==========
* Documentation des fonctions du Happy C64 *
==========

*** Version du 16/10/2020 ***

Pour la librairie 0,1,010

*** Crédit ***

Codage de happy C64 : Jean Monos
Aide divers : Eric Boez.

*** Caractéristique du C64 ***

Mémoire Ram : 64ko
Résolution écran : 320 x 200 pixel
Résolution des tiles : 8x8 pixel (soit 8x8 points en mono couleur, ou 4x8 points logique en multicouleur. (point doublé en largeur pour avoir un tile de 8x8 pixel)
Nombre de tile à l'écran : 40x25 tiles (1000)
Nombre de sprite machine : 8
Taille d'un sprite : 24x21 pixel (12x21 point en multicouleur doublé en largeur)
Fonction des sprites : Position X,Y,Zoom x2 largeur et hauteur, detection de collision, Possibilité de passé derrière un tile.
Scrooling ; Décalage de l'écran de 0 a 7 pixel verticalement ou horizontalement.
Port joytick : Deux entrées. Haut/Bas/Gauche/Droite et bouton feu
Musique : SID avec trois voix !

*** Vocabulaire ***

Pattern : C'est tout simplement l'encodage en octet d'un élément graphique. (Tiles ou Sprite). Un Tiles carré peut être encodé avec les valeurs binaire suivant ! 0B11111111,0b10000001,0b10000001,0b10000001,0b10000001,0b10000001,0b10000001,0B11111111

Tiles: Un tile c'est un élément graphique affichable sur l'écran dans un quadrillage. Il peut représenter une police de caractère, (texte), ou des éléments du décore. On peux utiliser le terme de character , tuile ou caractère.

Tilemap : C'est la représentation en mémoire de l'écran. Elle se situe dans la mémoire écran (screen mémorie) en organisation linéaire. Le premier octet de la mémoire écran c'est la position 0,0 (en case) et la valeur contenu dans cette octet représente l'index du tile à afficher à cette endroit.(La taille de la tilemap fait 1000 octets)

Sprite : Ou Lutin, MOB (Movable Object Block), BOB, sont des éléments graphiques qui peuvent être placé au pixel près sans effacé ce qui se trouve à l'arrière. Le C64 gère 8 sprites simultanément de 24x21 points. (ou 12x21 points logique doublé sur la largeur en mode multicolore).

==========
* Le VIC (Video) *
==========

* Configurer la plage de lecture du VIC *

Le Vic est le co-processeur graphique du commodore 64. Il ne peut adresser que 16ko de mémoire. Il faut donc le configurer pour connaître la plage possible de lecture du Vic dans la Ram. (Il n'a pas sa propre mémoire contrairement a d'autre machine (MSX, et 99 % des consoles de jeu video 8/16 bitouné.)

void *set_vic_bank(**unsigned char** id_bank);*

id_bank est une valeur comprise entre 0 et 3 pour avoir un choix de 4 banks. (16ko*4 = 64ko le compte est bon.) Ceci dit plus on choisis une bank élevé, plus nous sommes en Haut de la Ram à l'adresse $0000. Des macros existent pour plus de clarté.

Macro des Banks de mémoire du VIC

```
#define VIC_BANK_0   3   // Adresse $0000 -> $3FFF * Valeur par défaut
#define VIC_BANK_1   2   // Adresse $4000 -> $7FFF * La Rom Tiles n'est pas dispos.
#define VIC_BANK_2   1   // Adresse $8000 -> $B000
#define VIC_BANK_3   0   // Adresse $C000 -> $FFFF * La Rom Tiles n'est pas dispo
```

* Activer/Désactiver l'affichage video *

La fonction ***void*** *screen_on()* permet d'activer l'affichage vidéo.
La fonction ***void*** *screen_off()* permet de désactiver l'affichage vidéo. Le background prend provisoirement la couleur du border.

* Modifier l'emplacement de la mémoire écran *

Le C64 réserve "1008 octets pour stocker la tilemap(1000 octet) et les pointeurs de sprite (8 octets). L'adresse de la mémoire écran au démarrage, se trouve à l'adresse 1024. ($0400) Il est possible de choisir un autre emplacement pour la mémoire écran.

void *Set_adresse_screen_memory(**unsigned char** screen_memory_pointeur);*

screen_memory_pointeur est un multiple de 16. Une série de macro qui correspond au adresse mémoire possible est prévus. Voici les défines utilisable.
Note : Ceci est un offset par apport à l'adresse du VIC.
Par exemple si le vic II est branché entre $8000. SM_0400 placera le screen memory en $8400.

```
#define SM_0         0  // Adresse $0
#define SM_0400     16  // Adresse $0400 (par défaut)
#define SM_0800     32  // Adresse $0800
#define SM_0C00     48  // Adresse $0C00
#define SM_1000     64  // Adresse $1000
#define SM_1400     80  // Adresse $1400
```

#define SM_1800	96 // Adresse $1800
#define SM_1C00	112 // Adresse $1C00
#define SM_2000	128 // Adresse $2000
#define SM_2400	144 // Adresse $2400
#define SM_2800	160 // Adresse $2800
#define SM_2C00	176 // Adresse $2C00
#define SM_3000	192 // Adresse $3000
#define SM_3400	208 // Adresse $3400
#define SM_3800	224 // Adresse $3800
#define SM_3C00	240 // Adresse $3C00

*** Attendre le début du Vblank ***
----
void *wait_vbl();* permet d'attendre le début du retour du balayage écran. (Le V blank)

unsigned int *get_raster();* permet de connaître à qu'elle ligne se situe le faisceaux.

*** Activer/Désactiver les interruptions ***
----
set_interruption_on()
set_interruption_off()

Active ou Désactive les interruptions.
(Note : Les routines d'interruption et Raster pas au point)

*** Modifier la taille d'affichage ***
----
void set_38_columns(**void**);
void set_40_columns(**void**);

Permet de passer en mode 38 (utile pour un scrolling Horizontale) ou 40 tiles en largeur.

void set_24_raws(**void**);
void set_25_raws(**void**);

Permet de passer en mode 24 (utile pour un scrolling verticale) ou 25 tiles en hauteur.

*** Scrolling ***
----
void set_scrolling_horizontal(**signed char** Scroll_X);
void set_scrolling_verticale(**unsigned char** Scroll_Y);

Permet de scroller l'écran horizontalement ou / et verticalement. Scroll_X et Scroll_Y accepte une valeur comprise ente 0 et 7. (On scroll que d'un tiles au pixel)

*** Modifier la couleur du border et du Backgounrd ***

Sur C64 il y a une bordure d'écran qui possède une couleur.

void *set_color_border (**unsigned char** color_id);*
Cette fonction permet de choisir la couleur à afficher dans le border.

Il est possible aussi de choisir la couleur du fond. (Paper/background)
C'est la couleur qui sera affiché quand un pixel d'un character et d'un sprite n'est pas activé au même endroit. (Couleur transparente).

void *set_color_background (**unsigned char** color_id);*

*** Récupérer si le commodore est en PAL ou NTSC ***
---

unsigned char get_system(); permet de connaître le mode d'affichage du C64. (0 en NTSC et 1 en PAL)

==========

* Gestion des tiles *
==========

Les tiles sont des éléments graphiques de 8x8 pixels qui s'affiche à l'écran sur un quadrillage invisible. Différents type de nom sont données au tiles. (font,charactère, characters,tuile...) mais c'est belle et bien la même chose. Il permette de représenter les graphismes de votre jeu. (Murs, sol, porte…)
Le Commodore possède un jeu de tile en Rom, mais il est possible de créer vos propre set graphique.
Une ligne d'un tile est contenu dans un octet. (8 bits), chaque bit en mode normale représente un point. (allumé ou non) et comme il y a 8 lignes, il faut donc 8 octets pour représenter les graphismes d'un tile. L'organisation des 8 octets se nomme donc un pattern !

Le c64 possède donc des patterns près enregistré qui permet d'afficher les lettres quand on écrit en basic. Elle se situe à l'adresse $1000. Mais il est possible de choisir un autre emplacement de la ram qui doit se situer bien sur dans la plage lisible du VIC.

*** Modifier l'emplacement de lecture des tiles ***
----
void *set_location_character(**unsigned char** id) ;*
Cette fonction permet de choisir l'emplacement de lecture des tiles. Utilisez les macros suivants :
Note : Tout comme le screen Memory, l'adresse est un offset par apport à l'adresse de départ du VIC II.

```
------------------------------------------------------------
ID : Adr Decimal : Adr Hexadecimal : Note     ------------------------------------------------------------
0 :     0      $0000-$07FF  ------------------------------------------------------------
2 :  2048      $0800-$0FFF (Semble être le plus fréquent)  ------------------------------------------------------------
4 :  4096      $1000-$17FF (Rom Bloc 0 et 2)  ------------------------------------------------------------
6 :  6144      $1800-$1FFF (Rom Bloc 0 et 2)  ------------------------------------------------------------
8 :  8192      $2000-$27FF   ------------------------------------------------------------
10 : 10240      $2800-$2FFF   ------------------------------------------------------------
12 : 12288      $3000-$27FF   ------------------------------------------------------------
14 : 14336      $3800-$3FFF   ------------------------------------------------------------
```

----
* Modifier l'adresse interne de tilemap *

void *set_adresse_tilemap(unsigned int adresse*
Permet de gérer manuellement la variable général interne du sdk pour linker le screen memory qui est utilisé par les fonctions draw*_character() et draw_full_character().*
Techniquement à chaque changement de bank du vicII et du pointeur de screen memory en passant par les fonctions du sdk, cette variable est recalculé pour bien pointer sur le screen memory.

*** Transférer des patternes au bonne endroit ! ***

Void *set_data_charcter(**unsigned int** adr_cible,**unsigned char*** data_character,**unsigned char** nb_pattern) ;*

Cette fonction permet de transférer des données d'un tableau à l'endroit voulu de la mémoire (donc l'adresse choisis pour mémoriser les tiles.)
adr_cible c'est l'adresse ou va se situer vos tiles (0x0800 par exemple)
data_character c'est le nom de votre tableau que contient les donnés des patterns.
nb_pattern c'est le nombre de pattern à copier/coller à l'endroit que vous voulez. (Et non le nombre d'octet)

exemple :

load_pattern(0x800,data_character,1) ;

*** Afficher un tile à l'écran ***

La résolution du commodore 64 est de 320 x 200 pixel. Ce qui permet d'afficher 40 tiles en largeur et 25 tiles en hauteur.
La mémoire écran permet de mémoriser l'index du tiles à afficher à l'écran. (Entre 0 et 255). Et ce qui est cool c'est que l'encodage des tiles est linéaire. (1er octet de la mémoire écran = position 0,0. 2em octet = position 1,0…

***void** draw_character (**unsigned char** position_x,**unsigned char** poition_y,**unsigned char** id_character);*
Cette fonction permet donc de poser un tile à l'écran en choisissant les coordonné X et Y. (En case pas en pixel, un tile ne se pose pas au pixel près sur commodore 64)

*** Modifier la couleur d'une case ***

L'affichage de la couleur d'un tile est définie dans son pattern. Chaque point du tile (représenter par un bit) est soit éteint (0) ou allumé (1). Et c'est tout. Pour connaître la couleur afficher par un tiles il faut se repérer à la couleur associée à sa case . (Un espace mémoire dans la ram permet de mémoriser tout ça)

***void** set_color_map(**unsigned char** position_x, **unsigned char** position_y,**unsigned char** color_id);*
On choisir la case au coordonnée X et Y et le numéros de la couleur.

Pour la couleur voici les macros qui peuvent être utilisés.

```
#define C_BLACK          0
#define C_WHITE          1
#define C_RED            2
#define C_TURQUOISE      3
#define C_PURPLE         4
#define C_GREEN          5
#define C_BLUE           6
#define C_YELLOW         7
#define C_ORANGE         8
#define C_BROWN          9
#define C_LIGHT_RED      10
#define C_GREY           11
#define C_GREY_2         12
#define C_LIGHT_GREEN    13
#define C_LIGHT_BLUE_2   14
#define C_GREY_3         15
```

*** Remplire la color ram d'une même couleur ***

La fonction *void cls_color_ram(unsigned char color); permet tout simplement de remplir en totalité la colors ram avec la couleur de votre choix.*

*** Afficher un tile et choisir sa couleur ! ***

***void** draw_full_character(**unsigned char** position_x, **unsigned char** position_y, **unsigned char** id_character,**unsigned char** color_id);*
Cette fonction permet de poser un tile à l'endroit voulu et de paramétrer en même temps la couleur associé à la case.

*** Tirer des traits avec un tiles ! ***

Les deux fonctions :
void draw_character_line_V(unsigned char px,unsigned char py,unsigned char size, unsigned char id_character,unsigned char color);

et

*void draw_*character*_line_H(unsigned char px,unsigned char py,unsigned char size, unsigned char id_*character*,unsigned char color);*

Permette de multiplier un tile sur la longueur ou la largeur sur la map.
(Utile pour les hud ? A vous de voir)
PX : Position X de départ (en case)
PY : Position Y de départ (en case)
Size : Nombre de fois que le tile va être dupliqué.
Id_character : Index du character.
Color : Couleurs à mémorier dans la color map.

*** Mode Multicolore ***

SET_MULTICOLOR_MODE_ON;

Cette fonction permet de passer en mode multicolore pour les tiles.

Chaque point d'un tile peut prendre une des 4 configurations suivante:
Afficher la couleur transparente du Background. (%00)

Afficher la couleur contenu dans Background 1.(%01)
Afficher la couleur contenu dans Background 2.(%10)
Afficher la couleur de la case.(%11) (Mais que sur les 8 premiers couleurs du nuancier)

Ce qui veux dire qu'un tile peux avoir 4 couleurs. (Avec des contraintes de trois couleurs général à tous les tiles et la couleur de la case qui cette fois si ne peux être que les 8 première nuances de la palette).

La pattern d'un tile est toujours encodé sur 8 octets. En mode multicolore on peux définir que des tiles de 4x8 points, mais le tile est doublé en largeur pour arriver à 8x8 pixels. (Donc on fait des points de 2x1 pixels !)

SET_MULTICOLOR_MODE_OFF;
Permet de désactiver le mode multicolor**.**

---
* Mode Étendu *

SET_EXTENDED_BACKGROUND_COLOR_ON;
Cette fonction permet d'activer le mode Etendu du C64.

Le mode étendu permet d'afficher une des 4 couleurs au "fond" + la color ram du bloc 8x8 pixel.
La gestion du fond se fait dans les 2 dernier bit du screen memory.
00xxxxxx : La couleur du background.
01xxxxxx : La couleur du background 1.

10xxxxxx : La couleur du background 2.
11xxxxxx : La couleur du background 3.

Ce qui laisse que 6 bits pour choisir l'index du character. (64 possibilité)

SET_EXTENDED_BACKGROUND_COLOR_OFF
Cette fonction permet de désactiver le mode étendu du C64.

*** Mode Bitmap ***

SET_STANDARD_HIGHT_RESOLUTION_BMM_ON;
Active le mode Bitmap

SET_STANDARD_HIGHT_RESOLUTION_BMM_OFF;
Desactive le mode bitmap;

*** Modifier les couleurs du Background 1 , Background 2 et Background 3 ***
----
void *set_color_background_1 (**unsigned char** color_id);*
void *set_color_background_2 (**unsigned char** color_id);*
void *set_color_background_2 (**unsigned char** color_id);*

Les trois fonctions permettent de choisir les couleurs associées au background 1 et background 2 des tiles en mode multicolore + le background 3 réservé en mode Extended

==========
* Les Sprites *
==========

Les sprites (ou lutin,MOB..) sont des graphismes qui peuvent être placé au pixel près sur l'écran. Elles ont la particularités de ne pas effacer le background. C'est le Vic qui gère les sprites. Le commodore 64 peut gérer et afficher 8 sprites à l'écran. Chaque sprite à des propriétés. (Position X, Position Y, activer ou non le sprite à l'écran, s'afficher derrière tiles, doublé la taille en hauteur un sprite, doublé la taille en largeur, tester si le sprite est en collision avec un autre sprite ou tiles, le passer en mode multicouleur, et choix de la couleur personnelle du sprite.
Note : A cause de l'overlay, pour être visible, le sprite doit se trouver en px : 24 et py 50

*** Copier les datas d'un sprites contenu dans un tableau ***

Tous comme les tiles, la fonction suivante permet de copier un ou plusieurs sprite(s) contenu dans un tableau à partir d'une adresse définie.

void set_sprite_data(**unsigned int** adr_cible,**unsigned char*** adr_data,**unsigned char** nb_sprite);

adr_cible c'est l'adresse mémoire de destination.
adr_data c'est le nom du tableau (ou pointeur) ou les données que représente le pattern d'un sprite est mémorisé.
nb_sprite : nombre de sprite contenu dans le tableau à copier.
(Un sprite est encodé sur 64 octets).

*** Configurer le pointeur de pattern d'un sprite ***

Le pattern d'un sprite est encodé sur 64 octets. Le VIC à besoin de connaître l'emplacement de départ du pattern.(La suite d'octet qui représente graphiquement un sprite) Celui si doit se trouver dans la plage d'adresse accessible au VIC.(Dans sa bank)
Le pointeur est défini pour chacun des 8 sprites. Il se situe dans les 8 derniers octets de la mémoire écran.

Pour configurer le pointeur de pattern au bonne endroit c'est simple. La formule c'est valeur du pointeur (0-255) * 64 + adresse de départ du VIC.

La fonction suivante permet de configurer le pattern.

void set_sprite_pointers(**unsigned char** id_sprite,**unsigned char** value);

id_sprite c'est le numéro du sprite entre. (0-7)
value c'est la valeur de base du calcule (0-255)

*** Activer un sprite à l'écran ***
----
void show_sprite(**unsigned char** id_sprite);
void hide_sprite(**unsigned char** id_sprite);

show_sprite() permet d'afficher le sprite à l'écran. hide_sprite() permet de le rendre invisible à l'écran.
id_sprite c'est l'index du sprite entre 0 et 7.

*** Configurer la position d'un sprite à l'écran ***
----
void draw_sprite(**unsigned char** id_sprite,**unsigned int** position_x,**unsigned char** position_y);

Permet de configurer la position d'un sprite à l'écran.
id_sprite : C'est l'index du sprite à configurer. (0-7).
position_x : Ce sont les coordonnées X du sprite à l'écran.
position_y : Ce sont les coordonnées Y du sprite à l'écran.

Note : La position x d'un sprite est contrôle sur 9 bits. La fonction s'occupe d'activer ou non le 9em bit de contrôle pour dépasser les 255 pixels à l'écran.

*** Doubler la hauteur d'un sprite ***
----

void double_height_sprite_on(**unsigned char** id_sprite);
void double_height_sprite_off(**unsigned char** id_sprite);

double_height_sprite_on() permet d'élargir en hauteur un sprite par 2.
double_height_sprite_off() permet de revenir à la normal dans la hauteur.

*** Doubler la largeur d'un sprite ***
----

void double_height_sprite_on(**unsigned char** id_sprite);
void double_height_sprite_off(**unsigned char** id_sprite);

double_width_sprite_on() permet d'élargir en largeur un sprite par 2.
double_width_sprite_off() permet de revenir à la normal dans la largeur.

*** Priorité d'un sprite par apport au tile ***
----

void sprite_priority_on(**unsigned cha**r id_sprite);
void sprite_priority_off(**unsigned char** id_sprite);

sprite_priority_on() permet de faire passer le sprite "derrière" les tiles.
sprite_priority_off() permet de faire passer le sprite devant les tiles.

*** Detecter les collisions d'un sprite ***
----
get_collision_sprite()
get_collision_character()

une collision se produit quand le sprite en question se superpose à un pixel non transparent d'un sprite ou d'un tile (character). La fonction renvois 1 octet. Et en fonction si le bit de l'octet est 0 (pas de collision) ou 1 (collision) on peu connaître qu'elle sprite est en collision !

Bit 7 = sprite 7 en collision
Bit 6 = sprite 6 en collision
Bit 5 = sprite 5 en collision
Bit 4 = sprite 4 en collision
Bit 3 = sprite 3 en collision
Bit 2 = sprite 2 en collision
Bit 1 = sprite 1 en collision

Bit 0 = sprite 0 en collision

*** Mode Multicolore pour les sprites ***
---

void set_sprite_multicolore_on(**unsigned char** id_sprite);
void set_sprite_multicolore_off(**unsigned char** id_sprite);

Tous comme les tiles, les sprites peuvent passer en mode multicolore. (Et individuellement). Dans ce mode il faut tous comme les tiles, 2 bits pour afficher un point. Ce qui permet 4 possibilités. (La couleur transparente, la couleur individuel du sprite, et deux couleurs choisis généralement pour les sprites.)

set_sprite_multicolore_on() active le mode sur id_sprite.
set_sprite_multicolore_off() désactive ce mode sur id_sprite.

*** Modifier les couleurs liés au sprite ***
---

void set_color_sprite(**unsigned char** id_sprite,**unsigned char** color_id);
void set_sprite_color_1(**unsigned char** color_id);
void set_sprite_color_2(**unsigned char** color_id);

set_color_sprite permet de choisir la couleur d'un sprite individuellement. C'est la couleur propre du sprite.

set_sprite_color 1 et 2 sont les deux couleurs choisis pour l'ensemble des sprites.

==========
* Les commandes *
==========

*** Tester les deux joystick ***

----

get_joystick_1()
get_joystick_2()

get_joystick renvois un octet pour tester si une direction du joystick ou le bouton feu est enclenché.
Utilisez les macros pour tester l'état du manche à ballet !

```
----------
#define UP    1<<0
----------
#define DOWN  1<<1
----------
#define LEFT  1<<2
----------
#define RIGHT 1<<3
----------
#define FIRE  1<<4
----------
```

Ce qui fait par exemple

```
if (get_joystick_1() & FIRE)
{
  // Votre code si le bouton feu du joystick 1 est utilisé...
}
```

*** Tester le clavier ***
----

get_keyboard_key()

cette fonction permet de récupérer une valeur de la touche du clavier utilisé. Voir le tableau de macro à la fin du document pour utiliser les défines approprié au touche.

Exemple :

```
if (get_keybord_key()==KEY_A)
{
  // Votre code si la touche A est utilisé.
}
```

*** La mémoire ***

*** Peek et Poke ***

PEEK(addr);
POKE(addr,val);
PEEKW(addr);
POKEW(addr,val);

4 petites fonctions (Define en réalité) qui permetent de taper dans la mémoire du C64

* Valeur 8 bits *

PEEK permet de lire 1 octet contenu dans l'adresse mémoire choisi.
POKE permet d'écrire 1 octet dans l'adresse mémoire choisi.

* Valeur 16 bits *

Les deux fonctions permettent de travailler sur des valeurs comprissent entre 0 et 65535

PEEKW permet de lire 2 octets contenu dans l'adresse mémoire choisi.
POKEW permet d'écrire 2 octets dans l'adresse mémoire choisi.

*** Desactiver / Activer la rom le basic ***
---

set_loram_basic()
set_loram_ram()

set_loram_ram() permet de désactiver le basic à l'adresse 40960 ($A000). Une plage de 8ko est récupéré pour vos donnés.

set_loram_basic permet de récupérer le basic.

Note : Normalement la bank du basic est désactivé de base avec CC65.

*** Désactiver / Activer la rom Kernal ***

```
 set_hiram_ram();
 set_hiram_kernal();
```

set_hiram_kernal() active le kernal du Commodore 64 (utiles pour détecter les touches du clavier et le joystick.

set_hiram_ram() : permet de récupérer 8ko de mémoire à l'adresse 57344 ($E000). Attention à bien switcher correctement cette bank de donnée sous peine de bug ! (Je conseille au débutant de ne pas toucher à cette partie et de garder le Kernal activé)

---
* Configuration de la mémoire du C64 *

La C64 peux avoir de multiple configuration pour sa mémoire. Sa Ram est découpé en morceau.
Deux gros morceaux de 16ko de ram au début de celle si est disponible : $000 => $3FFF et $4000 => $7FFF.
Avec le SDK et CC65, en gros les programmes se charge en $800 et la mémoire video se trouve en $400.
Une autre partie de 8ko se trouve en $8000 => $9FFF
Quand une cartouche est inséré, son programme se trouve dans cette partie. (Si elle est en auto_start!)

Une autre ban de 8ko se trouve à l'adresse $A000 =>$BFFF.
Elle est branché sur la Rom Basic, mais elle se désactive pour récupéré de la RAM !
(Nativement CC65 désactive le Basic !)

4ko existe en $C000 => $CFFF Pratique pour encore mémoriser des data ! (Notons que de base CC65 utilise les deux derniers ko pour sa pile.)

Deux autre banks ou j'ai envie de dire pas touche à cela petit con existe :
$D000 => $DFFF qui est la gestion des i/o et charactère. (4ko
$E000 => $FFFF (8ko) qui est le kernal.

D'autre zone libre existe encore :
$42
$52
$00FB à $00FE (4 Bytes)
$02A7 à $02FF (89 bytes)
$313
$334 à $033B (8 Bytes)
$33C à 03FB(192 Bytes disponible mais c'est le buffer datasette)
$03FC-$03FF (4 Bytes)
$400 à $7FF (1024 Bytes) si le pointeur adresse screen memory n'est pas positionné à cette adresse.

Il peut être intéressant de passer le Vic dans à l'adresse $8000 de passer dans la mème occasion l'adresse du screen mémorie à l'offset 0 et le pointeur de character à $800.

```
set_vic_bank(1);
set_adresse_screen_memory(0);
set_location_character(2);
```

Le programme compilé par CC65 à $800
Ce qui fait le screen memory à $8000
Le pointeur de character à $8800
De la ram en $C000

Une organisation possible !

Autre information importante sur la gestion de mémoire native de CC65 !

- Les variables locales de vos fonctions se place dans la pile. Vous avez 2ko de ram.
- Les variables global, tableau,structure se place à la fin de votre de cotre. Même les variables / tableau en constante.

==========
* Le SID *
==========

Le Sid est le processeur sonore du C64. HappyC64 permet de sortir du son avec le Sid.

*** Modifier le volume sonore du C64 ***
----

void set_volume(**unsigned char** volume);

Permet de gérer le volume de sortie du SID. (0 et 15)

*** Jouer un Son ***

```
void set_sound(
            unsigned char voice, // Utiliser le macro VOICE_1 , VOICE_2 , VOICE_3
            unsigned char lb_freq,
            unsigned char hb_freq,
            unsigned char lb_pulse,
            unsigned char hb_pulse,
            unsigned char waveforme,// Utiliser le macro TRIANGLE,SAWTOOTH...
            unsigned char attaque_decay,
            unsigned char sustain_release

           );


  // * Macro pour le canal *
  #define VOICE_1    0
  #define VOICE_2    7
  #define VOICE_3   14

  // * Macro pour le Type du canal
  #define TRIANGLE     17
  #define SAWTOOTH     33
  #define PULSE        65
  #define NOISE       129
```

==========
*** Divers ***
==========

*** Générateur Pseudo Aléatoire ***
----

unsigned char get_rnd(**unsigned char** nombre_max);

Fonction qui retourne une valeur comprise entre 0 et nombre_max.
(Utilise le générateur de bruit du SID)

---
* Decompression RLE *

HappyC64 possède une fonction simple pour décompresser des "data" au format RLE avec une contrainte ! Vos datas RLE doivent se finir par un octet qui vaut 0 pour mettre fin à la routine de décompression.

Le format RLE est simple : Voici une série d'octet au format RLE

2,4,1,0,7,5,4,4,0

Ce qui fait une fois décompressé !
4,4,0,5,5,5,5,5,4,4,4,4

void rle_decrompression(unsigned int source,unsigned int destination);

Source est l'adresse de lecture des datas compressé.
Destination c'est l'adresse de départ d'enregistrement des datas qui sont décompressés.

---
* Compression RLE *

La fonction : *void rle_compression(unsigned int source,unsigned int destination,unsigned int size)*
Permet d'appliquer une simple compression RLE. A la fin de la routine, 1 octet supplémentaire est appliqué avec la valeur 0.

Source : L'adresse des datas à compresser.
Destination : L'adresse source ou va se ranger les datas compresser.
SIZE : Le nombre d'octet à lire (source) pour la compression.

==========
* Cassette et Disquette *
==========

Note : Le retour des erreur n'est pas encore opérationnel en 0,0,5,0

==========
** Sauvegarder des datas dans un fichier **
==========
unsigned char *save_file(unsigned char*name,const void* buffer, unsigned int size,unsigned char device);*

Cette fonction permet de sauvegarder des données contenu dans un tableau (ou à partir d'une adresse mémoire) dans un fichier disquette (ou cassette).

#name : Le nom du fichier. un ,"option" permet de typer le fichier.
d : del
s : Sequenciel (seq)
u : USR
p : PRG (defaut)
l : REL
exemple : "data,s"
#buffer : pointeur ou Nom du tableau
#size : Taille du fichier à enregistrer. (Sur disquette 2 octets va s'ajouter au début du fichier)
#device : Id du device. 1 pour casette, 8 pour lecteur disquette 1, 9 pour lecteur disquette 2 ...

==========
** Charger des datas dans un fichier **
==========
unsigned int *load_file(const char*name, const void* buffer, unsigned char device);*

Cette fonction permet de charger les données binaire d'un fichier dans un tableau (ou à partir d'une adresse mémoire)

#name : Le nom du fichier. un ,"option" permet de typer le fichier.
d : del
s : Sequenciel (seq)
u : USR
p : PRG (defaut)
l : REL
exemple : "data,s"

#buffer: pointeur ou nom du tableau de buffer.
#device : Id du device. 1 pour casette, 8 pour lecteur disquette 1, 9 pour lecteur disquette 2 ...

Note : Les deux octets du "header" ne sont pas enregistré dans le buffer.

Note :La Fonction renvois le nombre octet chargé. Si le nombre est égale à 0, alors il y a une erreur de chargement.
Erreur à récupéré avec get_error()

Exemple ouverture de fichier :

```
if (load_file("map1,s",(void*)0x8000,8)!=0)
{
  error = get_error() ;
}
```

==========
* Gestion des erreurs *
==========

La Commande *get_error()* permet de récupérer une erreur de chargements avec l'utilisation de la commande load_file()
Les erreurs possibles.

==========
* Un petit mode d'emplois sur les pointeurs *
==========

Les deux fonctions pour manipuler les datas dans un fichier utilise une adresse mémoire de départ pour la lecture, ou la sauvegarde. C'est utile pour charger/sauvegarder des datas à un emplacement voulu, le buffer demande un pointeur. Voici comment les gérer si vous n'avez pas l'habitude de cela.

Le nom d'un tableau est pointeur. Donc un tableau avec le nom buffer[] :
save_file("sauvegarde,s",buffer,128,8) ; sauvegardera dans le fichier sauvegarde.seq, 128 octets du tableau buffer sur la disquette…

On peux utiliser un pointeur avec une adresse définie (ou autre…)
unsigned char **buffer =(char**)0xC000

Dans ce cas la, en utilisant le mot buffer on sauvergarde 128 octet à partir de l'adresse 0XC000.

On peux placer une adresse directe dans la fonction aussi avec un cast !

Voici l'exemple avec 0xC000
save_file("sauvegarde,s",(void*)0xC000,128,8);

==========
*** Text Engine ***
==========

Le SDK permet d'afficher du texte. Ceci dit une petite préparation est à effectué. La représentation du texte dans la mémoire de character doit avoir une certaine organisation. - Il suit l'organisation ASCII
- Le premier élément c'est "l'espace".
- Vous n'êtes pas obligé de placer les majuscules et minuscules pour gagner de la place. Vous pouvez représenter des minuscules dans la partie majuscule mais vous devez écrire vos textes en majuscule !
- Vous devez garder le pattern du mode vidéo appliqué. Un A en mode standard n'aura pas la même gueule qu'un A en mode Multicolor !

==========
* Modifier le pointeur de texte *
==========
void set_text_pointeur(unsigned char pointeur);
Cette fonction vous permet de redéfinir la place du premier caractère (l'espace) dans votre plage mémoire de pointeur de caractère. Exemple à 0, votre jeu de police d'écriture est au début de la plage des caractères. A 5, le départ sera le 6em caractères.

==========
* Afficher un texte à l'écran *
==========
void draw_text(unsigned char px,unsigned char py,unsigned char* text,unsigned char color);

Cette fonction permet d'afficher du texte à l'écran.
PX et PY sont la position en case du début du texte.
Text c'est le texte à afficher(entre parenthèse ou d'un buffer)
color : c'est la couleur de la case du texte. (Color ram).

Exemple : draw_text(0,0,"Hello World",5);
Un Hello World jaune s'affiche en 0,0

==========
* Afficher une valeur à l'écran *
==========
void draw_valeur_8 (unsigned char px,unsigned char py,unsigned char valeur,unsigned char color);

void draw_valeur_16 (unsigned char px,unsigned char py,unsigned int valeur,unsigned char color);

Les deux fonctions sont pratiquement identique. Elles permettent d'afficher une valeur à l'écran. La première permet d'afficher une valeur sur 1 octet. (0 à 255), la deuxième fonction permet d'afficher une valeur sur 2 octets. (0 à 65535).
Les deux fonctions peuvent prendre aussi comme paramètre des variables de type unsigned char pour la première ou unsigned int pour la 2nd.

==========
* Index des touches du clavier *
==========

// ==========
**// ** Lettre de l'alphabet **

```
// ============================
        #define KEY_A 10
        #define KEY_B 28
        #define KEY_C 20
        #define KEY_D 18
        #define KEY_E 14
        #define KEY_F 21
        #define KEY_G 26
        #define KEY_H 29
        #define KEY_I 33
        #define KEY_J 34
        #define KEY_K 37
        #define KEY_L 42
        #define KEY_M 36
        #define KEY_N 39
        #define KEY_O 38
        #define KEY_P 41
        #define KEY_Q 62
        #define KEY_R 17
        #define KEY_S 13
        #define KEY_T 22
        #define KEY_U 30
        #define KEY_V 31
        #define KEY_W 9
        #define KEY_X 23
        #define KEY_Y 25
        #define KEY_Z 12

// ============================
// ** Valeurs numeriques    **
// ============================
        #define KEY_0 35
        #define KEY_1 56
        #define KEY_2 59
        #define KEY_3 8
        #define KEY_4 11
        #define KEY_5 16
        #define KEY_6 19
        #define KEY_7 24
        #define KEY_8 27
        #define KEY_9 32

// ============================
// ** Touche Divers        **
// ============================
        #define KEY_L_ARR 57
        #define KEY_CLR 51
        #define KEY_DEL 0
        #define KEY_RET 1
        #define KEY_DN 4
        #define KEY_RT 2
        #define KEY_STOP 63 // Echape
        #define KEY_SPC 60 // Espace
        #define KEY_EMPTY 64 // Pas de touche

// ============================
// ** Touches Arithmétiques  **
// ============================
        #define KEY_PLUS 40
        #define KEY_MOINS 43
        #define KEY_DIVISER 48
        #define KEY_MULTIPLIER 49

// ============================
// ** Touches de Fonctions   **
// ============================
        #define KEY_F1 4
        #define KEY_F3 5
        #define KEY_F5 6
        #define KEY_F7 3
```