

PROGRAMMING COLECOVISION GAMES

IN C LANGUAGE  
WITH HI-TECH C COMPILER  
UNDER WINDOWS/DOS ENVIRONMENT

BY DANIEL BIENVENU

Version 1-k

Last update: October 26, 2005

## Table of content

<b>TABLE OF CONTENT.....</b>	<b>2</b>
<b>INTRODUCTION.....</b>	<b>8</b>
<b>SETUP YOUR DEVELOPMENT ENVIRONMENT.....</b>	<b>9</b>
<b>MEMORY.....</b>	<b>10</b>
ROM (Read Only Memory).....	10
RAM (Read-Write Memory).....	10
MEMORY MAP.....	10
RAM USED BY THE COLECOVISION BIOS.....	11
<b>INITIALIZING TABLES AND ARRAYS.....</b>	<b>12</b>
<b>DATA TYPES.....</b>	<b>13</b>
char.....	13
byte.....	13
int.....	13
unsigned.....	13
float.....	14
char [n].....	14
* (pointer).....	14
void.....	14
<b>SPECIAL DATA TYPES.....</b>	<b>15</b>
sprite_t.....	15
sound_t.....	15
score_t.....	15
<b>STRUCTURES (CUSTOM DATA TYPE).....</b>	<b>16</b>
typedef struct.....	16
<b>OPERATORS.....</b>	<b>17</b>
USUAL SET OF BINARY ARITHMETIC OPERATORS:.....	17
INCREMENT (++) AND DECREMENT (--)......	17
BITWISE OPERATORS.....	17
COMBINED OPERATORS .....	17
RELATIONAL OPERATORS.....	18
LOGICAL OPERATORS.....	18
<b>IF STATEMENT.....</b>	<b>19</b>
SIMPLE IF.....	19
IF ... ELSE.....	19
<b>LOOPS.....</b>	<b>19</b>
FOR LOOP.....	19
WHILE LOOP.....	19
DO WHILE LOOP.....	19
<b>SCREEN MODES.....</b>	<b>20</b>

## PROGRAMMING COLECOVISION GAMES

<b>CHARACTERS.....</b>	<b>21</b>
VIDEO MEMORY FOR CHARACTERS.....	21
CHARACTER PATTERN.....	22
EXAMPLE - SPACESHIP.....	22
UPLOAD CHARACTER SET.....	24
<b>SPRITES.....</b>	<b>25</b>
SPRITES COLOR.....	25
SPRITES LOCATIONS ON SCREEN.....	25
SPRITES PATTERN.....	25
8x8 SPRITE.....	25
16x16 SPRITE.....	26
SPRITES ROUTINES.....	26
<b>JOYSTICK (Hand Controller) .....</b>	<b>27</b>
JOYPAD .....	27
KEYPAD .....	27
<b>OTHER CONTROLLERS.....</b>	<b>28</b>
Super Action Controller.....	28
Expansion Module 2: Turbo Drive.....	28
Roller Controller.....	28
<b>SPINNERS VARIABLES.....</b>	<b>28</b>
<b>SOUND.....</b>	<b>29</b>
THE SOUND ROUTINES .....	29
update_sound (); .....	29
start_sound (sound_data,sound_priority); .....	29
sound_pointer = start_sound(sound_data, sound_priority); .....	29
stop_sound(sound_pointer); .....	29
sound_on();.....	30
sound_off();.....	30
play_dsound(sound_pointer, step);.....	30
THE WAY I ADD SOUNDS IN MY COLECO PROJECTS .....	31
<b>TOOLS.....</b>	<b>32</b>
WAV2CV.....	32
WAV2CVDS.....	33
I.C.V.G.M. v2.....	34
I.C.V.G.M. v3.....	35
BMP2PP.....	36
PP2C and PP2ASM.....	37
CVPAINTE.....	38
CCI - Coleco Compiler Interface.....	39
How to use CCI?.....	39
<b>LIBRARY: COLECO.....</b>	<b>40</b>
ROUTINES IN COLECO LIBRARY.....	40
rle2ram.....	40
rle2vram.....	40
put_vram .....	40
get_vram.....	40
fill_vram.....	40
put_vram_ex.....	40
put_vram_pattern.....	40

## PROGRAMMING COLECOVISION GAMES

set_default_name_table.....	41
vdp_out.....	41
screen_on.....	41
screen_off.....	41
disable_nmi.....	41
enable_nmi.....	41
update_sound.....	41
start_sound.....	42
stop_sound.....	42
sound_on.....	42
sound_off.....	42
delay .....	42
get_random.....	42
upload_ascii.....	42
utoa.....	42
sprites struct and table.....	43
update_sprites.....	43
check_collision.....	43
<b>LIBRARY: COLECO OPTIMIZED FOR 4K.....</b>	<b>44</b>
NEW ROUTINES IN THIS COLECO LIBRARY.....	44
play_sound.....	44
stop_sound.....	44
reflect_vertical.....	44
reflect_horizontal.....	44
rotate_90.....	44
<b>LIBRARY: GETPUT.....</b>	<b>45</b>
GETPUT.....	45
cls.....	45
get_char.....	45
put_char.....	45
center_string.....	45
print_at.....	46
pause .....	46
GETPUT 1.....	47
pause_delay.....	47
rnd.....	47
rnd_byte.....	47
str.....	47
show_picture.....	47
screen_mode_2_bitmap.....	48
screen_mode_2_text.....	48
upload_default_ascii.....	48
paper.....	48
load_color.....	48
load_namerle.....	48
load_patternrle.....	49
load_spatternrle.....	49
change_pattern.....	49
change_spattern.....	49
change_color.....	49
fill_color.....	50
change_multicolor.....	50
change_multicolor_pattern.....	50
choice_keypad_1 and choice_keypad_2.....	50

## PROGRAMMING COLECOVISION GAMES

updatesprites.....	50
sprites_simple.....	51
sprites_double.....	51
sprites_8x8.....	51
sprites_16x16.....	51
Set of "AND" masks for the joystick: UP, DOWN, LEFT, RIGHT and FIREs .....	51
wipe_off_down.....	51
wipe_off_up.....	51
New routines in Getput1 library in years 2003-2004.....	52
play_dsound.....	52
put_frame.....	52
get_bkgnd.....	52
load_colorrle.....	52
strlen.....	52
put_frame0.....	52
screen.....	53
swap_screen.....	53
put_at.....	53
fill_at.....	53
Extra routines in Getput library version 1.1.....	54
score_reset.....	54
score_add.....	54
score_str.....	54
score_cmp_lt.....	54
score_cmp_gt.....	54
score_cmp_equ.....	54
intdiv256.....	54
utoa0.....	54
load_ascii.....	55
rlej2vram.....	55
GETPUT-1 VIDEO MEMORY MAP.....	56
Screen Mode 2 Text - Video Memory Map.....	56
Screen Mode 2 Bitmap - Video Memory Map.....	56
<b>LIBRARY: C.....</b>	<b>57</b>
memcpy.....	57
memset.....	57
sizeof.....	57
switch case.....	57
<b>SHOW BITMAP PICTURE WITHOUT GETPUT 1.....</b>	<b>58</b>
<b>SHOW BITMAP PICTURE WITH GETPUT 1.....</b>	<b>60</b>
<b>FACES - SPRITES DEMO.....</b>	<b>61</b>
<b>REBOUND.....</b>	<b>63</b>
THE IDEA.....	63
THE PROGRAM.....	65
COMPILING.....	67
STILL BUGY?.....	71
CAN IT BE BETTER?.....	72
<b>SMASH - A VIDEO GAME VERSION OF REBOUND.....</b>	<b>73</b>
PADDLE GRAPHIC.....	75
THE PROGRAM.....	76

## PROGRAMMING COLECOVISION GAMES

<b>DEBUG EXERCISE.....</b>	<b>80</b>
A TEST PROGRAM TO DEBUG.....	80
SOLUTION.....	81
<b>OPTIMIZATION TRICKS.....</b>	<b>82</b>
Trick #1 : divide and multiply by using bit shifting.....	82
Trick #2 : a useful pointer .....	82
Trick #3 : fill up RAM with memset.....	83
Trick #4 : load_ascii VS upload_ascii VS upload_default_ascii.....	83
Trick #5 : do your own sprite routines.....	83
<b>THAT'S ALL?.....</b>	<b>84</b>
<b>APPENDIX A - MORE TECHNICAL INFORMATION.....</b>	<b>85</b>
<b>HARDWARE SPECIFICATIONS.....</b>	<b>85</b>
<b>CARTRIDGE (ROM) HEADER.....</b>	<b>86</b>
<b>SOUND GENERATION HARDWARE.....</b>	<b>87</b>
TONE GENERATORS.....	87
NOISE GENERATOR.....	87
CONTROL REGISTERS.....	88
SOUND DATA FORMATS.....	88
NOTES TABLE CONVERSION: FREQUENCIES (Hz) <-> HEX values.....	89
SCALES.....	89
<b>MARCEL's SOUND DATA FORMAT.....</b>	<b>90</b>
Sound Header.....	90
Sound Body.....	90
<b>VDP - VIDEO DISPLAY PROCESSOR.....</b>	<b>91</b>
REGISTERS.....	91
Control registers.....	91
Status register.....	91
VDP register access.....	92
NMI Non maskable interrupt.....	92
Screen modes.....	93
Mode 0 - Graphic I.....	93
Mode 1 - Text.....	93
Mode 2 - Graphic II.....	93
Mode 3 - Multicolor.....	93
<b>COLECO SCREEN MODE 1 (TEXT MODE).....</b>	<b>94</b>
<b>COLECO SCREEN MODE 0 &amp; 2 (GRAPHIC I &amp; II MODE).....</b>	<b>95</b>
<b>COLOR PALETTE.....</b>	<b>96</b>
<b>COLECO ASCII TABLE.....</b>	<b>97</b>
<b>APPENDIX B - ORIGINAL OS7' BIOS INFORMATION.....</b>	<b>99</b>

## PROGRAMMING COLECOVISION GAMES

<b>JUMP TABLE.....</b>	<b>99</b>
<b>OTHER OS SYMBOLS.....</b>	<b>100</b>
<b>MEMORY MAP.....</b>	<b>101</b>
COLECOVISION GENERAL MEMORY MAP.....	101
GAME CARTRIDGE HEADER.....	101
COMPLET OS 7' RAM MAP.....	102

## INTRODUCTION

Dear ColecoVision programmers,

This document shows you the basic of the ColecoVision capabilities with technical information, programming tools and simple codes in C language. This document is "a good starting point" to learn how to program ColecoVision games in C language like I did. However, don't hesitate to seek for more information and to ask questions to other ColecoVision programmers.

The first part of this document contains the basic concepts of the ANSI C language based on the Coleco library and the Hi-Tech C compiler. If you don't know how to program in ANSI C language, look for C programming (not C++ or C#) in the Internet.

The second part of this document talks about tools (for Windows) Marcel de Kogel and I programmed to speed up the development process.

The third part of this document is about specific libraries already made for the ColecoVision game development, and programming samples.

The annexe gives you technical information for programmers about the ColecoVision.

Before trying to do your first ColecoVision project, improve your programming skills first by testing all the concepts mentioned in this documentation before thinking of releasing new ColecoVision games.

This document can be used as a tutorial and a reference guide.

Have fun making new games for the ColecoVision game system! Good Luck! ☺

Best regards,

*Daniel Bienvenu*



### ***SETUP YOUR DEVELOPMENT ENVIRONMENT***

If you are using a Windows environment or a Linux with a great DOS emulator, you may have no problem to set up this programming environment.

First choice:

First, you need to download the Hi-Tech C compiler for CP/M (freeware).

Second, you need to download a CP/M emulator for your system. 22NICE for DOS is my personal choice but you need to read carefully the text files because you have to modify the Hi-Tech C compiler executables.

Third, you need to download the Coleco library by Marcel de Kogel. You have to extract files into the Hi-Tech C compiler directory.

For Windows users, you can avoid all these steps by downloading an archive named “z80.zip” from the web site.

Use your preferred text editor to code in C your Coleco projects. Make sure your C code is in a pure text file format before trying to compile it. My personal choice is NOTEPAD or Win32PAD.

You have two choices now to access to your Coleco environment:

- Create a shortcut on your desktop to run the CP/M emulator and go directly into the Hi-Tech C compiler directory.
- Use a GUI (front-end) to use this environment. I personally made one for windows named CCI (Coleco Compiler Interface). You just need to put the EXE file into your project directory to avoid using the command-line based interface under DOS.

Note: There is an online help section in the web site to show you how to use the compiler with a sample code.

Second choice:

Download and install the Hi-Tech C compiler for DOS (shareware) and try using the ColecoVision and Getput1 libraries with this environment. Because it's the same company, this solution may work just fine.

Third choice:

Download and install a cross C compiler for your system to be able to compile a binary file for the Zilog 80 processor. Download the libraries source code to adapt them for your cross-compiler.

### **MEMORY**

Before programming a ColecoVision project, you must learn about the memory "limits" for this game system.

#### **ROM (Read Only Memory)**

The binary code in the cartridge is named ROM because it's a read-only memory. The memory space for the game starts at 8000. Today, we have no problem using big memory capacity so we try to use all the 32K available: 8000 - FFFF. At 8000, there is a header. The header starts with 55 AA or AA 55. After this 2 bytes, there are hex values in the header to say where start the game, what to do if there is an interrupt (example: rst 08h, NMI), etc.

The ColecoVision BIOS start at the address 0000 and it's the first thing running in the ColecoVision game system. First, the BIOS check if a cartridge is inserted by looking for the first two bytes of the ROM (cartridge). If these two bytes are "55 AA" or "AA 55", the Coleco BIOS check the start address in the ROM header then start the game, otherwise, a default screen appear about how to insert a cartridge: "TURN GAME OFF BEFORE INSERTING CARTRIDGE OR EXPANSION MODULE". Second, the BIOS had many routines like sounds and sprites routines. Using the BIOS routines gives more free ROM space for the game itself but you have to be careful to not use address in RAM used by the BIOS.

By using the Coleco library by Marcel de Kogel, don't worry about the header of your Coleco project because it's already compiled into the "crtcv.obj" file.

#### **RAM (Read-Write Memory)**

The ColecoVision RAM is a bit weird. There is only 1K RAM (with a copy of itself at another memory location) and this RAM is not fully available if you use routines in the Coleco BIOS. The Coleco BIOS routines use some parts of the RAM at specific memory locations above 73B8. This is important to know to avoid memory corruption by using too big tables in RAM. Don't worry, normally, your first ColecoVision project will never use more than the free memory space available... except if you implement complex AI and/or big dynamic environment.

### **MEMORY MAP**

The ColecoVision BIOS starts at the address 0000 and ends at the address 1FFF.

The cartridge (ROM) starts at the address 8000 and ends at the address FFFF.

The read-write memory space is only 1K. The real addresses for the RAM are 7000-73FF. The RAM space addresses available in memory for your ColecoVision projects are 7000-73B8. The stack pointer is initialised at 73B9 (firsts instructions in BIOS) but the stack really started at 73B8, 73B7, 73B6, etc. If you need more RAM space, you can use the video memory.

## PROGRAMMING COLECOVISION GAMES

### RAM USED BY THE COLECOVISION BIOS

Thanks to Steve Bégin for all the informations (for expert only).

7020-702A: Used by BIOS sound routines  
73B9-73C2: Used but we don't know by which routines yet  
73C3: Used by video Read/Write routines. Copy of the video control register number 0.  
73C4: Used by video Read/Write routines. Copy of the video control register number 1.  
73C5: Seems to be unused by any BIOS routines (free)  
73C6: Used by BIOS sprite routines.  
73C7: Used by BIOS sprite routines.  
73C8-73C9: Used by BIOS to generate Random numbers (call 1FFD).  
73CA-73D2: Used by BIOS sprite routines.  
73D3-73D6: Used by BIOS timer routines.  
73D7-73EA: Used by BIOS joystick routines. (call 1FEB)  
73EB: Used by BIOS joystick routines. Value of spinner on port#1  
73EC: Used by BIOS joystick routines. Value of spinner on port#2  
73ED: Seems to be unused by any BIOS routines (free)  
73EE-73F1: Used by BIOS joystick routines. Raw joystick data from ports (Call 1F76).  
73F2-73F3: Used Address of sprites attribute in VRAM.  
73F4-73F5: Used by video Read/Write routines. Address of sprites pattern in VRAM.  
73F6-73F7: Used by video Read/Write routines. Address of screen image (NAME).  
73F8-73F9: Used by video Read/Write routines. Address of character pattern (PATTERN).  
73FA-73FB: Used by video Read/Write routines. Address of character color pattern (COLOR).  
73FC-73FD: Seems to be unused by any BIOS routines (free)  
73FE-73FF: Temporary used when a call at routine in 1fbe is made.

Ok, it's more information than you really need to know especially if you program your ColecoVision projects in C like me. The most important to know is you can't use the address 73B9-73FF in RAM for your own purpose. The only exception is the RAM used by the BIOS for the sound routines. Marcel de Kogel writes his own sound routines in the ColecoVision library so you can use the addresses 7020-702A for your game without any problem. The only problem is the data sound format is not the same between BIOS sound routines and ColecoVision library sounds routines. In this document, you will not find information about BIOS sound routines.

If you don't really know what is a stack, you may have a problem to understand why you can't really use all the free memory space 7000-73B8. If you program in C language, the Hi-Tech C compiler add some codes "pop" and "push" to stock information in the stack like the registers status. If you program in ASM, you have to add by yourself each "pop" and "push" instructions so you have more control but you have also more responsibility if your program doesn't run well. The stack is filled with data by decreasing first the stack pointer then by adding information. So, the real RAM space you can use depends on how big your stack can be. I think you must not use RAM address over 7300, otherwise you may have a problem of memory corruption.

## INITIALIZING TABLES AND ARRAYS

Tables in another C file are declared with the instruction "extern".

init.c

```
#include <coleco.h>

extern byte pattern[];
```

tables.c

```
#include <coleco.h>

byte pattern[] = {0xff,0xfe,0xfc,0xf8,0xf0,0xe0,0xc0,0x80};
```

Tables restricted in a C file are declared with the instruction "static".

init.c

```
#include <coleco.h>

static byte pattern[] = {0xff,0xfe,0xfc,0xf8,0xf0,0xe0,0xc0,0x80};
```

Tables in ROM are initialised directly outside functions.

init.c

```
#include <coleco.h>

byte pattern_rom[] = {0xff,0xfe,0xfc,0xf8,0xf0,0xe0,0xc0,0x80};
```

Tables in RAM are initialised inside functions by using other tables, or filled with a value.

init.c

```
#include <coleco.h>

byte pattern_ram[8];
byte pattern_rom[] = {0xff,0xfe,0xfc,0xf8,0xf0,0xe0,0xc0,0x80};

void init_pattern_ram(void) {
    memcpy(pattern_ram, pattern_rom, 8); /* Initialize pattern_ram with pattern_rom */
    memset(pattern_ram, 0x00, 8); /* Set the 8 bytes of pattern_ram table to 0x00 */
}
```

## ***DATA TYPES***

### **char**

Definition: character or short integer  
1 byte and signed  
Values : [ -128, 127 ]

Example:

```
char c = 'A';
```

### **byte**

Definition: byte or short unsigned integer  
1 byte and unsigned  
Values : [ 0, 255 ]  
This type is defined in the coleco.h file.

Example:

```
byte i = 200;
```

### **int**

Definition: integer  
2 bytes and signed  
Values : [ -32768, 32767 ]

Example:

```
int i = -1000;
```

### **unsigned**

Definition: unsigned integer  
2 bytes and unsigned  
Values : [ 0, 65535 ]

Example:

```
unsigned score = 1000;    print_at (10,0,str(score));
```

## PROGRAMMING COLECOVISION GAMES

### float

Definition: floating point  
2 bytes and signed  
Values : [ ??, ?? ]

Example:

```
float ratio = 10.0/4.0;
```

Note: Try to never use floating point variables.

### char [n]

Definition: character array or string  
Max 256 bytes long

Note: A string ends with the character '\0'.

Example:

```
char numbers[]={ '0','1','2','3','4','5','6','7','8','9' };
```

### \* (pointer)

Definition: pointer, address in memory where is the data  
2 bytes  
Values : [ 0000 , FFFF ] (0,65535)

Example:

```
int score1, score2;  
int *score;      /* pointer to integer value named score */  
score = &score1; /* set score pointer to score1 */  
char *msg;       /* char pointer, also used as a variable size string variable */
```

Note: A character pointer is not a character array.

### void

Definition: no defined type, void  
2 bytes by default

Example:

```
void *msg;
```

Note: This data type must be reserved for routines only.

### ***SPECIAL DATA TYPES***

The following data types are defined into specialized libraries, not included with the C compiler.

From Marcel's Coleco library

#### **sprite\_t**

Definition: Sprites table data format, consist of 4 bytes named: y, x, pattern and colour.

Example:

```
extern sprite_t sprites[];
put_vram(0x1b00, sprites, 13); /* 1b00 is default vram address for sprite_attribute_table */
```

From modified version of Marcel's Coleco library, to use Coleco bios sound routines.

#### **sound\_t**

Definition: Songs table data format, consist of a pointer to sound data and an address to a sound area.  
Predefined sound areas addresses are : SOUNDAREA1, SOUNDAREA2... SOUNDAREA7.

Example:

```
static byte sound1[] = {0x43,0x00,0x72,5,0x11,0x90,0xF3,0x11,0x50};
sound_t snd_table[] = {sound1,SOUNDAREA1};
play_sound(1); /* Play the first sound data specified in the sound table */
```

From Getput v1.1 library.

#### **score\_t**

Definition: Score format is composed into two unsigned values. Needs special functions.  
Values: [0,655359999]

Exemple:

```
score_t score;
score_reset(&score);
score_add(&score,1000);
print_at(10,12,score_str(&score,9));
```

## ***STRUCTURES (CUSTOM DATA TYPE)***

### **typedef struct**

definition: conglomerate data structure

Example:

```
/* coordinates (x,y) */
typedef struct {
    byte x;
    byte y;
} coorxy;

coorxy position[10];

position[0].x = 2;
```

Note: A structure like this one is used in the Coleco library for the sprites table in ram.

Important: When using a pointer to a structure element, you access data by using the arrow “->” syntax, not the dot.

Exemple :

```
/* coordinates (x,y) */
typedef struct {
    byte x;
    byte y;
} coorxy;

coorxy position[10];

coorxy *ptr_position = &position[index];

ptr_position->x = 2;
```



## OPERATORS

### USUAL SET OF BINARY ARITHMETIC OPERATORS:

- multiplication (\*)
- division (/)
- modulus (%)
- addition (+)
- subtraction (-)

Note: Unary minus performs an arithmetic negation. Unary plus is supported.

### INCREMENT (++) AND DECREMENT (--)

The most well know unary operators are increment (++) and decrement (--). These allow you to use a single operator that "adds 1 to" or "subtracts 1 from" any value. The increment and decrement can be done in the middle of an expression, and you can even decide whether you want it done before or after the expression is evaluated.

Example:

```
int sum = a + b++; /* sum = a + b then increment 'b' */  
int sum = --a + b; /* decrement 'a' then sum = a + b */
```

### BITWISE OPERATORS

- shift left (<<)
- shift right (>>)
- AND (&)
- OR (|)
- XOR (^)
- NOT (~)

### COMBINED OPERATORS

The expression form:

<variable> = <variable> <operator> <expression>;

Can be replaced with:

<variable> <operator>= <expression>;

Example:

a +=b; /\* is the same thing as the expression "a = a + b;"

### RELATIONAL OPERATORS

Relational operators allow you to compare two values, yielding a result based on whether the comparison is true or false. If the comparison is false, then the resulting value is 0.

- greater than (>)
- greater than or equal (>=)
- less than (<)
- less than or equal (<=)
- equal to (==)
- not equal to (!=)

### LOGICAL OPERATORS

There are three logical operators:

- AND (&&)
- OR (||)
- NOT (!)

Example:

```
If (!(a==b && b==c)) /* If not (a equal to b and b equal to c) then... */  
If (a<b || a<c) /* If a less than b or a less than c then... */
```

Note: Do not be confused with the bitwise operators (&,!,<) previously mentioned. If you use "MASK" in a complex if condition you must isolate the bitwise operation with ( and ) like this: if ((a&b)==c) /\* If a with an "AND MASK" b is equal to c then... \*/

## ***IF STATEMENT***

### **SIMPLE IF**

if (condition) statement1;

Example:

```
if (x==0) {x++;}
```

### **IF ... ELSE**

if (condition) statement1; else statement2;

Example:

```
if (x==0) {x++;} else {x--;}
```

## ***LOOPS***

### **FOR LOOP**

Example:

```
for (x=0;x<10;x++)
```

### **WHILE LOOP**

Example:

```
x=0; while (x<10) {x++;}
```

### **DO WHILE LOOP**

Example:

```
x=0; do {x++;} while (x<10);
```

## SCREEN MODES

Before seeing any "HELLO WORLD" on screen, you have to setup the screen mode and update the video ram memory with an appropriate characters set.

To find the information about the Video Display Processor (VDP), look at this txt file.

URL:

<http://www.msxnet.org/tech/tms9918a.txt>

[http://home.swipnet.se/~w-16418/tech\\_vdp.htm](http://home.swipnet.se/~w-16418/tech_vdp.htm)

For a text adventure, the screen mode 1 with 40 columns should be a good choice. But for the other Coleco projects, the screen modes 0 and 2 are the most appropriate choices. The screen mode 3 (very big pixels on screen) is not a good choice. Do not use the undocumented screen mode. For all my ColecoVision projects, I use the screen mode 2. This screen mode allow me to do bitmap title screen and colorfull characters set. Compute the VDP control registers values based on your own requierments.

In the Coleco library, "vdp\_out" is a routine to update the VDP control registers' values:

```
vdp_out(register,value);
```

The most important control registers are 0 and 1.

Reg. 0	-	-	-	-	-	-	M2	ExtVID
Reg. 1	4K/16K	BLANK	IE	M1	M3	-	SIZE	MAG

M1, M2 and M3 are the bits to set the screen mode.

EXVID is External VDP input (always disable this one with bit 0)... see TXT file

To set the screen mode 2 (like in my all my Coleco projects), I compute M1=0 (disable), M2=1 (enable), M3=0 (disable), ExtVID=0 (disable), 4/16K=1 (16K), BLANK=1 (enable display), IE=1 (enable NMI interruptions), SIZE=1 (16x16 sprites), MAG=0 (normal sprites, not doubled in size).

Reg. 0	0	0	0	0	0	0	1	0	02
Reg. 1	1	1	1	0	0	0	1	0	E2

SCREEN MODE 2 (normal with 16x16 sized sprites) :

```
vdp_out(0,2);
```

```
vdp_out(1,0xe2);
```

We need to setup the other VDP registers too to complete the screen mode setup.

For the special screen mode 2, we have to imagine the screen divided in three parts: TOP, MIDDLE, BOTTOM. Each part (of 8 lines) use the same character set or different character sets. Normally, we use three different character sets to do a bitmap title screen. Otherwise, only one character set is needed. You must read carefully the text files about VDP before trying to compute yourself the control registers values.

### **CHARACTERS**

The characters are used for most of the graphics on screen. They can be copied many times on screen without any problem. All the letters, numbers and symbols are characters. A character is 8x8 pixels sized except for screen mode 1 where a character is only 6x8. Normally, there is 32x24 spaces on screen where characters can be placed except for screen mode 1 (40 columns).

### **VIDEO MEMORY FOR CHARACTERS**

The names of the three tables in Video RAM for characters are:

NAME: The screen (24x32)

PATTERN: The characters pattern (256 characters: HEX values 00-FF)

COLOR: The characters color(s)

A character has the same pattern and color anywhere on screen (one exception in screen mode 2). So you can't use the same character to print a blue 'A' and a red 'A' side-by-side. The solution is using two characters with the same pattern but with different colors.

In the ASCII code, the character 'I' is the character 49 (31 in HEX value). You must understand the difference between the ASCII code and the symbol. The ASCII code for the character 'A' is 65 (41 in hex value).

Now, if the color of the character 'A' is blue and if you change the pattern of the character 'I' to looks like an 'A' but with a red color, then you just have to print the characters 'A' and 'I' side-by-side on screen to show two 'A' side-by-side... one blue and one red.

In the NAME table, there is HEX values 41 for the 'A' and 31 for the 'I'.

In the PATTERN table, there are identical HEX values for the character 'A' and 'I'.

In the COLOR table, there are different HEX values to have blue color(s) for the character 65 ('A') and red color(s) for the character 49 ('I').

DACMAN is a good example of a video game based on characters graphics.

In screen mode 0, there are only two colors (one color for bits 1 and one color for bits 0) for each bloc of 8 characters in the character set.

In screen mode 1, there are only two colors (one color for bits 1 and one color for bits 0) for all the characters.

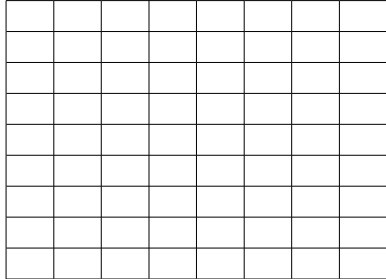
In screen mode 2, there are two colors (one color for bits 1 and one color for bits 0) by line of 8 pixels for all the characters in the character set.

For the screen mode 0 and 2, if you want to see the background color set in the VDP register, you have to use the INVISIBLE color "0".

## PROGRAMMING COLECOVISION GAMES

### CHARACTER PATTERN

A character pattern is an 8x8 graphic. Some guys name this kind of graphic: tile.



### EXAMPLE - SPACESHIP

Spaceship pattern

0	0	0	1	1	0	0	0	18
0	0	0	1	1	0	0	0	99
1	0	0	1	1	0	0	1	99
1	0	1	1	1	1	0	1	BD
1	1	1	0	0	1	1	1	E7
1	1	1	0	0	1	1	1	E7
1	0	1	1	1	1	0	1	BD
0	0	1	1	1	1	0	0	3C

In screen mode 0, a character can use only two colors (one for bits 1 and one for bits 0).

Spaceship colors

Bits 1	Bits 0	Code
		E1

Spaceship pattern with colors

0	0	0	1	1	0	0	0
1	0	0	1	1	0	0	1
1	0	0	1	1	0	0	1
1	0	1	1	1	1	0	1
1	1	1	0	0	1	1	1
1	1	1	0	0	1	1	1
1	0	1	1	1	1	0	1
0	0	1	1	1	1	0	0

## PROGRAMMING COLECOVISION GAMES

In screen mode 2, a character can use more than two colors but limited to two colors per line.

### Spaceship colors

Bits 1	Bits 0	Code
		81
		A1
		E1
		E1
		EF
		E7
		E1
		81

### Spaceship pattern with colors

0	0	0	1	1	0	0	0
1	0	0	1	1	0	0	1
1	0	0	1	1	0	0	1
1	0	1	1	1	1	0	1
1	1	1	0	0	1	1	1
1	1	1	0	0	1	1	1
1	0	1	1	1	1	0	1
0	0	1	1	1	1	0	0

### UPLOAD CHARACTER SET

To upload a character set in video memory, you can use the “upload\_ascii” routine from Coleco library.

upload\_ascii (number of the first character to upload, number of characters to upload, offset, flags for the character format);

Example: upload\_ascii (29,128-29, chrpos+29\*8, BOLD); /\* chrpos is the address location in video memory for the character patterns \*/

But, you can use your own character set. You can use one of my tools named Win-ICVGM (I.C.V.G.M for Windows) to create your own character set. This tool is based on the screen mode 0 but can be used for the screen mode 2 too (except for the color patterns).

URL:

<http://www.geocities.com/newcoleco/tools.html>

When your character set is done, save as a C file and upload your character set into video memory by using one of these routines from Coleco library.

put\_vram (video memory location for character patterns, pointer to character set table without RLE compression, size of the character set table);

rle2vram (pointer to character set table with RLE compression, video memory location for character patterns);



## SPRITES

The sprites are easy to use because you can place them anywhere on screen. Each sprite can be identified like a layer on screen. Normally, the size of a sprite is 16x16 but there is also the 8x8 format. The limits for using sprites are : never more than 4 sprites in a row, on the same scan line and never more than 32 sprites on screen at the same time. All sprites can be magnified by 2 (by changing size of the pixels in sprites).

To display a sprite on screen, you need a vector of 4 bytes (Position Y, Position X, Pattern and Colour) in the right video memory location.

## SPRITES COLOR

The bits 0 are already replaced by the invisible color so there is only one color per sprite.

To use more than one color, you have two solutions:

- Use more than one sprite (one for each color)
- Use a combination of sprites and characters like the solution used for the ghosts in Atarisoft PacMan game: the eyes are one or two characters in grey color and the body is a sprite.

## SPRITES LOCATIONS ON SCREEN


The Y location of a sprite can be any values between 0 and 255 except 208. The special value 208 tells the video chip to stop checking for sprites to display on screen. If you want to not show sprite#1 but you want to show sprite#2, use a value like 207 for the Y location of sprite#1.

## SPRITES PATTERN

### 8x8 SPRITE

A 8x8 sprite looks like a character in screen mode 0 but all bits 0 are colored with the invisible color 0. If we re-use the spaceship example, a 8x8 spaceship sprtie could be something like this.

Spaceship color

Color	Code
	4

Spaceship pattern with color

0	0	0	1	1	0	0	0
1	0	0	1	1	0	0	1
1	0	0	1	1	0	0	1
1	0	1	1	1	1	0	1
1	1	1	0	0	1	1	1
1	1	1	0	0	1	1	1
1	0	1	1	1	1	0	1
0	0	1	1	1	1	0	0

# PROGRAMMING COLECOVISION GAMES

## 16x16 SPRITE

A 16x16 sprite is a combination of four (4) 8x8 patterns. These patterns are displayed like this:

1	3
2	4

Win-ICVGM can be used to create sprites pattern 16x16.

Sprite pattern with color code A (10)

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	00	00
0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	03	E0
0	0	0	0	1	1	1	1	1	1	1	0	0	0	0	0	0F	F8
0	0	0	1	1	0	0	0	1	1	1	0	0	1	1	0	19	CC
0	0	1	1	0	1	1	0	1	0	1	1	0	1	1	0	36	B6
0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	3F	FE
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	7F	FF
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	7F	FF
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	7F	FF
0	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	70	07
0	0	1	1	0	0	0	0	0	0	0	0	1	1	0	0	30	06
0	0	1	1	1	0	0	0	0	0	0	0	1	1	1	0	38	0E
0	0	0	1	1	1	1	0	0	0	1	1	1	1	0	0	1E	3C
0	0	0	0	1	1	1	1	1	1	1	1	1	0	0	0	0F	F8
0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	03	E0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	00	00

The sprite pattern is coded like this in video memory:

00, 03, 0F, 19, 36, 3F, 7F, 7F, 7F, 70, 30, 38, 1E, 0F, 03, 00, 00, E0, F8, CC, B6, FE, FF, FF, FF, 07, 06, 0E, 3C, F8, E0, 00

## SPRITES ROUTINES

Note: These routines are available in the Coleco library.

*/\* Update sprites attributes in VRAM \*/*

update\_sprites (number of sprites, sprites table);

*/\* Make sprites disappear by changing their Y location (in video memory only) to 207 \*/*

clear\_sprites (first sprite to clear, number of sprites to clear);

*/\* Make sure no sprite is showed on screen, use the following code \*/*

clear\_sprites (0, 64);

*/\* Check collision is complex to understand. Read Coleco library section \*/*

check\_collision (...);

## JOYSTICK (Hand Controller)

The Coleco joystick is divided in two parts: keypad and joypad.

### JOYPAD

Use joypad\_1 (joystick in port#1) and joypad\_2 (joystick in port#2) to know the direction(s) and the "pressed" fire button(s).

How to use the joypad\_1 and joypad\_2 values?

The value is code as a byte where each bit represents a fire button or a direction. You have to use an AND mask to extract what you need from the joypad value.

Fire 1	Fire 2	Fire 3*	Fire 4*	Left	Down	Right	Up
--------	--------	---------	---------	------	------	-------	----

\*: Only available for the Super Action controller.

Example: I want to do nothing until a fire button is pressed on port#1.

*/\* Bits for fire buttons are the four (4) highest bits in the joypad value. So, you need to do an AND mask with four (4) bits 1 and four (4) bits 0 like this: 11110000 = F0 \*/*

*while ( (joypad\_1 & 0xf0) == 0 ); /\* loop if no fire button is pressed (port#1) \*/*

With Getput1 library, you can also use the predefined constants : FIRE1, FIRE2, FIRE3, FIRE4, LEFT, DOWN, RIGHT and UP. It makes coding more easier to read and write.

*If (joypad\_1 & FIRE1) /\* if Fire 1 is pressed \*/*

### KEYPAD

Use keypad\_1 and keypad\_2 to know which number was pressed. The keypad\_1 is for the keypad on port#1. The keypad\_2 is for the keypad on port#2. The first time you call keypad\_1 and keypad\_2, the result 0 means nothing. After this first time, the keypad value will match with the right key number pressed.

0-9 = 0-9

10 = \* (standard to pause the game and replay game with the same game option)

11 = # (standard to return to menu after the game)

12-14 = unused

15 = no key pressed

### **OTHER CONTROLLERS**

#### **Super Action Controller**

This special hand controller has four fire buttons and a speed roller. The speed roller is not quite effective, so you need to make it moving really fast. This controller is used in several games: Front Line, Rocky Super Action Boxing, Super Action Baseball, Super Action Football and Super Action Soccer.

The speed roller is a bit slow to be used as a paddle. It's why the speed roller is most a speed indicator like in the Super Action Baseball game to make running slow or fast baseball players. The signed value of the spinner shows you the direction of the movement and also the speed movement. Note that the spinner in port#2 works the same way as the spinner in port#1.

#### **Expansion Module 2: Turbo Drive**

This Turbo Drive controller allows you to actually drive vehicles in games: Bump'n Jump, Destructor, Dukes of Hazzard, and Turbo. The steering wheel is plugged into port#1. It needs four (4) C batteries and a joystick plugged into port#2. The accelerator pedal is only a trigger.

The spinner in port#1 is working backward for the Turbo Drive module. You have to subtract the port#1 spinner value to the 'x' position to obtain the new position.

#### **Roller Controller**

This free-rolling control ball (track ball) gives you a 360 degrees field of lightning fast movement. This controller is plugged into joystick ports to allow movement in all directions and power supply port to avoid using batteries like the Turbo Drive module. There are four fire buttons on the Roller Controller, these buttons are (left) fire 1 and 2 for port#1 and (right) fire 2 and 1 for port#2. You need at least one joystick plugged into one of the joystick ports on the roller controller to select game options before playing. This controller is used in two games: Slither and Victory.

The spinner in port#1 is working backward for the roller controller, but the spinner in port#2 is working forward. You have to subtract the port#1 spinner value to the 'x' position and add the port#2 spinner value to the 'y' position to obtain the new position.

### **SPINNERS VARIABLES**

In the Coleco library by Marcel de Kogel, the spinners values are updated in global variables named `spinner_1` (port#1) and `spinner_2` (port#2) each time a NMI interrupt occurs. It's a good idea to update the player position into the nmi routine, otherwise, the player movements may be not fluid.

Note: Declared as "extern byte" in the "coleco.h" file, the spinners variables type is much a char (signed value [-128,127]) than a byte (unsigned value [0,255]).

## PROGRAMMING COLECOVISION GAMES

### **SOUND**

I use the sound routines included in the Coleco library by Marcel de Kogel so I can talk a little about making some sounds in Coleco projects.

The way I learn how to generate sounds in my Coleco project is by looking the source code of Cosmo Challenge by Marcel de Kogel and the technical information about tone generator.

To avoid computing myself a sound effect, I created a tool named WAV2CV to convert a sound from a normal WAV file into a C file to be used with the Coleco library sound routines. I have done two programs about sound effects with menus. The source code is available: visit my Coleco web site.

URL:

<http://www.geocities.com/newcolecto>

Ok! My Coleco web site is not up-to-date but this is the web page where you can found some of my codes. Only in French, sorry!

URL:

<http://www.geocities.com/newcolecto/dev/devfr.html>

### **THE SOUND ROUTINES**

At the end of the NMI routine, I add the following instruction:

#### **update\_sound ();**

This instruction placed in the nmi will update the sound at each vertical retrace. Yes, in the NMI routine, it's the best place to put this instruction.

#### **start\_sound (sound\_data,sound\_priority):**

This instruction adds information in the RAM like the pointer to the sound in ROM and the sound priority. The information in RAM will be used by the update\_sound routine to play the sound. When 3 sounds are played and another sound is started, the sound\_priority is used to see which sound must be ignored because there is only 3 sound channel. A sound priority number 10 win over a sound priority number 1. (I hope you understand)

#### **sound\_pointer = start\_sound(sound\_data, sound\_priority):**

You can get the information about the pointer in RAM where the sound information is placed. This way, you can use the following instruction to stop this particular sound without stops all sounds:

#### **stop\_sound(sound\_pointer):**

You cannot use stop\_sound without the sound\_pointer otherwise all sounds may stop forever.

## PROGRAMMING COLECOVISION GAMES

### **sound\_on();**

This instruction enables sound output.

### **sound\_off();**

This instruction disables sound output.

### **play\_dsound(sound\_pointer, step);**

This routine, originally from DSound library but now in Getput library, plays digital sounds. You can set the sound pitch speed with 'step' parameter. You must disable NMI before using this routine.

### THE WAY I ADD SOUNDS IN MY COLECO PROJECTS

I don't fully understand how works the sound routine "update\_sound" but I can talk about my tool WAV2CV.

It's very easy. Simply open wav2cv and select the WAV file you want to convert. You can also change the parameters before selecting the WAV file to increase or decrease the sound quality. If it's a simple sound like BEEP, use only one channel. If it's a complex sound like a digitalized sound, use two or three channels. Note: WAV2CV can freeze if you ask for more channels than you really need (it's a bug i can't fix). Normally, WAV2CV will be minimized for short laps of time and take 90% of your CPU time. More longer is the sound, more longer you will freeze your Windows.

WAV2CV applies FAST FOURRIER TRANSFORM to find frequencies of the sound... for each laps of time based on the vertical retrace frequency (NTSC 60Hz or PAL 50Hz).

The generated C file can be renamed and added into your Coleco project. You can also copy-paste the C code into your source code where you want to avoid having too much C files to compile. I suggest using one big C file to regroup all the sound data for your Coleco project.

WAV2CV cannot convert noisy sounds. To add a noise sound effect, you must refer to the sound data you can found in "Cosmo Challenge" source code.

```
byte shoot_sound[]={
    1,
    0xf8,0xe4, 1,0xf2, 1,0xe4, 1, 0x01,0xe5,
    1,0xe4,
    1,0xe5, 1,0xe4, 1,0xe5, 1,0xe4,
    1,0xe5, 1,0xe4, 1,0xe5, 1,0xe4,
    5,
    0,0,0
};
```

Note: Normally, a sound data table ends with three zero like this: "0,0,0".

For more details, look at the "MARCEL's SOUND DATA FORMAT" (page 90).

## TOOLS

The following pages show you some tools for Windows made by Marcel de Kogel and Daniel Bienvenu to help you in your ColecoVision projects. You may have to create your own tools if you are not satisfied.

Let's start with the only one who can convert sounds into ColecoVision sound format.

### WAV2CV

*by Daniel Bienvenu*

WAV2CV is programmed to convert uncompressed mono WAV files into a ColecoVision format. But, the current version of WAV2CV is based on the ColecoVision library, not on the ColecoVision BIOS. It uses the Fast Fourier Transform to extract frequencies from the WAVE for each time pitch. The first conversion result is never perfect but you can use different parameters to see if the result can be better.

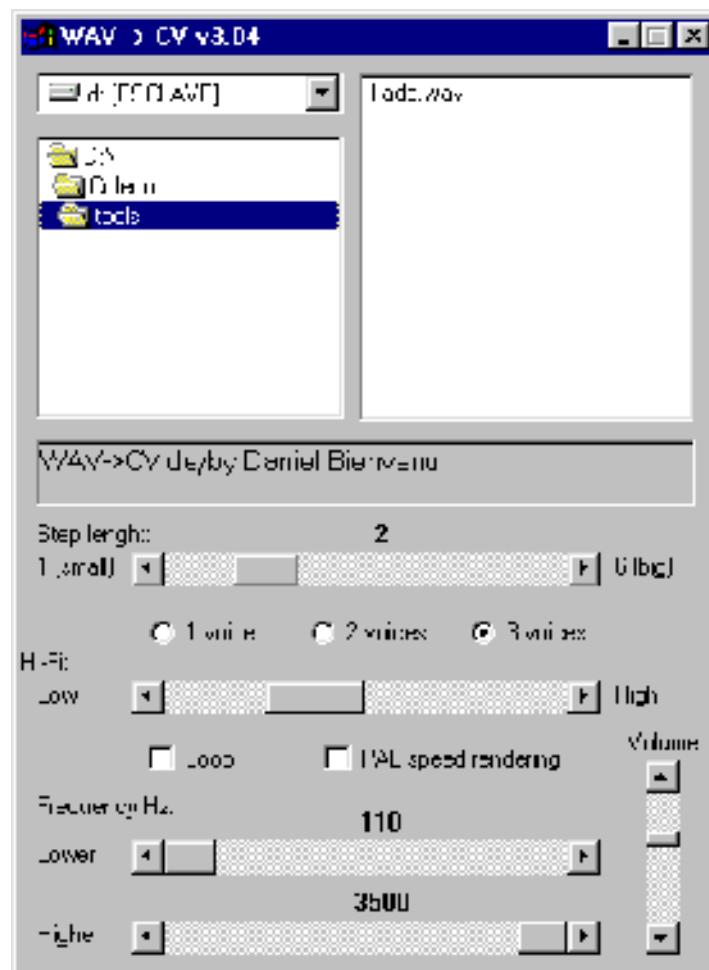


Figure 1 WAV2CV user interface



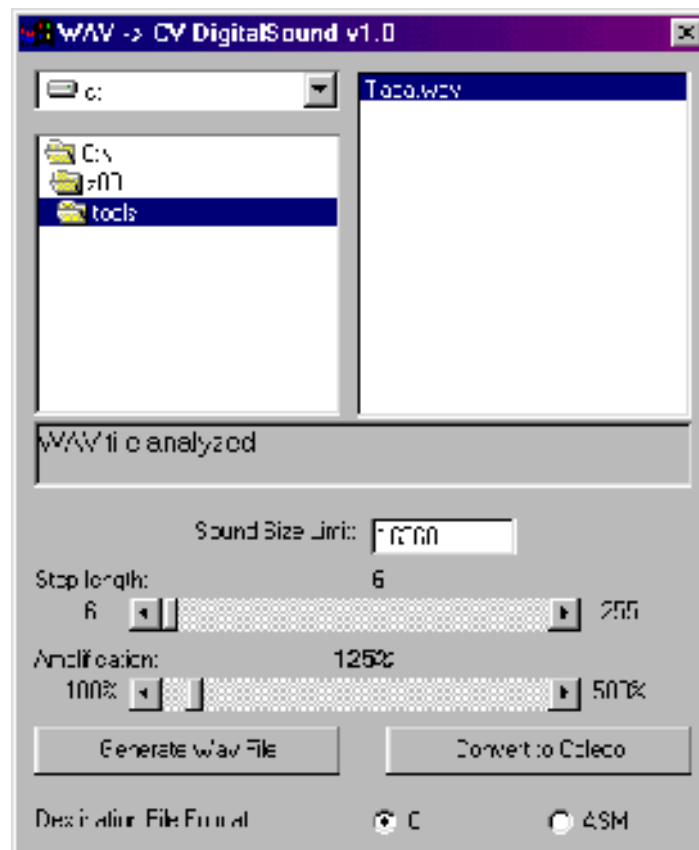
### WAV2CVDS

*by Daniel Bienvenu*

WAV2CVDS is the only tool who can convert mono WAV files into a digital sound format for the ColecoVision. The digital sound format used is 4 bits per sample with special code \$00 to indicate an RLE compression or the "end of sound" with another \$00.

How to use it

1. Select a WAV file in the filelist box. The software read the WAV file primary information like the sample rate.
2. Write the sound size limit (in ROM) you want. The software compute the minimum step length you can use.
3. Set the step length of the digital sound. A big step gives you a sound with a poor quality but with less memory space than a small step.
4. Set the amplification.
5. Click on the "Generate Wav File" button to listen the digital sound.
6. Select your destination file format and click on the "Convert to Coleco" button.



**Figure 2 WAV2CVDS user interface**

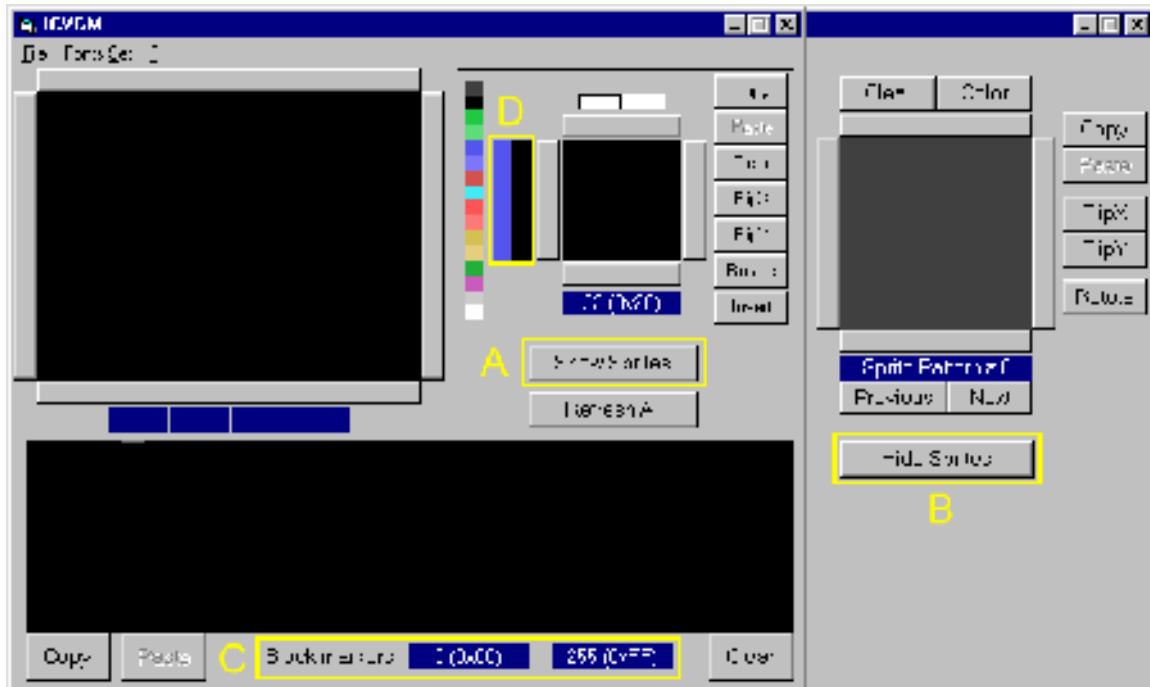
Use play\_dsound routine from DSound library to play the digital sounds generated with this tool.



## I.C.V.G.M. v3

by Daniel Bienvenu

This special version of my best graphic editor software is only for screen mode 2 (aka Graphic Mode II), and limited to one character set. It is not an update version of ICVGM v2. ICVGM v3 came after a suggestion received by email to simply create and edit multicolor characters.



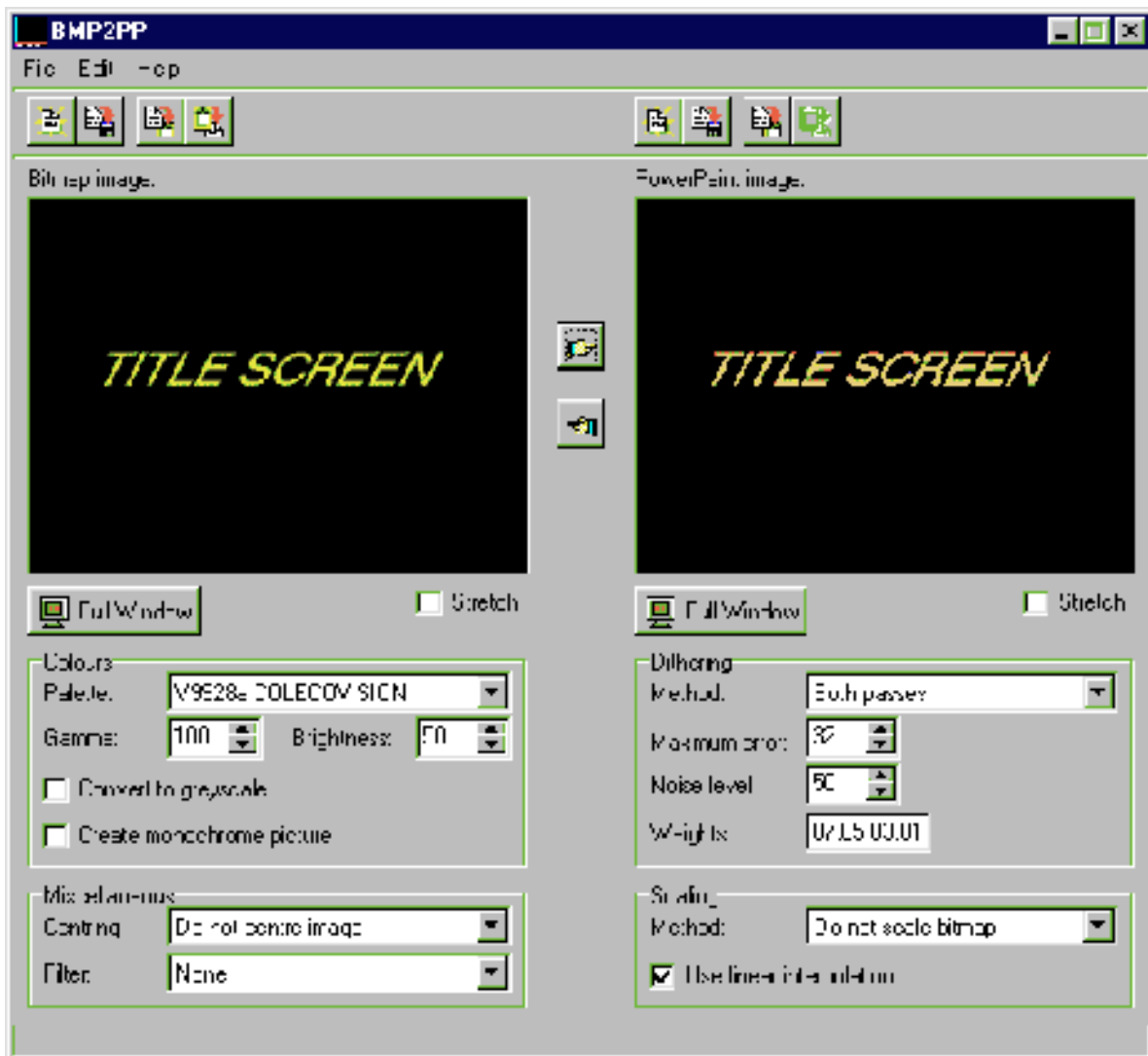
**Figure 4 I.C.V.G.M. version 3 user interface**

- A. This button allow the user to show the sprites editor.
- B. This button allow the user to hide the sprites editor.
- C. This text show the block markers information. By clicking on the character set with the right mouse button, you set a block of markers to be able to copy, paste or clear character patterns.
- D. Because this software version is based on screen mode 2, you can set colors for each line of a character pattern.

## BMP2PP

*by Marcel de Kogel*

BMP2PP convert BMP pictures and also pictures from the clipboard into a valid ColecoVision format named PowerPaint. You can also convert a PowerPaint picture into a BMP file. Adjust the parameters to convert as well as possible your bitmap picture. Go in the File menu to save the result into a PowerPaint file ".pp". If you need to edit some pixels on the converted picture, I suggest you to use CVPAIN.T. Otherwise you can only use PP2C or PP2ASM to convert and compress PowerPaint files into data for your ColecoVision projects.



**Figure 5 BMP2PP user interface**

## PP2C and PP2ASM

by Daniel Bienvenu

PP2C and PP2ASM are useful to quickly convert and compress (RLE encoding based on Marcel's ColecoVision library) PowerPaint pictures. Use PP2C to convert your picture into C format, and PP2ASM to convert into [T]ASM format. PP2C and PP2ASM are also integrated in CVPAINTE tool. Using RLE compressed data allow to add more than one picture in your ColecoVision projects without taking too much memory space (ROM space).

First of all, (figure 6) you need to double-click on the PowerPaint file ".pp" you want. After seeing the picture in the preview zone, click OK to continue. After, (figure 7) write or select the file name to export data. And finally, (figure 8) write the name of this generated data table.

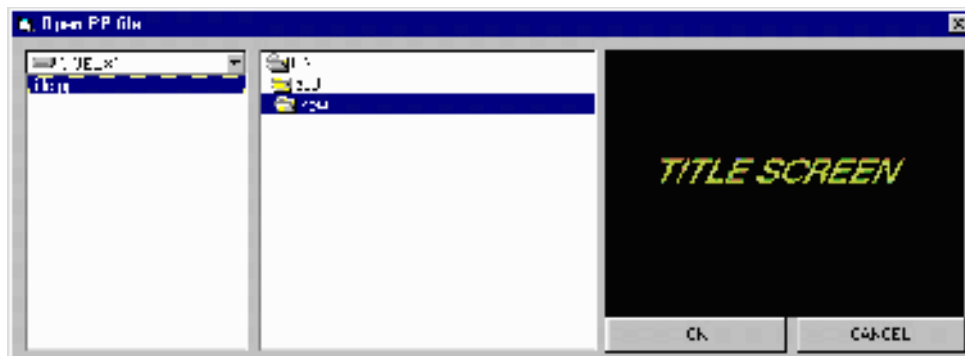


Figure 6 PP2C - Open PP file window

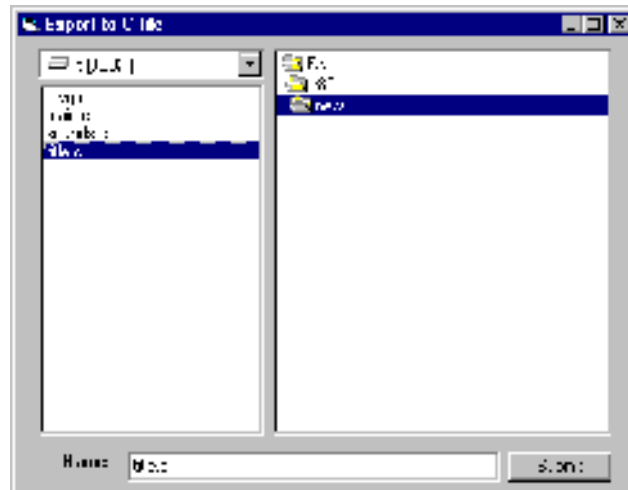


Figure 7 PP2C - Export to C file window



Figure 8 PP2C - Picture table name window

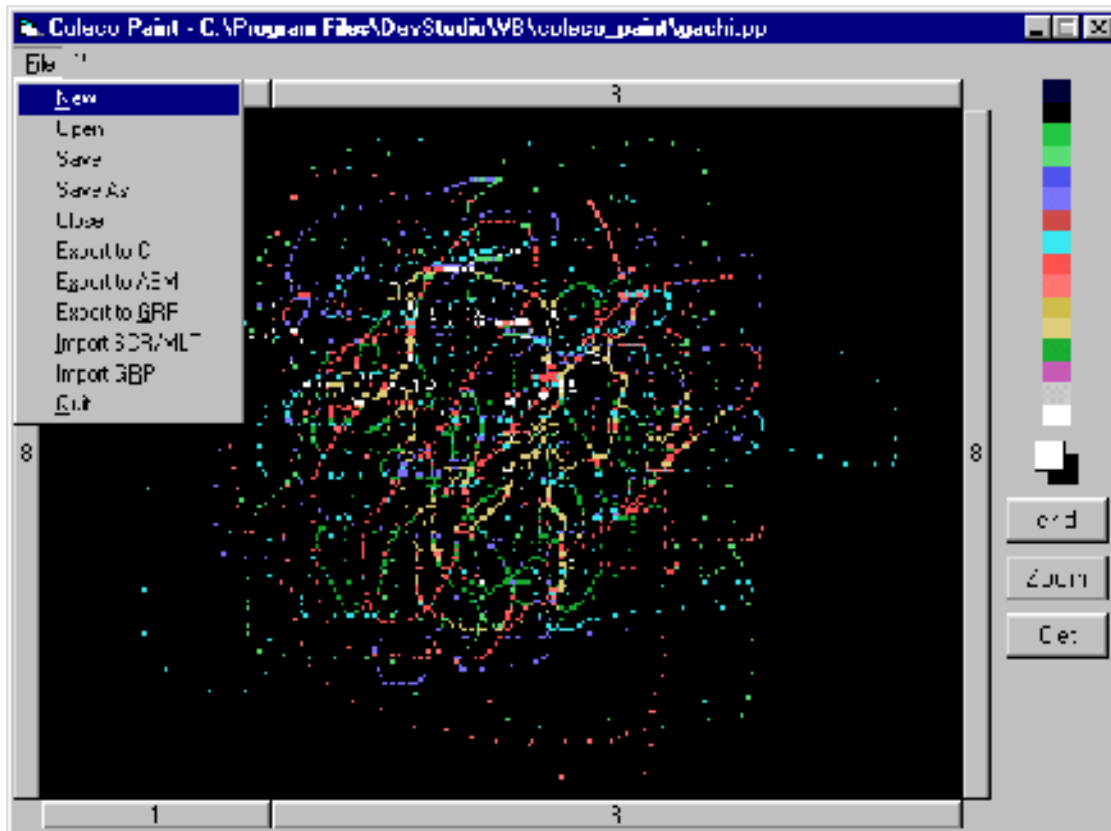
### CVPAINT

*by Daniel Bienvenu*

CVPAINT still a beta version. We will not talk too much about this software. The reason I programmed this tool is to avoid using many times BMP2PP to convert pictures just because I add some pixels on it. CVPAINT is (for now) the only graphic editor tool for Windows who respects the limit of the ColecoVision graphic chip. You can draw pixels, zoom and use a grid but you can't do a COPY-PASTE (not well programmed).

You can also import pictures from other formats like the one used for the ZX Spectrum (SCR and MLT). I added also the "Export to C" and "Export to ASM" routines from PP2C and PP2ASM to avoid using too many tools.

The "Get" button is a primitive "copy" function but it doesn't work well if you don't know how to use it. This is how it works. When you click on the "Get" button, the name change for "Select" and you have to select an area on screen to copy. When it's done, press the "Select" button. It creates a new layer at the top left of the screen. Use your mouse to drag this layer on screen where you want to copy it. Use the right click button on your mouse to merge the layer on screen. Don't use "Zoom" or "grid" button otherwise the copy can't be done. There is no "Back" button so be careful.



**Figure 9 CVPAINT user interface with "File" menu opened**

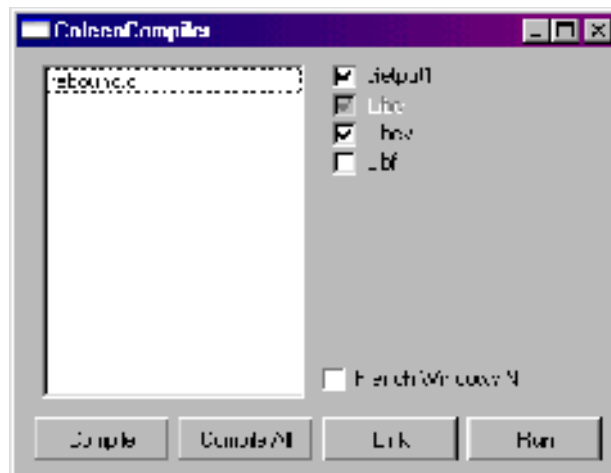
### CCI - Coleco Compiler Interface

This tool uses the Hi-Tech C compiler (for CP/M) and 22NICE CP/M emulator (for DOS). This tool helps you to compile and link your project without editing batch files and writing command lines.

#### How to use CCI?

Before using this software you must copy it in your Coleco project directory.

1. To use the "Compile" button, you must select a file in the file list box first.
2. Use the "Compile All" button to compile your entire project in one step.
3. Select the libraries you need by checked the appropriated checkboxes.
4. If you use a french version of Windows NT, 2000 or XP, then you may need to check the "French Windows NT" checkbox.
5. After selecting the right libraries, use the "Link" button to link them with your project.
6. If every things ok, you will see a valid "result.rom" file in your project directory with a "map.txt" file.
7. The "Run" button will start the "result.rom" file with VirtualColeco emulator. If you want to use another emulator, add a batch file named "run.bat" in your project directory. This batch file must have the right instructions to play "result.rom" file with another emulator.



**Figure 10 CCI user interface**

See a compiling example in the programs section in this document.

## PROGRAMMING COLECOVISION GAMES

### ***LIBRARY: COLECO***

The ColecoVision library made by Marcel de Kogel is programmed in ASM to be used with Hi-Tech C compiler. This library is divided in two parts: "crtcv.obj" and "cvlib.lib". "crtcv.obj" is the header of the ROM started at address 8000. "cvlib.lib" is the ColecoVision library itself with many useful routines.

### **ROUTINES IN COLECO LIBRARY**

The following routines are the most important ones available in the ColecoVision library.

#### **rle2ram**

*/\* RLE decode specified data to specified RAM area. Returns pointer to first unused free \*/*  
void rle2ram(void \*rldata, void \*dest);

#### **rle2vram**

*NMI\**  
*/\* RLE decode specified data to specified VRAM area. Returns pointer to first unused free \*/*  
void rle2vram(void \*rldata, unsigned dest);

#### **put\_vram**

*NMI\**  
*/\* Upload RAM to VRAM. count should be a multiple of 256 \*/*  
void put\_vram (unsigned offset,void \*ptr,unsigned count);

#### **get\_vram**

*NMI\**  
*/\* Get array of VRAM bytes. count should be a multiple of 256 \*/*  
void get\_vram (unsigned offset,void \*ptr,unsigned count);

#### **fill\_vram**

*NMI\**  
*/\* Fill VRAM area with specified value \*/*  
void fill\_vram (unsigned offset,byte value,unsigned count);

#### **put\_vram\_ex**

*NMI\**  
*/\* Upload RAM to VRAM, applying specified AND and XOR masks \*/*  
void put\_vram\_pattern (unsigned offset,void \*ptr, unsigned count, byte and\_mask, byte xor\_mask);

#### **put\_vram\_pattern**

*NMI\**  
*/\* Upload pattern to VRAM \*/*  
void put\_vram\_pattern (unsigned offset,void \*pattern, byte psize,unsigned count);



## PROGRAMMING COLECOVISION GAMES

### **set\_default\_name\_table**

NMI\*

/\* Upload default name table \*/

void set\_default\_name\_table (unsigned offset);

Note: set\_default\_name\_table is used in bitmap mode to fill the screen with 3 set of characters from 00 to FF. This is necessary to show a bitmap picture on screen by using all the characters patterns.

### **vdp\_out**

NMI\*

/\* Write specified VDP register \*/

void vdp\_out (byte reg, byte val);

### **screen\_on**

NMI\*

/\* Turn display on \*/

void screen\_on (void);

### **screen\_off**

NMI\*

/\* Turn display off \*/

void screen\_off (void);

Note: "screen\_on" and "screen\_off" are used to display or not the screen. You can update the screen without showing the modification unless it's done.

### **disable\_nmi**

/\* Disable NMI \*/

void disable\_nmi (void);

### **enable\_nmi**

/\* Enable NMI \*/

void enable\_nmi (void);

Note: Use *disable\_nmi* to stop NMI interruptions when you do some important update then use *enable\_nmi* to restart NMI interruption. In Getput-1 library, all the "print" routines already use *disable\_nmi* and *enable\_nmi* except *put\_char*, *get\_char*, *put\_frame* and *get\_bkgrnd*.

### **update\_sound**

NMI\*

/\* Check for new sound events \*/

void update\_sound (void);

## PROGRAMMING COLECOVISION GAMES

### **start\_sound**

*/\* Setup a sound channel. Returns pointer to sound channel allocated \*/*  
void `*start_sound` (void `*data`, byte priority);

### **stop\_sound**

*/\* Stop specified sound channel \*/*  
void `stop_sound` (void `*channel`);

### **sound\_on**

*/\* Enable sound output \*/*  
void `sound_on` (void);

### **sound\_off**

*/\* Disable sound output \*/*  
void `sound_off` (void);

### **delay**

*/\* wait specified VBLANKs \*/*  
void `delay` (unsigned count);

Note: "delay" is necessary to slowdown the execution. Otherwise, the gameplay is too fast, unplayable.

### **get\_random**

*/\* Fast random routines. Return a byte value between 0 and 255. \*/*  
byte `get_random` (void);

### **upload\_ascii**

**NMI\***

*/\* Upload ASCII characters \*/*  
void `upload_ascii` (byte first, byte count, unsigned offset, byte flags);

```
#define NORMAL      0
#define ITALIC     1
#define BOLD       2
#define BOLD_ITALIC (ITALIC | BOLD)
```

Note: If you use the Getput-1 library, you must use "upload\_default\_ascii" routine.

### **utoa**

*/\* Convert unsigned integer to ASCII. Leading zeros are put in buffer \*/*  
void `utoa` (unsigned value, void `*buffer`, byte null\_character);

Note: If you use Getput-1 library and you don't know well how to use *utoa*, you must use *str*.

## PROGRAMMING COLECOVISION GAMES

### **sprites struct and table**

```
/* sprite_t: position Y,X, pattern number and colour code */
typedef struct
{
    byte y;
    byte x;
    byte pattern;
    byte colour;
} sprite_t;
extern sprite_t sprites[64];
```

### **update\_sprites**

NMI\*

```
/* Upload sprites to VRAM. Arguments are maximum number of sprites to upload
   (normally 32) and the sprite attribute table offset */
void update_sprites (byte numsprites,unsigned sprtab);
```

Note: If you use Getput-1 library and you don't know well how to use *update\_sprites*, you must use *updatesprites*.

### **check\_collision**

```
/* Check collision between two sprites (between areas). Sizes decode as follows:
   lobyte - first pixel set
   hibyte - number of pixels set */
byte check_collision (sprite_t *sprite1,sprite_t *sprite2,
                     unsigned sprite1_size_hor,unsigned sprite1_size_vert,
                     unsigned sprite2_size_hor,unsigned sprite2_size_vert);
```

NMI\* : Be aware to not use these functions when NMI interrupt may occurs. Use them directly in the nmi routine or use *disable\_nmi*.

### ***LIBRARY: COLECO OPTIMIZED FOR 4K***

This modified version of Marcel's ColecoVision library is optimized to produce smaller games by using ColecoVision BIOS functions and no support for : spinner, 3<sup>rd</sup> fire button, 4<sup>th</sup> fire button and less sprites.

### **NEW ROUTINES IN THIS COLECO LIBRARY**

The following routines are the new routines implanted in the ColecoVision library for this special version.

#### **play\_sound**

*/\* Play the N<sup>th</sup> sound data specified in the sound table "snd\_table". play\_sound(1) plays the 1<sup>st</sup> sound \*/*  
void play\_sound (byte number);

#### **stop\_sound**

*/\* Stop the N<sup>th</sup> sound data specified in the sound table "snd\_table" if playing \*/*  
void stop\_sound (byte number);

#### **reflect\_vertical**

**NMI\***  
*/\* Reflect N times 8 bytes of pattern around the vertical axis \*/*  
*/\* table\_code : 0 = sprite\_name, 1 = sprite\_generator, 2 = name (screen), 3 = pattern, 4 = color \*/*  
void reflect\_vertical (byte table\_code, unsigned source, unsigned destination, unsigned count);

#### **reflect\_horizontal**

**NMI\***  
*/\* Reflect N times 8 bytes of pattern around the horizontal axis \*/*  
*/\* table\_code : 0 = sprite\_name, 1 = sprite\_generator, 2 = name (screen), 3 = pattern, 4 = color \*/*  
void reflect\_horizontal (byte table\_code, unsigned source, unsigned destination, unsigned count);

#### **rotate\_90**

**NMI\***  
*/\* Rotate clockwise N times 8 bytes of pattern \*/*  
*/\* table\_code : 0 = sprite\_name, 1 = sprite\_generator, 2 = name (screen), 3 = pattern, 4 = color \*/*  
void rotate\_90 (byte table\_code, unsigned source, unsigned destination, unsigned count);

**NMI\*** : Be aware to not use these functions when NMI interrupt may occurs. Use them directly in the nmi routine or use disable\_nmi.

## PROGRAMMING COLECOVISION GAMES

### ***LIBRARY: GETPUT***

This is my own toolbox programmed in C to be used in any ColecoVision game projects. This library is based on screen mode 0 or 2 and Cosmo Challenge source code.

### **GETPUT**

The first version of "getput" made in year 2000 has only 6 routines.

#### **cls**

NMI\*

/\* cls is a CLEAR SCREEN routine \*/  
void cls(void);

Example:

```
cls();
```

#### **get\_char**

NMI\*

/\* get\_char: This routine return the character value (in video memory) at location X,Y \*/  
char get\_char (byte x,byte y);

Example:

```
char c;  
c = get_char (15,11);
```

#### **put\_char**

NMI\*

/\* put\_char: This routine put a character (in video memory) at location X,Y \*/  
void put\_char (byte x,byte y,char s);

Example:

```
char c;  
c = 'A';  
put_char(15,11,c);
```

#### **center\_string**

NMI\*

/\* center\_string: This routine PRINT a string on screen in the middle of the line L \*/  
void center\_string (byte l,char \*s);

Example:

```
center_string (11,"GAME OVER");
```

## PROGRAMMING COLECOVISION GAMES

### **print\_at**

NMI\*

/\* print\_at: This routine PRINT a string on screen at location X,Y \*/

void print\_at (byte x, byte y,char \*s);

Example:

```
print_at (0,0,"HELLO WORLD");
```

### **pause**

NMI\*\*

/\* pause: This routine waits for any pressed fire button on port#1 or port#2. \*/

void pause (void);

Example:

```
pause ();
```

NMI\* : Be aware to not use these functions when NMI interrupt may occurs. Use them directly in the nmi function or use disable\_nmi.

NMI\*\* : These functions doesn't work if NMI is disabled. Use enable\_nmi.

## PROGRAMMING COLECOVISION GAMES

### GETPUT 1

Extended version of "getput" made in years 2002-2004 has over 20 more routines. This new library is compatible with the original version "getput" but compiled in a real library format.

#### **pause\_delay**

*/\* pause\_delay: waits for a laps of time but can be interrupted by pressing a fire button \*/*  
void pause\_delay(unsigned i);

Example:

```
    pause_delay (200);
```

#### **rnd**

*/\* rnd: Return a random number between min and max [0, 65535] \*/*  
unsigned rnd(unsigned min, unsigned max);

Example:

```
    unsigned number = rnd(1,1000);
```

#### **rnd\_byte**

*/\* rnd\_byte: Return a random number between min and max [0, 255] \*/*  
byte rnd\_byte(byte min, byte max);

Example:

```
    byte number = rnd_byte(1,6);      /* dice 6 faces*/
```

#### **str**

*/\* str: Convert an unsigned value into a string \*/*  
char \*str(unsigned value);

Example:

```
    print_at (10,10,str (100));
```

#### **show\_picture**

**NMI\***

*/\* show\_picture: Show a picture (encoded RLE) on screen \*/*  
void show\_picture(void \*picture);

Example:

```
    show_picture (title);
```

## PROGRAMMING COLECOVISION GAMES

### **screen\_mode\_2\_bitmap**

NMI\*

/\* screen\_mode\_2\_bitmap: Set the screen mode 2 to show a bitmap picture \*/  
void screen\_mode\_2\_bitmap(void);

Example:

```
screen_mode_2_bitmap ();
```

### **screen\_mode\_2\_text**

NMI\*

/\* screen\_mode\_2\_text: Set the screen mode 2 to print text on screen \*/  
void screen\_mode\_2\_text(void);

Example:

```
screen_mode_2_text ();
```

### **upload\_default\_ascii**

NMI\*

/\* upload\_default\_ascii: Set the default character set \*/  
void upload\_default\_ascii(byte flags);

Example:

```
upload_default_ascii (BOLD);
```

### **paper**

NMI\*

/\* paper: Set the background color by using the VDP register 7 \*/  
void paper(byte color);

Example:

```
Paper (4); /* A dark blue background color */
```

### **load\_color**

NMI\*

/\* load\_color: Set the color of the character set (ICVGM) \*/  
void load\_color(byte \*color);

Example:

```
load_color (color);
```

### **load\_namerle**

NMI\*

/\* load\_namerle: Show a screen (rle encoded) on screen (ICVGM) \*/  
void load\_namerle(byte \*namerle);

Example:

```
load_namerle (namerle);
```



## PROGRAMMING COLECOVISION GAMES

### **load\_patternrle**

NMI\*

/\* load\_patternrle: Set the characters pattern (ICVGM) \*/  
void load\_patternrle(byte \*patternrle);

Example:

```
load_patternrle (patternrle);
```

### **load\_spatternrle**

NMI\*

/\* load\_spatternrle: Set the sprites pattern (ICVGM) \*/  
void load\_spatternrle(byte \*spatternrle);

Example:

```
load_spatternrle (spatternrle);
```

### **change\_pattern**

NMI\*

/\* change\_pattern: update N characters starting with the character c \*/  
void change\_pattern(byte c, byte \*pattern, byte N);

Example:

```
byte pattern[] = {1,2,4,8,16,32,64,128};  
change_pattern ('A',pattern,1);
```

### **change\_spattern**

NMI\*

/\* change\_spattern: update N sprites patterns, starting with sprite pattern number s \*/  
/\* N = number of 8x8 sprites patterns to update OR 4x number of 16x16 sprites pattern \*/  
/\* If sprites are sized 16x16, s must be 4x the number of the first sprite pattern to update \*/  
void change\_spattern(byte s, byte \*pattern, byte N);

Example:

```
byte spattern[] = {1,2,4,8,16,32,64,128, 128,64,32,16,8,4,2,1, 128,64,32,16,8,4,2,1, 1,2,4,8,16,32,64,128};  
/* Can be one big 16x16 sprite pattern OR 4 little 8x8 sprite patterns */  
change_spattern (0,spattern,4);  
/* Note: s = (sprite number 0) x 4 = 0, N=(1 "big 16x16 pattern") x 4 = 4 */
```

### **change\_color**

NMI\*

/\* change\_color: update N characters colour (starting with the character c) with color data \*/  
void change\_color(byte c, byte \*color, byte N);

Example:

```
byte pattern_font[] = {0xA0, 0x70, 0x80};  
/* change the color of characters A (in yellow),B (in cyan) and C (in red)*/  
change_color ('A',pattern_font,3);
```

## PROGRAMMING COLECOVISION GAMES

### **fill\_color**

NMI\*

```
/* fill_color: fill the N characters (starting with the character c) in a color */  
void fill_color(byte c, byte color, byte n);
```

Example:

```
/* The characters D,E,F and G will be in green color */  
fill_color('D', 0x20, 4);
```

### **change\_multicolor**

NMI\*

```
/* change_multicolor: update the colors of the character c */  
void change_multicolor(byte c, byte *color);
```

Example:

```
byte red_yellow_font [] = {0x60,0x80,0x90,0xA0,0x90,0x80,0x60,0x60};  
/* update the colors of the character H with the red-yellow color font */  
change_multicolor('H',red_yellow_font);
```

### **change\_multicolor\_pattern**

NMI\*

```
/* change_multicolor_pattern: update the colors of the N characters (c and the following) */  
void change_multicolor_pattern(byte c, byte *color, byte n);
```

Example:

```
byte green_yellow_font [] = {0xC0,0x20,0x30,0xA0,0x30,0x20,0xC0,0xC0};  
/* update the colors of the characters I and J with the green-yellow color font */  
change_multicolor('I',green_yellow_font,2);
```

### **choice\_keypad\_1 and choice\_keypad\_2**

```
/* Wait until a key is pressed on keypad#1 or keypad#2 between min and max */  
byte choice_keypad_1(byte min, byte max);  
byte choice_keypad_2(byte min, byte max);
```

Example:

```
byte choice;  
/* use keypad#1 to select a number between 1,2,3 and 4 */  
choice = choice_keypad_1 (1,4);  
/* use keypad#2 to select a number between 5,6,7 and 8 */  
choice = choice_keypad_2 (5,8);
```

### **updatesprites**

NMI\*

```
/* updatesprites: Update N(count) sprites data in video memory with the sprites table */  
void updatesprites(byte first, byte count)
```

Example:

```
/* Update 32 sprites [0,31] */  
updatesprites(0,32);
```

## PROGRAMMING COLECOVISION GAMES

### **sprites\_simple**

NMI\*

/\* sprites\_simple: To set sprites pixels at the normal size \*/  
void sprites\_simple(void);

### **sprites\_double**

NMI\*

/\* sprites\_double: To set sprites pixels two times bigger than normal \*/  
void sprites\_double(void);

### **sprites\_8x8**

NMI\*

/\* sprites\_8x8: To set the sprites to be 8x8 pixels \*/  
void sprites\_8x8(void);

### **sprites\_16x16**

NMI\*

/\* sprites\_16x16: To set the sprites to be 16x16 pixels \*/  
void sprites\_16x16(void);

### **Set of "AND" masks for the joystick: UP, DOWN, LEFT, RIGHT and FIRES**

/\* A set of CONSTANTS to help with the joystick return value (AND MASK) \*/  
/\* These constants are named : UP, RIGHT, DOWN, LEFT, FIRE1, FIRE2, FIRE3, FIRE4 \*/

Example:

If (joypad\_1&LEFT) /\* joystick#1 go left? \*/  
If (joypad\_2&RIGHT) /\* joystick#2 go right? \*/

### **wipe\_off\_down**

/\* This is a special effect to clean smoothly the screen in bitmap mode from top to bottom \*/  
void wipe\_off\_down(void);

### **wipe\_off\_up**

/\* This is a special effect to clean smoothly the screen in bitmap mode from bottom to top \*/  
void wipe\_off\_up(void);

NMI\* : Be aware to not use these functions when NMI interrupt may occurs. Use them directly in the nmi function or use disable\_nmi.

Note: Before GETPUT version 1.1, you needed to enable the NMI interrupts to properly use pause\_delay, wipe\_off\_down and wipe\_off\_up functions. Now, pause\_delay enables NMI by itself, and the wipe\_off\_down and wipe\_off\_up functions are working whatever if the NMI is enabled or not.