avec toute la palette de couleurs, et qui intègre également un outil de vérification (pas plus de 4 couleurs par ligne). Un grand merci à lui ! **Fugitif 31** bénéficiera donc de quelques retouches graphiques, que ce soit pour corriger ou améliorer un graphe, voir le modifier plus en profondeur pour les besoins du nouveau *gameplay*.



Correction de la perspective de l'escalier



Ajout de la navette dans le ciel – suppression de la ligne noire dans la verrière



Ajout de traces, pour une ambiance plus sombre – Amélioration de la perspective du poteau

# Outils de développement

Le développement de **Fugitif 31** est fait sous Linux, principalement avec **SjASMPlus**. Le gros intérêt de cet assembleur est d'être scriptable en **Lua**, ce qui permet de créer des macros très puissantes, en particulier pour *parser* les fichiers de *gameplay* (cf plus haut).

Un gros [Makefile](#) de près de 500 lignes permet de générer automatiquement tout le code, de la partie assembleur à la génération du DSK, en passant par la conversion de la musique, des images **multipaint**, ou de l'extraction des zones *clickables*, sans devoir tout reconstruire à chaque modification d'un fichier. En particulier, la compression des images ou des données associées à chaque lieu avec [zx0](#) n'est refaite que si nécessaire, car très gourmande en temps. De même que *parser* la description du GUI en **Lua**.

Concernant les zones *clickables*, elles sont créées dans [krita](#). Le graphe de base est automatiquement convertie et intégré depuis le fichier **multipaint**, et les zones sont définies sous forme de calques, par dessus.

Plusieurs scripts [Python](#) permettent les diverses transformations entre formats. C'est également un script **Python** qui crée le fichier DSK final, où chaque partie de code (moteur, images, données associées, musique...), est écrite directement sur les secteurs. Le DSK est au format *extended*: 2 faces, 80 pistes par face, et 10 secteurs par pistes. Soient 800ko au total. Je n'ai pas prévu de version sur disquettes 3" ; pour jouer à **Fugitif 31**, il faudra donc un *Gotek*, ou, pour les plus *geeks*, un lecteur 3"1/2 (et faire le transfert depuis le fichier DSK sur une vraie disquette).

Pour la simulation, c'est [RetroVirtualMachine](#) (v2.0-BETA 1 – r7) qui a été retenu, pour sa facilité de pilotage depuis un **Makefile**. Ce n'est certainement pas le plus précis au niveau simulation, mais son système multi-fenêtres de débogage est vraiment très pratique (bien que perfectible).

En fonction de la demande, **Fugitif 31** pourra être traduit en anglais et en espagnol, le support multilingues ayant été prévu dès le début.

## Conclusion

Si vous ne connaissez pas le jeu original, sorti en 1991, vous pouvez y jouer, cela ne divulguera en rien l'expérience avec **Fugitif 31**, les énigmes de ce dernier étant complètement revues. Et même si les graphes restent les mêmes, leur agencement est un peu différent. Cela vous permettra également d'apprécier la jouabilité de **Fugitif 31** par rapport à celle du jeu d'origine : plus d'icônes lourdingues, de mort subite, ni de *soft lock* ! L'excellent billet "[Why Adventure Games Suck](#)", de **Ron Gilbert**, a été d'une grande aide.

Je prends beaucoup de plaisir à développer ce *reboot*, et si cela fait maintenant près de 5 ans que je travaille sur ce projet, c'est parce que le voyage est presque plus important que la destination. J'espère que vous prendrez autant de plaisir à y jouer... lorsqu'il sera fini ^\_^

PS : j'ai déjà une autre idée de jeu *point'n'click* en tête, après **Fugitif 31** !