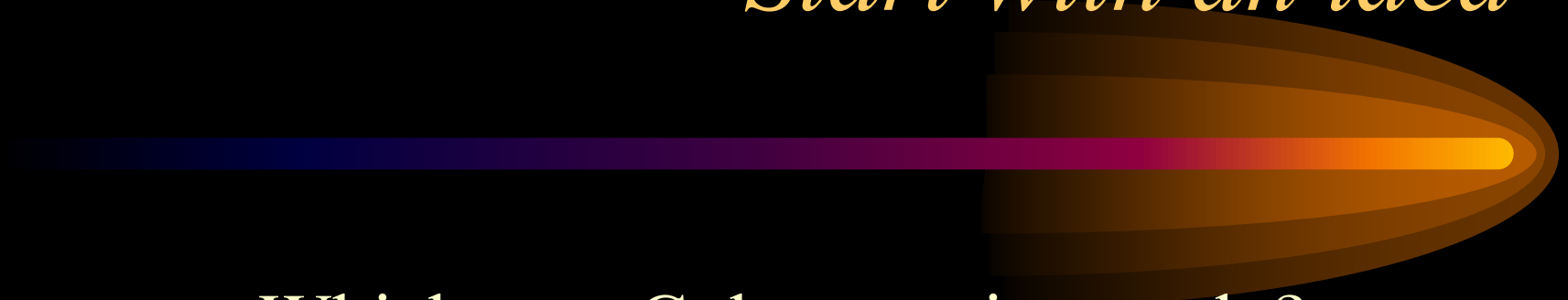
# *New ColecoVision Programming*

Programming a new ColecoVision game using Windows and DOS

# *Step by Step*

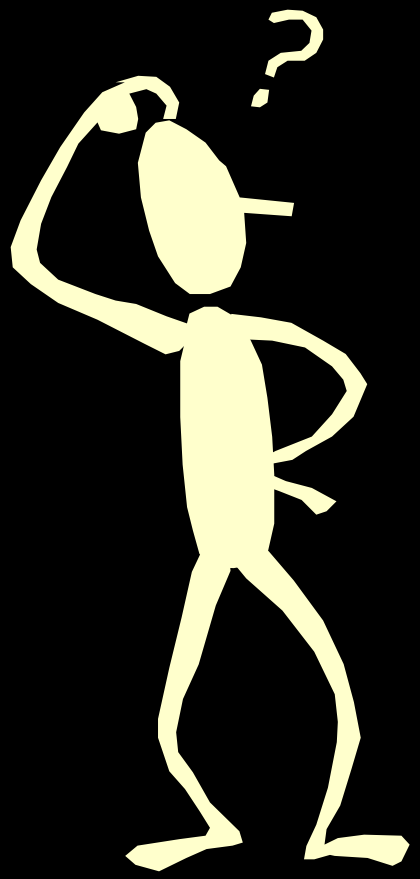
- Start with an idea
- Figure out how it can be done
- Graphics and sounds
- Program structure
- Algorithm
- Programming in C
- Compile and Test

# *Start with an idea*



Which new Coleco project to do?

# *Start with an idea*



- Use your imagination.
- Inspire yourself by looking around you.
- Do something which already exists to gain experience.

# *Example: Smash game*

- A ball
- A paddle
- A rebound effect
- Two sounds: rebound and lost
- A way to check if the ball is lost

# *Figure out how it could be done*

Hardware possibilities

# *Approach*

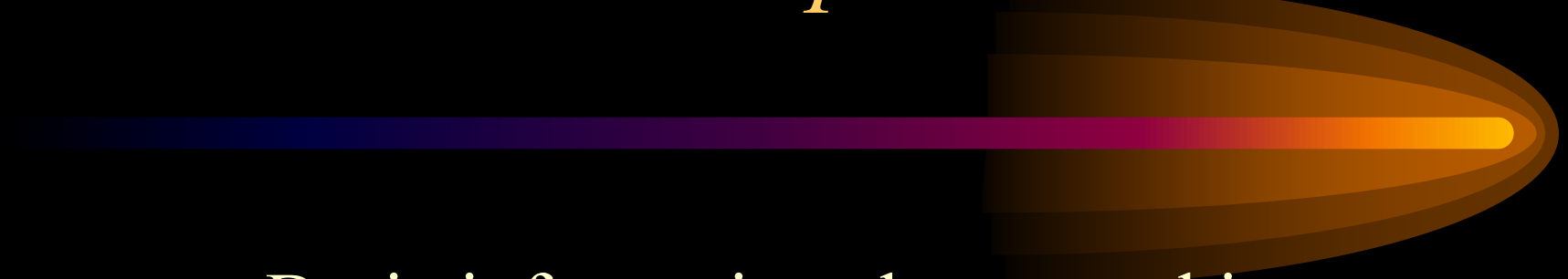
If you've read the ColecoVision specifications, you know the possibilities of this great game system.

- Avoid using too much RAM
- Limit of four (4) sprites in the same scanline
- Use screen mode 2 to do colorful graphics and a cool title screen
- No scrolling effects

# *Example: Smash game*

- Ball: character graphics
- Paddle: character graphics
- Sounds: tone generator + noise

# *Graphics and sound*



Basic information about graphics
and tools you can use

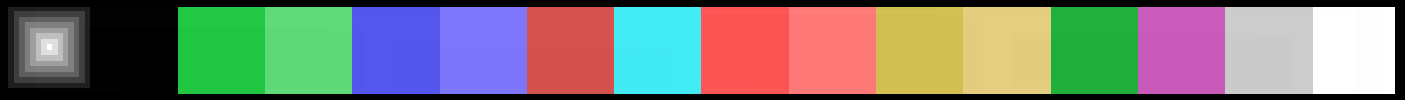
# *Graphics*

There are four screen modes available:

- Mode 0: 32x24 characters, two colors per 8 characters, sprites active
- Mode 1: 40x24 characters (6x8), 2 colors (forecolor and backcolor), sprites inactive
- Mode 2: 32x24 characters, 2 colors per 8 pixels, sprites active
- Mode 3: 64x48 pixels (4x4), sprites active

# *Color Palette*

- There are 15 colors plus a transparent color in the color palette.

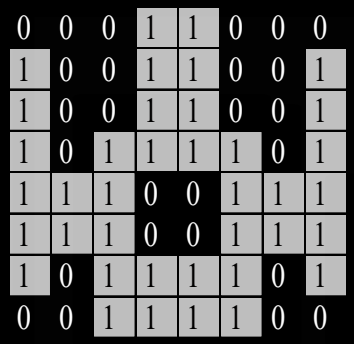


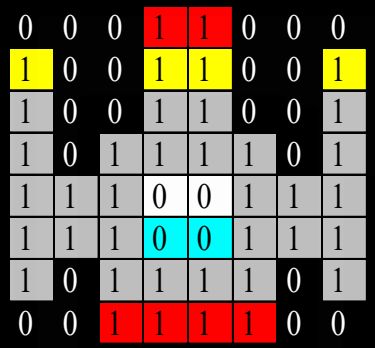
# *Graphics*

There are two kinds of graphics:

- Characters (tile graphics): They are useful to fill up the screen and are used to do bitmap title screens.
- Sprites (floating graphics): They can be anywhere on the screen but they use only one color. We can use more than one sprite (one per color) or a combination of sprites and characters to simulate a multicolor sprite.

# *Character Graphics*



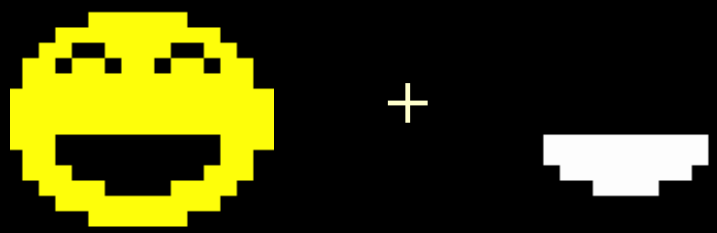



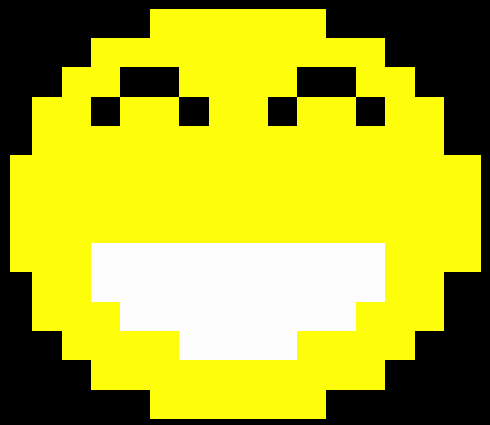

- Characters are 8x8 pixel graphics (except for screen mode 1 and 3)
- In screen mode 0 the characters are monochrome.
- In screen mode 2 the characters can be multi-color.

# *Character Graphics*

- The characters on screen are the result of three tables: NAME, PATTERN and COLOR.
- NAME: characters on screen (where they are placed on screen). A character can be duplicated on screen.
- PATTERN: character pattern
- COLOR: character color(s)

# *Sprite Graphics*





- Sprites are 8x8 or 16x16 pixel graphics (except for screen mode 1).
- Sprites can be magnified to look bigger (sprites pixels 2x2).

# *Sprites Graphics*

- The sprites on screen are the result of two tables: PATTERN and TABLE.
- PATTERN: sprite pattern
- TABLE: Y, X coordinate plus sprite pattern number and color. If the sprites are 16x16, the sprite pattern number must be 4x the real pattern number.
- Two sprites on screen can use the same pattern but it is not necessary for them to be the same color.

# *Example: Smash game*

| Character graphic: Ball | | | | | | | | Color | Pattern |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 40 | 3C |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 40 | 7E |
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 47 | 9F |
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 47 | 9F |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 40 | FF |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 40 | FF |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 40 | 7E |
| 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 40 | 3C |

- We use transparent, blue and cyan for the ball.

# *Example: Smash game*

| Paddle graphic: left | | | | | | | | Color | Pattern |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 80 | 7F |
| 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 89 | C3 |
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 89 | 9F |
| 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 89 | BF |
| 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 86 | FD |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 86 | F1 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 86 | C1 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 80 | 7F |

| Paddle graphic: block | | | | | | | | Color | Pattern |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | E0 | FF |
| | | | | | | | 1 | EF | 01 |
| 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | E7 | FB |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | E7 | 82 |
| 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | E7 | D8 |
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | E7 | 0A |
| 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | E7 | DF |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | E0 | FF |

| Paddle graphic: right | | | | | | | | Color | Pattern |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 80 | FE |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 89 | 83 |
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 89 | 9F |
| 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 89 | BF |
| 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 86 | FD |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 86 | F1 |
| 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 86 | C3 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 80 | FE |

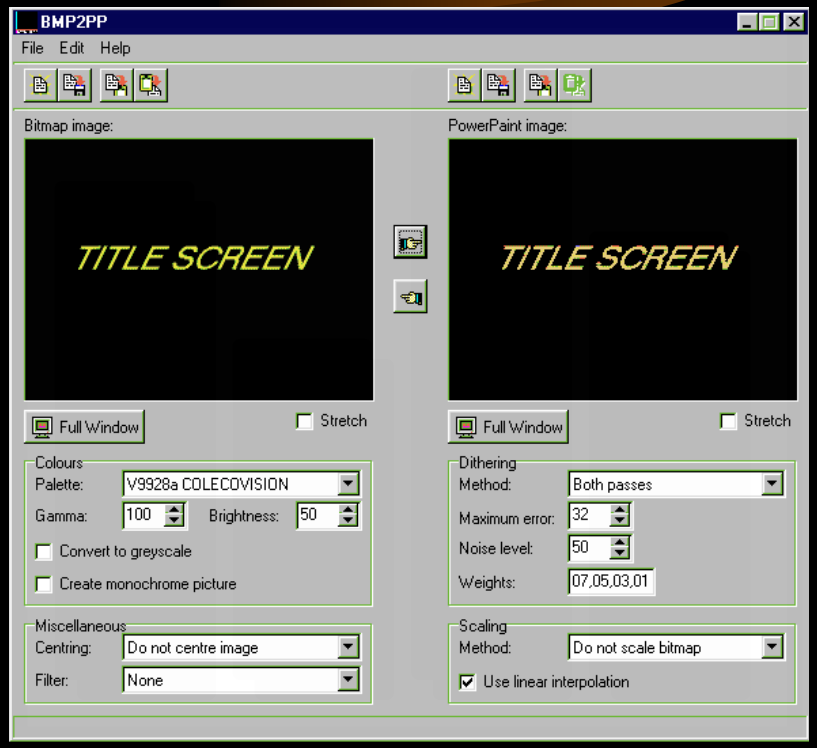
- We use pink and red for the extreme left and right of the paddle.
- We use grey, white and cyan for the middle part of the paddle.

# *Graphics tools*

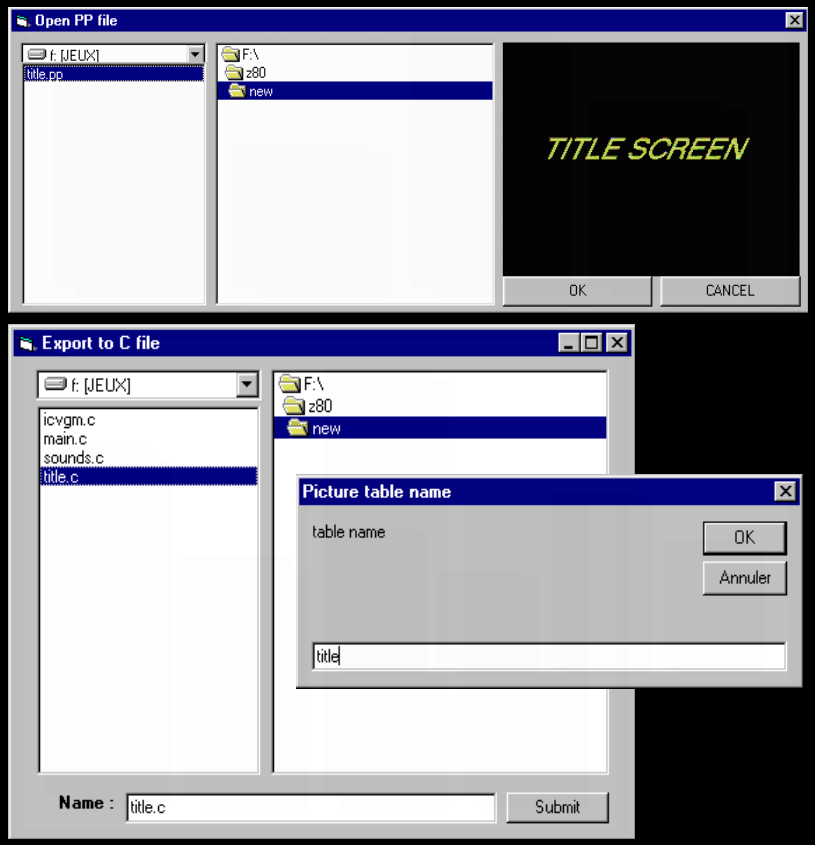
BMP2PP

by Marcel de Kogel

- Convert BMP files into PP (PowerPaint) files.
- Can use clipboard.



# *Graphics tools*




PP2C

by Daniel Bienvenu

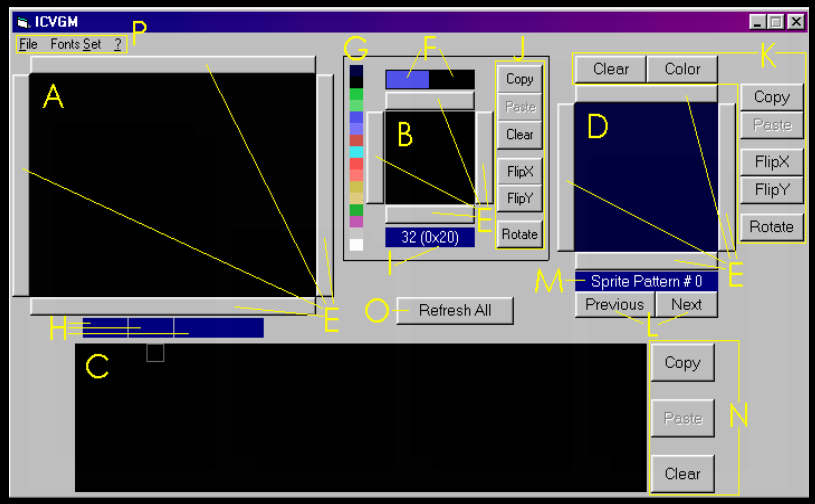
- Convert PP files into an array of HEX codes with compression (RLE).
- Generate a C file.

# *Graphics tools*

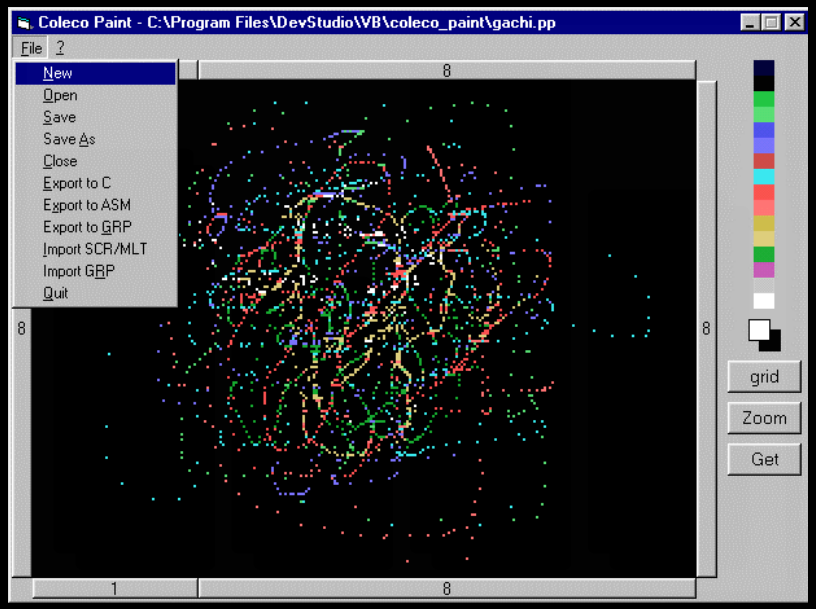
I.C.V.G.M.

by Daniel Bienvenu

- Create and edit a character set and sprites.



# *Graphics tools*




CVPAINT (beta)

by Daniel Bienvenu

- Load & Save PP files.
- Import ZX Spectrum pictures.
- Edit pixels within the limits of screen mode 2.
- Generate a C file.

# *Sounds*

There are four sound channels

- 3 tone channels: frequency + attenuation
- 1 noise channel: control + attenuation

# *Sounds*

## Tone channels

- The frequency is encoded in two bytes with the control registers
- Frenquency (in Hz) = 3 579 545 Hz / n
- n is the encoded value in two bytes
- The attenuation is encoded in 4 bits:
- (0000) = loud, (1111) = silence

# *Sounds*

## Noise channel

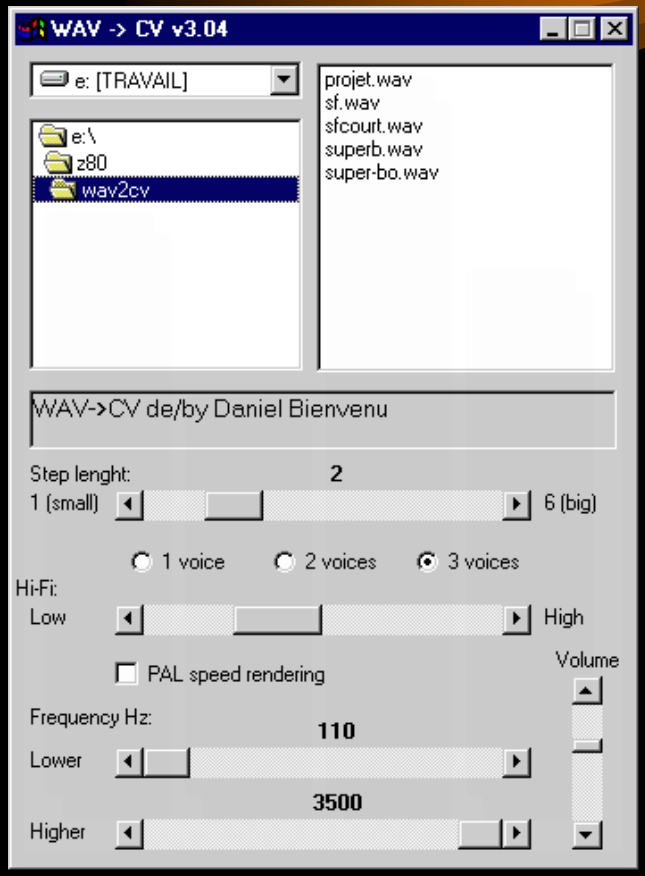
- There are only three frequencies available.
- To play more frequencies, the noise channel can use the tone channel 3 output.
- There are two modes to play noise: white and periodic.
- The attenuation works the same as the one for the tone channels (in 4 bits).

# *Sound tools*

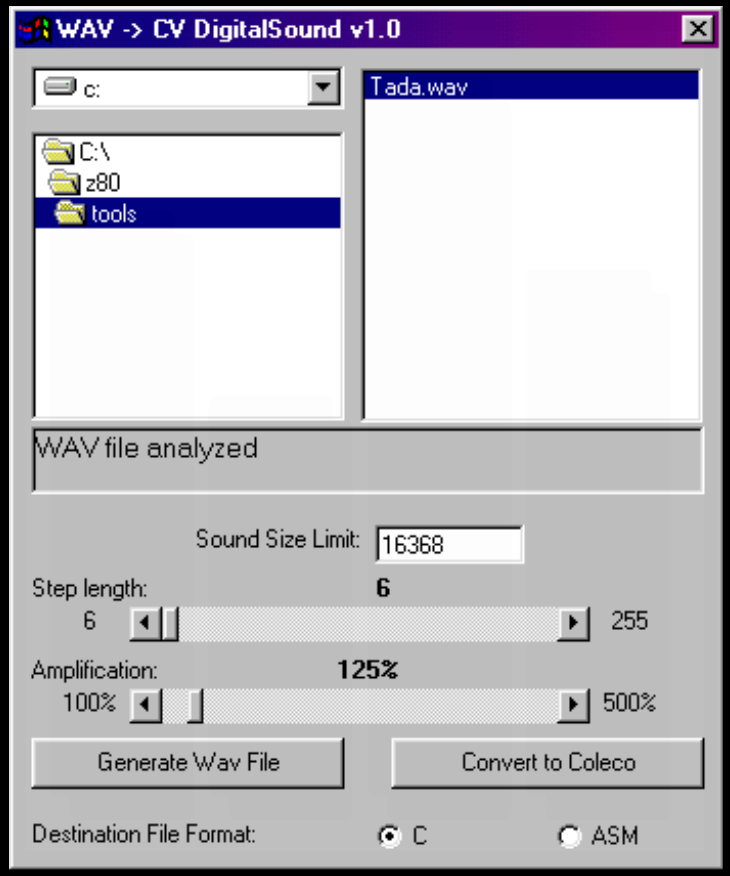
WAV2CV

by Daniel Bienvenu

- Convert mono WAV files into hex codes for the sound routines in the Coleco library.
- This tool uses the Fast Fourrier Transform



# *Sound tools*

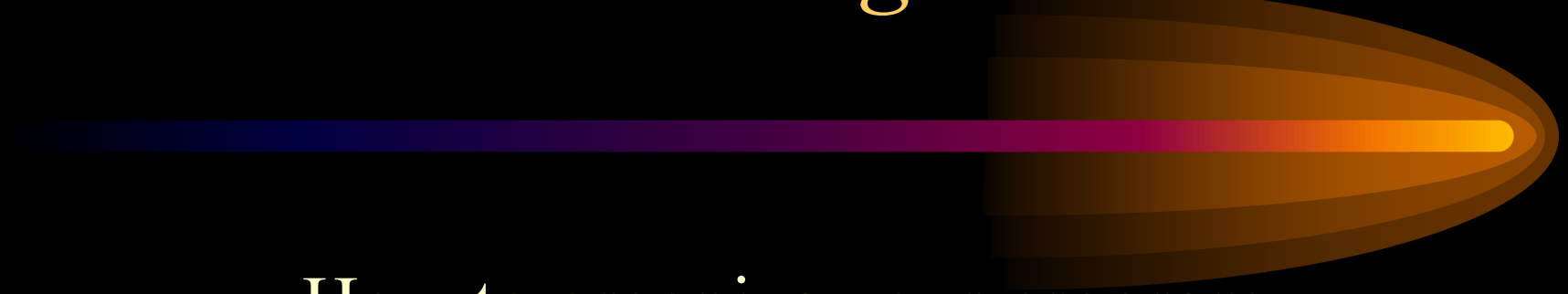



WAV2CVDS

by Daniel Bienvenu

- Convert mono WAV files into digital sound to be used with DSound library.
- Generate a C file.

# *Program Structure*



How to organize your program

# *How to organise your program:*

- Before starting to write your code, you can use paper to write and to draw what your game project will be.
- To help you, the following slides will show you a few things you can do to organize your idea on paper.

# *Storyboard*

- Necessary for big videogame projects, a storyboard is a set of pictures you draw on paper to get an idea of what your game will be.
- For simple videogames, a storyboard might be only one picture to represent the authors ' vision of the game.

# *Videogame parts*

- A videogame is divided into three (3) parts: opening, game engine, ending.
- Sometimes, a videogame has a story or a small animation to introduce the game.
- Sometimes, a game can show different endings.

# *Screens*

- A videogame project is divided into many screens: company logo(s), opening, title screen, menu, options, game screen(s), ending(s) and credits.
- The most important screens are: title screen, menu and game screen(s).

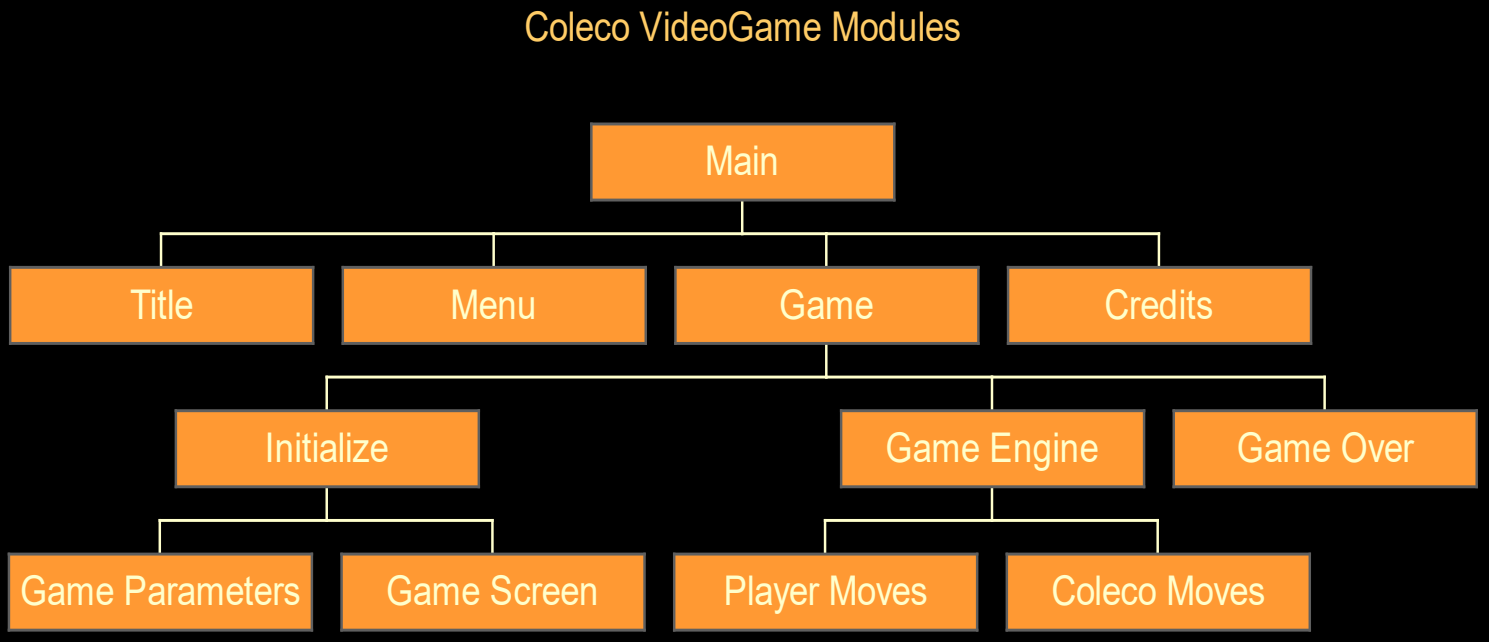
# *Important Screens*

- The title screen is the main entrance of the game: it should be professional.
- The menu screen is the best way to let the player modify the game parameters. It must be simple to use and undertstand.
- The game screens are what the gamer will use and remember most. To make a good impression, make them attractive.

# *Modules*

- A module is only a term I personally use to designate « a section of code » to do « a particular job » in the program.
- A module may need external data.
- A module can be in another C file.
- Generaly, a module is coded in a single routine.

# *Modules Sample Diagram*

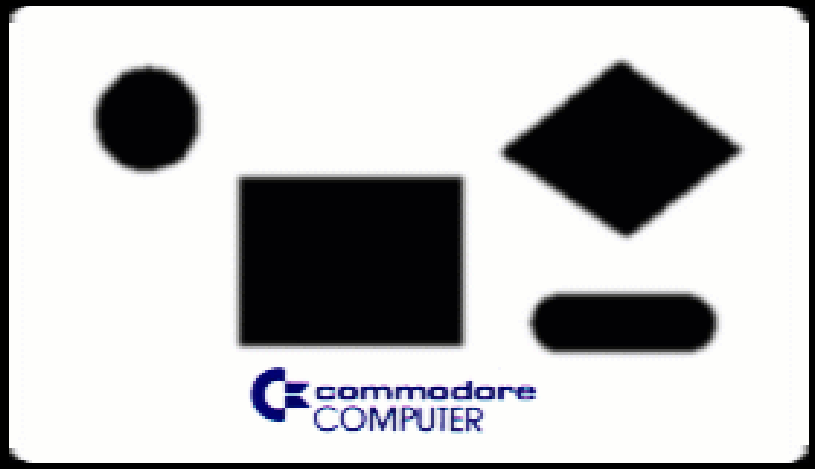


# *Flow Charts*

- A flow chart is a diagram which represents the normal execution of (a part of ) your code.
- This particular diagram uses a specific formalism with symbols like rectangles, rounded rectangles, circles and more.
- Using flow charts to represent the modules of your code is a good idea.

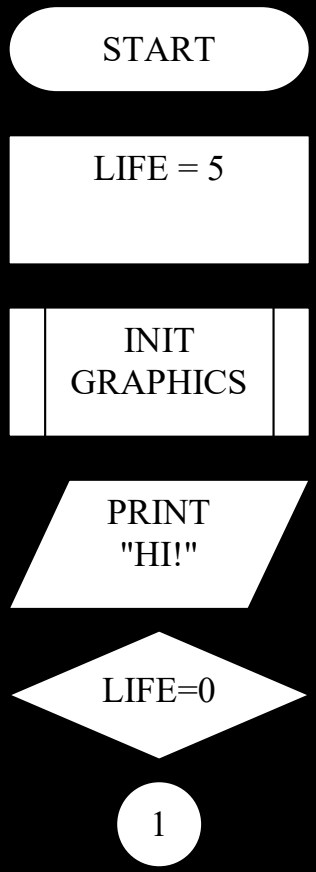
# *Flow Charts*

In the past, we used this kind of stencil to facilitate drawing flow charts on paper.




Now, with the computers, we use software to do exactly the same thing.

# *Flow Charts*




These symbols are the most common ones.

They are simple geometric graphics with text.

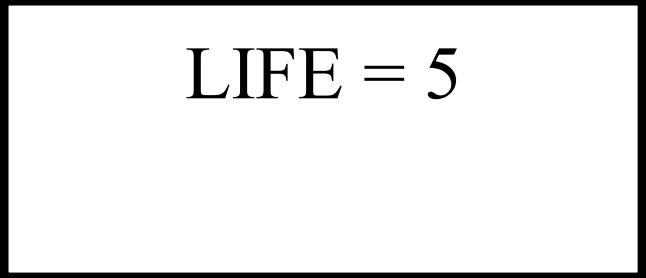
To see the execution path, the convention is to use arrows between symbols.

# *Flow Chart Symbols*

- This symbol is used to specify where your code starts and finishes.

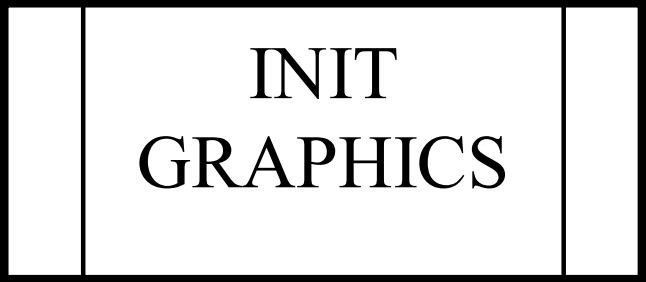
# *Flow Chart Symbols*

- This symbol is used to do a mathematical operation or to initialize a variable.



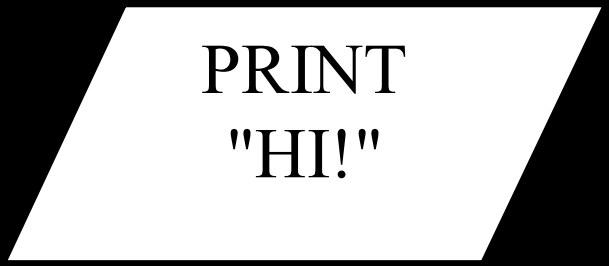
# *Flow Chart symbols*

- This symbol is used to avoid using too many symbols in your diagram. It can represent a module you call.



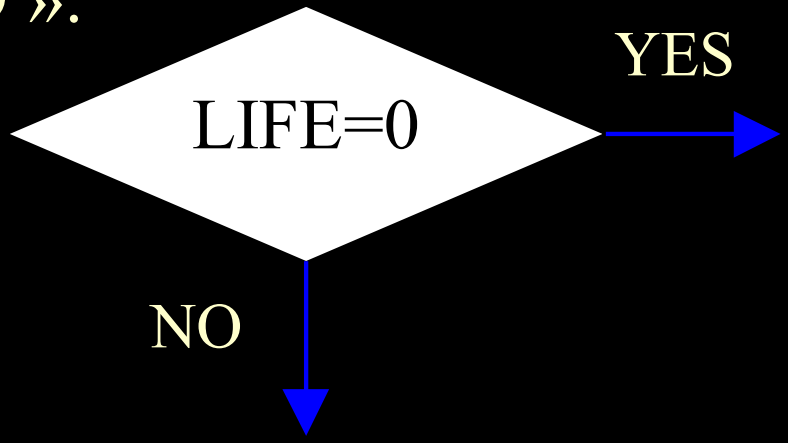
# *Flow Chart Symbols*

- This symbol is used for the output instructions. It may represent one or more lines of code in your program.



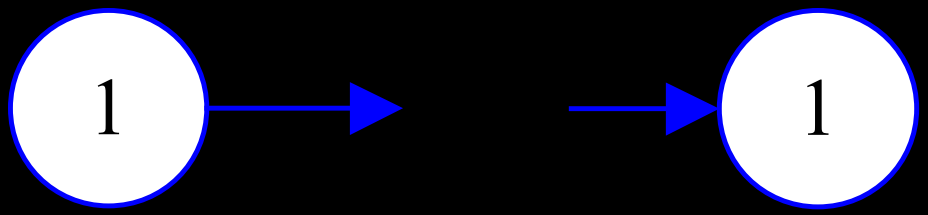
# *Flow Chart Symbols*

- This symbol is used for conditions. After this symbol, two arrows most be added: one for « YES », the other for « NO ».



# *Flow Chart Symbols*

- And this little symbol represents a label. You can use labels to mark where a jump is possible.




Label Go to Label

# *List of variables*

- Before doing a detailed flow chart, you must do a list of variables the game needs.
- Example: the variables for a bouncing ball on screen could be its coordinate and its orientation + speed (simply the incrementation in X and Y axes).
- A list of variables helps you to not omit important variables.


Between the idea and the code

# *What is an algorithm?*

- It ’s a representation of your code in a natural language.
- It uses keywords called « pseudo-code » to refer to the instructions you will need to write in the code.
- A flow chart is a graphical version of an algorithm.

# *Algorithm*

- An algorithm is a series of orders you give to the computer to be executed.
- To help your imagination, this is a sample algorithm to order a robot to buy apples at the market.

« Go outside. Turn Left. Walk to next corner. Turn left again. Walk 10 feet. Open door. Walk 3 feet. Buy apples. Come back. »

# *Pseudo-Code*

- There is no convention for pseudo-code but it must be something in natural language which represents instructions.

Ex.: « for each y position from 0 to 10 »

- When the time comes to write your code based on the algorithm, converting keywords to instructions is easy.

Ex.: « for (y=0;y<=10;y++) »

# *Programming in C*

General information about the
C programming language

# *Warning!*

- First of all, I need to specify that the C language I ’m talking here is not the C++ or C# language but the standard ANSI C.
- The following slides are only a very small introduction to the C programming. For more information, read books about C language.

# *C programming*

- A C file structure is simple if you understand this: what you need must be previously given. That is to say, no forward declarations
- At the top, you have the libraries, the constants, the headers of external routines, tables and global variables.
- At the bottom, you have routines and then routines who need others routines to run.

# *C programming*

```
#include<coleco.h>

#define potatoes 10

extern byte title[];

byte numbers[]={1,2,3};

byte numbers[3];
```

- Libraries are included in your C file with the instruction « #include »
- The constants are defined in your C file with the instruction « #define »

# *C programming*

```
#include<coleco.h>

#define potatoes 10

extern byte title[];

byte numbers[]={1,2,3};

byte numbers[3];
```

- The tables which are in another C file are included with the instruction « extern »

# *C programming*

```
#include<coleco.h>

#define potatoes 10

extern byte title[];

byte numbers[]={1,2,3};

byte numbers[3];
```

- The first array of bytes named ‘numbers’ here, is a typical example of a ROM table.
- The second array of bytes named ‘numbers’ here is a typical example of a RAM table.

# *Data types*

- char : character or signed short integer in one byte.
- byte : unsigned short integer in one byte.
- int : signed integer in two bytes.
- unsigned : unsigned integer in two bytes.
- [] : array
- char [] : array of char or string

# *Operators*

- Arithmetic operators: +, -, *, /, % (modulus)
- Increment ++ and Decrement --
- Bitewise operators: << (shift left), >> (shift right), & (and), | (or), ^ (x-or), ! (not)
- Combined operators: <variable> = <variable><operator><expression> become <variable><operator>=<expression>. Ex.: a=a+2 become a+=2

# *Operators*

- Relational operators: > (greater than), < (less than), == (equal), >= (greater than or equal), <= (less than or equal), != (not equal)
- Logical operators: && (and), || (or), ! (not)

# *If Statement*

- Simple if

```
if (value) statement1;
```

- if... else

```
if (value) statement1;
else statement2;
```

- A statement can be inside brakets: { and }

# *Loops*

There are many kinds of loops.

- for loop

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

- while loop

```
while (x<10) {}
```

- do... while loop

```
do {} while (x<10)
```

# *Functions*

- A function may return a value or not (void).
- A function may need parameters to execute.
- A function header syntax is:

<return type> function_name (<data type> parameter#1, <data type> parameter#2, ...)

int substract (int a, int b)

void main(void)

# *Functions*

- A function core is written in braces: { and }
- At the top we have the temporary variables used within the scope of a particular function.
- After the temporary variables, we add instructions (operations and command lines)
- When needed, we use braces again to group many instructions together.

# *Function Sample*

```
byte sum(byte l)
{
 byte j;
 byte k;

 k=0;
 for(j=1;j<=l;j++)
 {
   k += j;
 }
 return k;
}
```

- This routine needs a byte value and returns a byte value.
- The temporary variables j and k are declared at the top of the routine.
- And finally, the operations.

# *I know…*

It ’s not enough information to start programming in C. You ’ll have to read books to learn C language.

Now, let ’s talk about Coleco programming

# *Coleco programming in C*

What you need to do a new Coleco project:

- the coleco library
- an nmi routine (empty or not)
- a main routine
- and a lot of routines for your own purposes
- You can use more libraries
- You can use more than one C file

# *Coleco programming in C*

The execution steps must be something like:

- Initialize the video display (VDP registers and Video memory)
- Initialize global variables
- Start doing operations and commands
- Use a loop for the game engine
- Use variables to output things on screen
- Use a delay to slowdown the execution

# *Libraries*

The Coleco library by Marcel de Kogel is the SDK to program new Coleco projects in C.

Getput1 library is a toolbox with a lot of useful routines. This library uses coleco library routines.

But, because we have no time, I suggest to read the Coleco programming documentation to find more information about these libraries.

# *Compile and Test*

How to use CCI software in the Coleco development kit?

# *Step by step*

- Create a project directory as a sub-directory of the compiler.
- Save all your C files in your project directory.
- Copy CCI software in your project directory.
- Run CCI to compile and link your project.

# *Example*

We will do now a new Coleco project together.

Follow the instructions in the next slides to experience the joy of compiling a new Coleco project.

I hope you have un-archived the Coleco development kit on your PC.

Open Notepad. Good luck!

# *Sample code: « Hello World »*

- This code is the starting point for all new ColecoVision projects in C.
- #include <coleco.h> is the command to include the Coleco library by Marcel de Kogel in your project.

```
#include <coleco.h>

/* EMPTY NMI ROUTINE */


/* EMPTY MAIN ROUTINE*/
```

The nmi routine is needed to use the Coleco library… even if you don't use it.

The main routine is the starting point of the program.

# *Sample code: « Hello World »*

```
#include <coleco.h>
#include <getput1.h>

/* EMPTY NMI ROUTINE */


/* EMPTY MAIN ROUTINE*/
```

- Getput1 library is a toolkit with a lot of graphic routines and more.
- This library helps you to simplify code by grouping commonly used functions.
- You must use it.

# *Sample code: « Hello World »*

```
#include <coleco.h>
#include <getput1.h>

/* EMPTY NMI ROUTINE */


/* MAIN ROUTINE*/
```

- You need to initialize the VDP registers to setup the screen.
- Fortunately, Getput1 helps us. This library contains a routine named screen_mode_2_text.

# *Sample code: « Hello World »*

```
#include <coleco.h>
#include <getput1.h>

/* EMPTY NMI ROUTINE */


/* MAIN ROUTINE*/
```

- Now, you need to initialize the characters set in video memory.
- Again, the Getput1 library helps with another routine named upload_default_ascii.

# *Sample code: « Hello World »*

```
#include <coleco.h>
#include <getput1.h>

/* EMPTY NMI ROUTINE */


/* MAIN ROUTINE*/
```

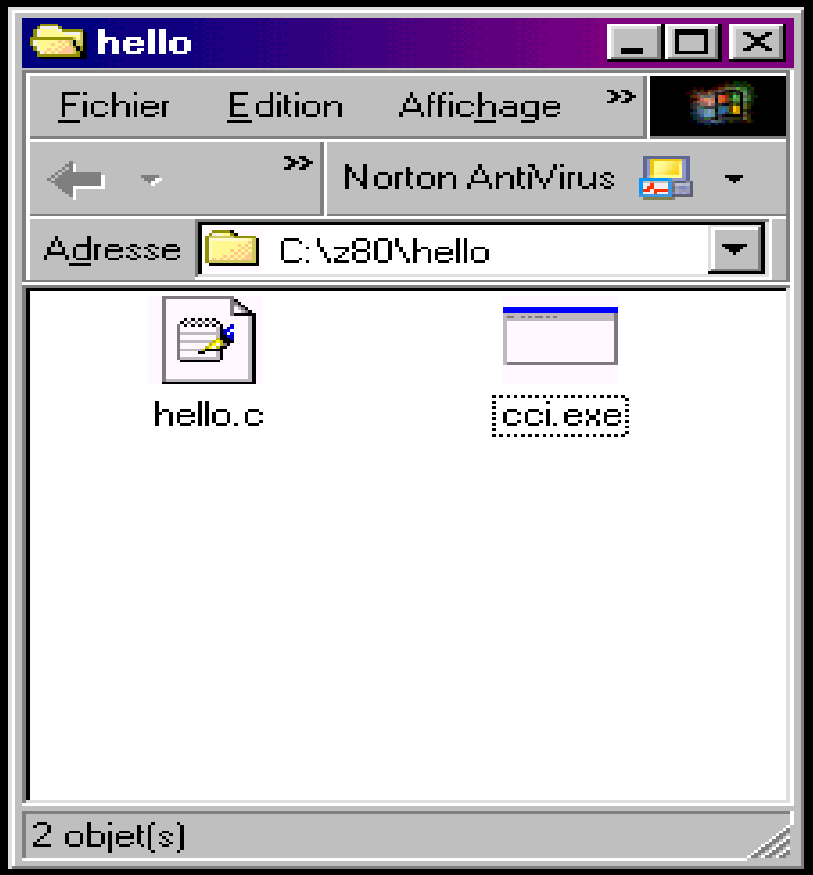
- Now, you are ready to print on screen.
- Use cls to clear screen
- Use center_string to center a string on screen.
- Use print_at to print on screen at a particular coordinate.

# *Sample code: « Hello World »*

```
#include <coleco.h>
#include <getput1.h>

/* EMPTY NMI ROUTINE */



/* MAIN ROUTINE*/



infiniteloop:
        infiniteloop
```

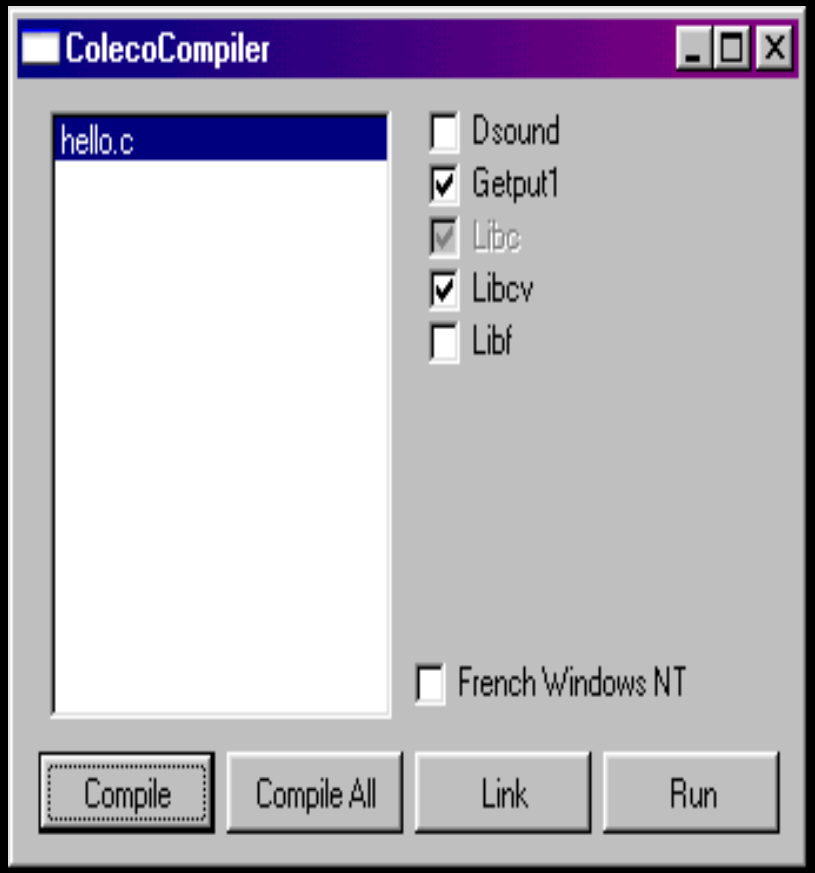
- Finaly, you must suspend normal execution to let the user see the messages.
- Use delay to suspend normal execution for a specified amount of time.
- Use pause to wait for a fire button.
- Use an infinite loop

# *Sample code: « Hello World »*



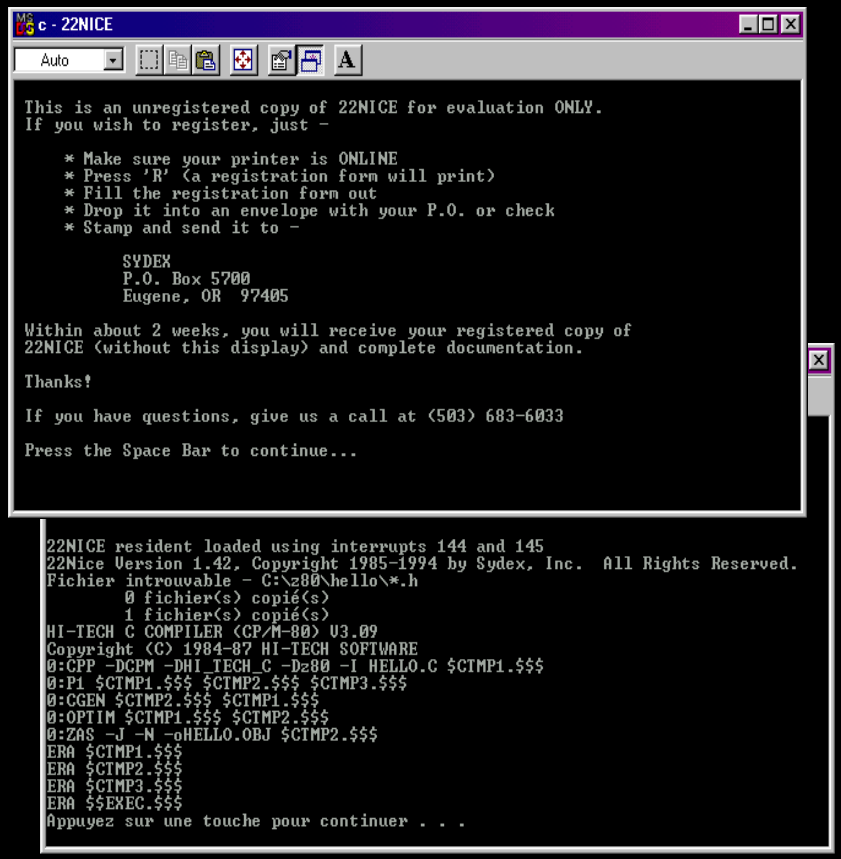

- Create a sub-directory in the compiler directory. This new directory is your project directory.
- Save your code as a C file in your project directory and add the CCI software.

# *Sample code: « Hello World »*



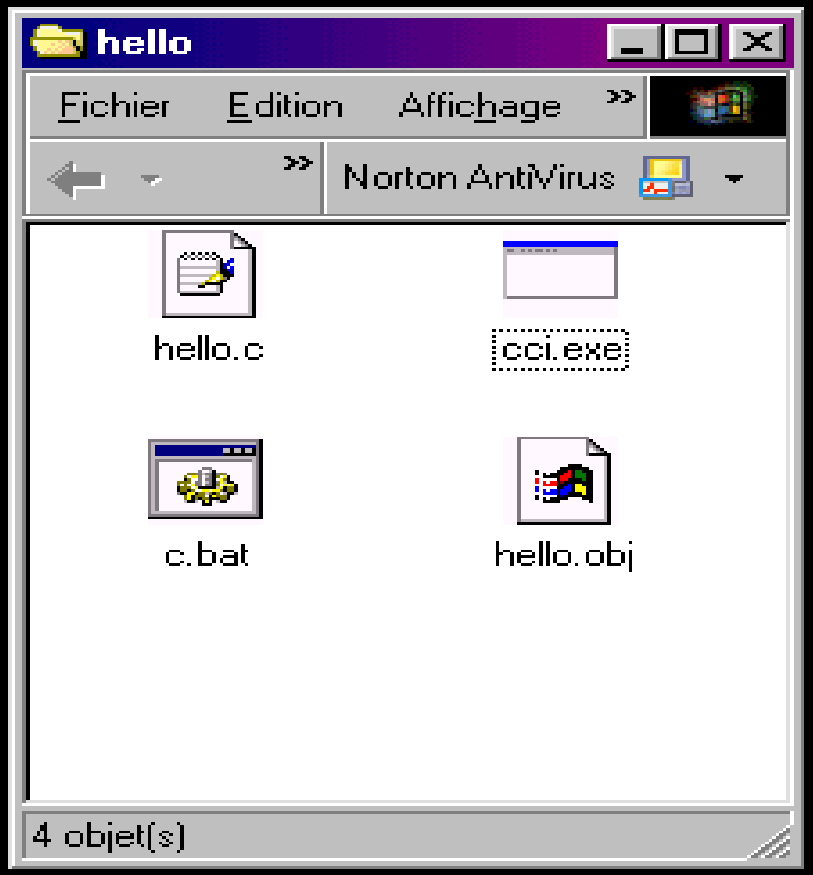

- Check Getput1 checkbox to let CCI use this library.
- Select your C file in the filelist box and click on Compile.

# *Sample code: « Hello World »*



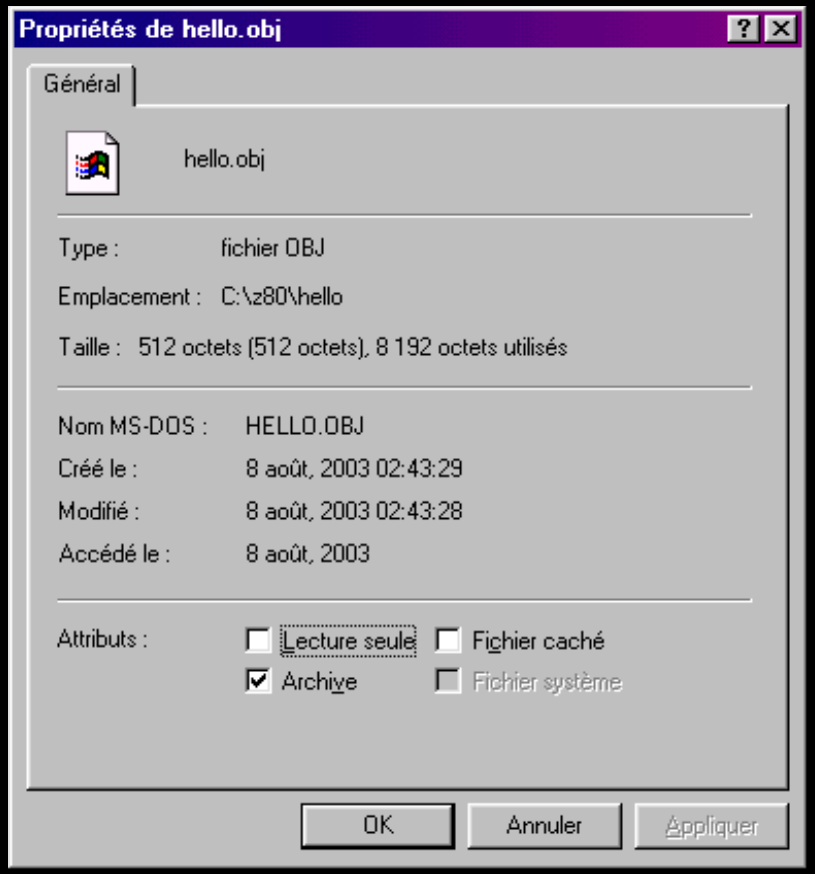

- A popup DOS window appear with 22NICE running.
- After pressing space bar, the compiler will start.
- After compiling your code, the DOS window waits. Close this window.

# *Sample code: « Hello World »*



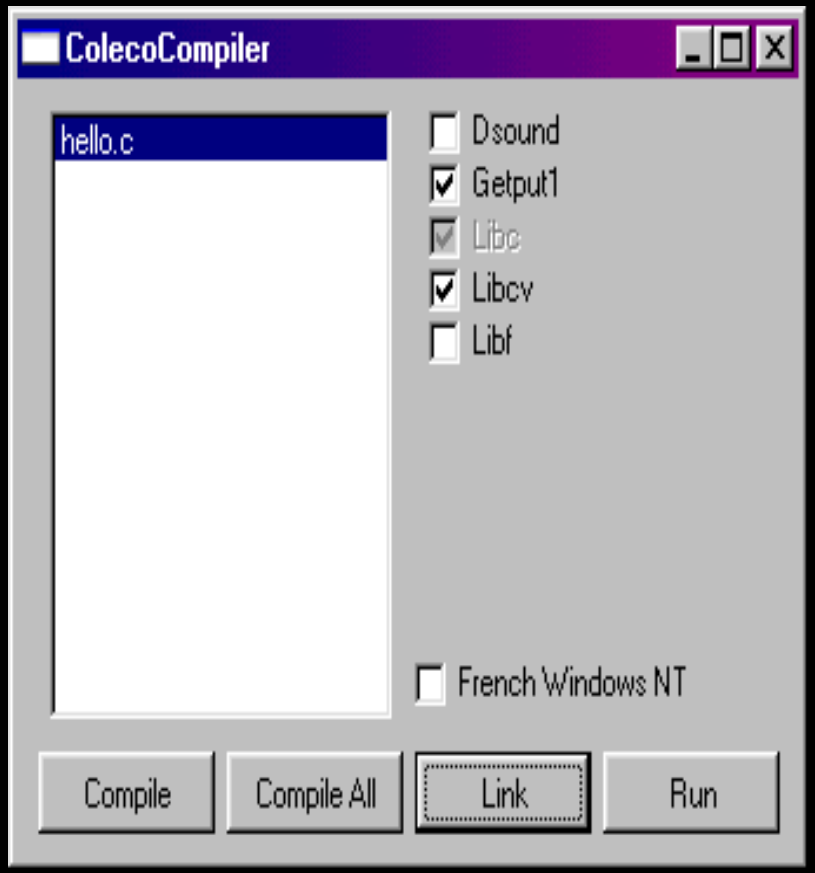

- CCI created a batch file named « c.bat »
- The compiler generated an object file named « hello.obj »

# *Sample code: « Hello World »*



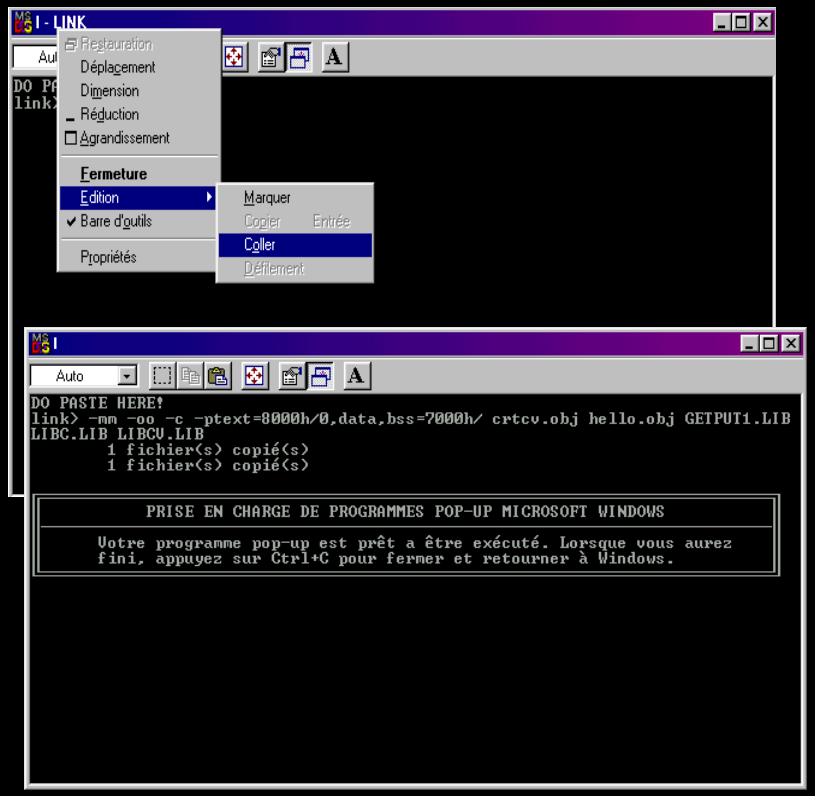

- Before using the linker, you must verify the filesize of the objets files.
- 512 bytes… it ’s very small but it ’s not zero.

# *Sample code: « Hello World »*




- Now, it ’s time to use the linker.
- Click on the Link button of the CCI software.

# *Sample code: « Hello World »*

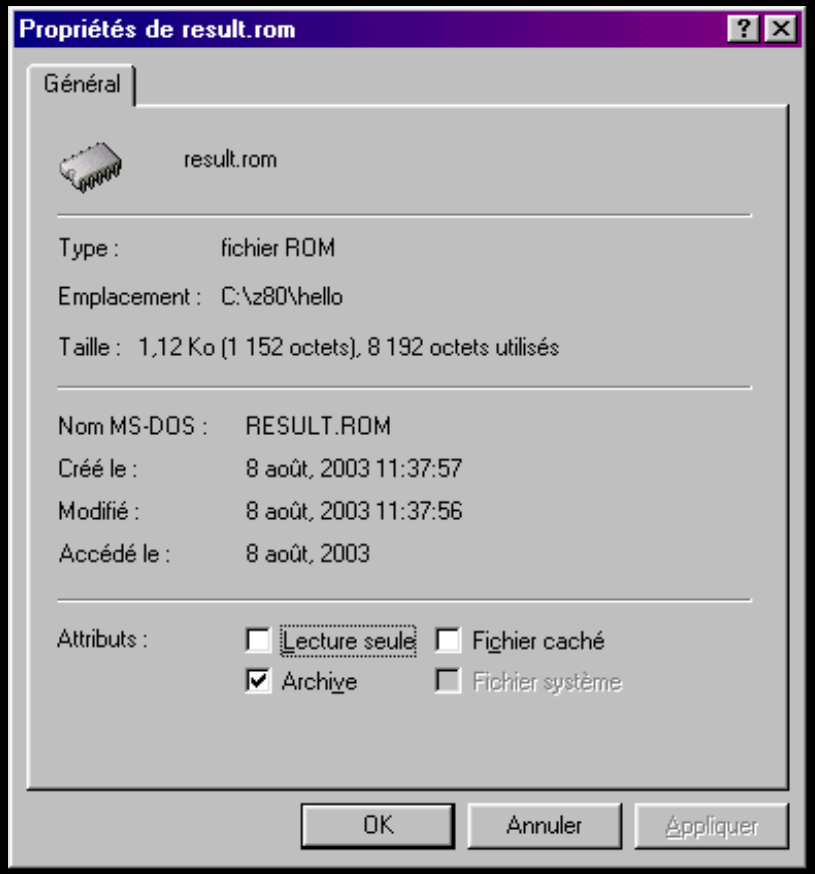



- 22NICE starts again.
- The linker is waiting for instructions.
- CCI will have copied the instructions to the clipboard.
- Do a « paste » in the DOS window.

# *Sample code: « Hello World »*


- CCI created a batch file named « l.bat »
- The linker generated two files: « map.txt » with the memory map information, and « result.rom » which is the rom file we wanted to create.

# *Sample code: « Hello World »*




- If Windows closed your DOS window, you may not see if the linker was able to link right the object files together with the libraries.
- To be sure, check the filesize of the rom file.

# *Sample code: « Hello World »*


- If you didn ’t close the DOS windows, do it now.
- Click on the « Run » button to start VirtualColeco.
- Well, it looks like you did it! :)

# *It 's working! :)*

- Now, you know how to use CCI. It 's time for you to program a bigger project.
- Note: A good way to do a project is to compile and link your code each time you update it.


Be prepared for the next:
New ColecoVision Programming
presentation

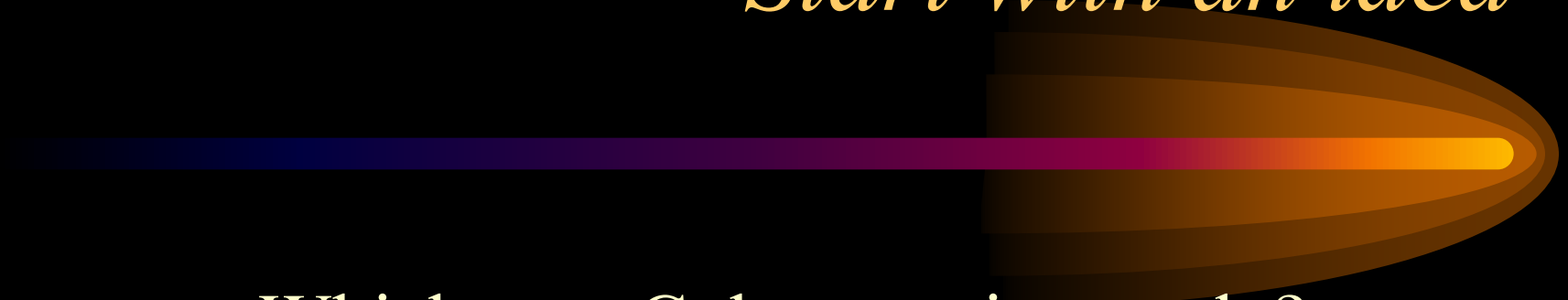
# *New ColecoVision Programming*

Programming a new ColecoVision game using Windows and DOS

# *Step by Step*

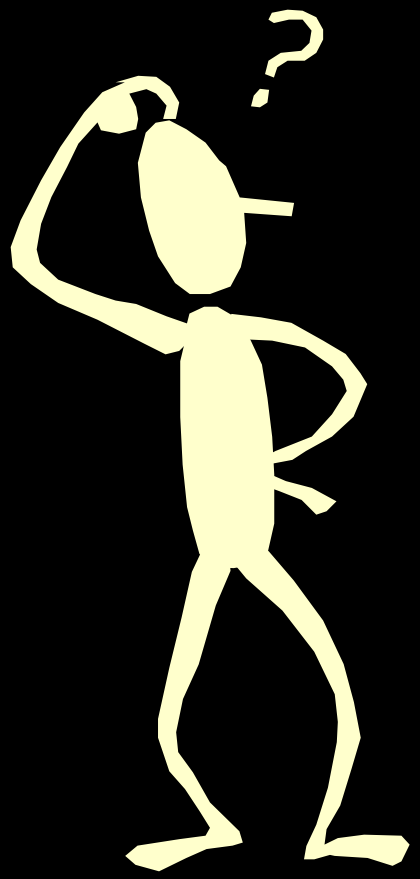
- Start with an idea
- Figure out how it can be done
- Graphics and sounds
- Program structure
- Algorithm
- Programming in C
- Compile and Test

# *Start with an idea*



Which new Coleco project to do?

# *Start with an idea*



- Use your imagination.
- Inspire yourself by looking around you.
- Do something which already exists to gain experience.

# *Example: Smash game*

- A ball
- A paddle
- A rebound effect
- Two sounds: rebound and lost
- A way to check if the ball is lost

# *Figure out how it could be done*

Hardware possibilities

# *Approach*

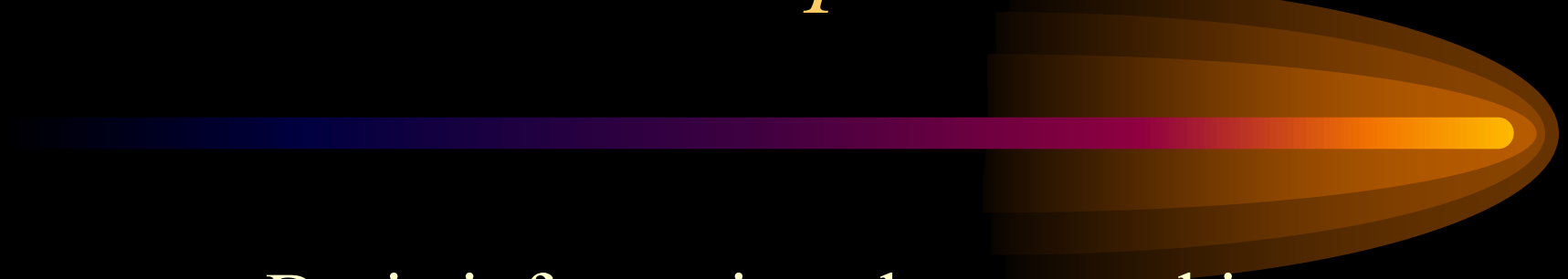
If you've read the ColecoVision specifications, you know the possibilities of this great game system.

- Avoid using too much RAM
- Limit of four (4) sprites in the same scanline
- Use screen mode 2 to do colorful graphics and a cool title screen
- No scrolling effects

# *Example: Smash game*

- Ball: character graphics
- Paddle: character graphics
- Sounds: tone generator + noise

# *Graphics and sound*



Basic information about graphics
and tools you can use

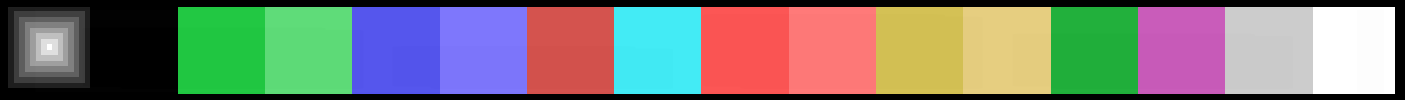
# *Graphics*

There are four screen modes available:

- Mode 0: 32x24 characters, two colors per 8 characters, sprites active
- Mode 1: 40x24 characters (6x8), 2 colors (forecolor and backcolor), sprites inactive
- Mode 2: 32x24 characters, 2 colors per 8 pixels, sprites active
- Mode 3: 64x48 pixels (4x4), sprites active

# *Color Palette*

- There are 15 colors plus a transparent color in the color palette.

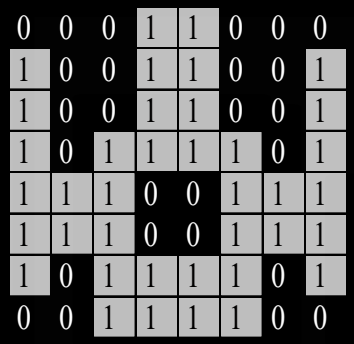


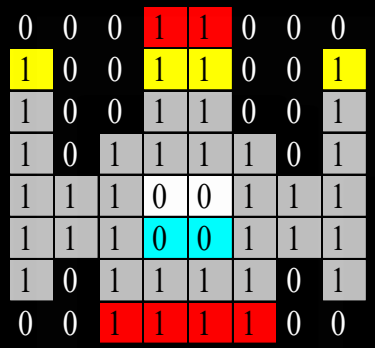
# *Graphics*

There are two kinds of graphics:

- Characters (tile graphics): They are useful to fill up the screen and are used to do bitmap title screens.
- Sprites (floating graphics): They can be anywhere on the screen but they use only one color. We can use more than one sprite (one per color) or a combination of sprites and characters to simulate a multicolor sprite.

# *Character Graphics*



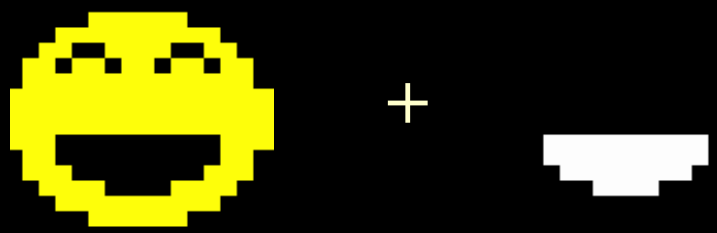



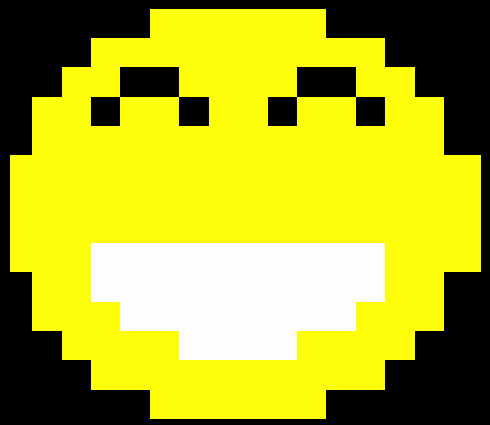

- Characters are 8x8 pixel graphics (except for screen mode 1 and 3)
- In screen mode 0 the characters are monochrome.
- In screen mode 2 the characters can be multi-color.

# *Character Graphics*

- The characters on screen are the result of three tables: NAME, PATTERN and COLOR.
- NAME: characters on screen (where they are placed on screen). A character can be duplicated on screen.
- PATTERN: character pattern
- COLOR: character color(s)

# *Sprite Graphics*





- Sprites are 8x8 or 16x16 pixel graphics (except for screen mode 1).
- Sprites can be magnified to look bigger (sprites pixels 2x2).

# *Sprites Graphics*

- The sprites on screen are the result of two tables: PATTERN and TABLE.
- PATTERN: sprite pattern
- TABLE: Y, X coordinate plus sprite pattern number and color. If the sprites are 16x16, the sprite pattern number must be 4x the real pattern number.
- Two sprites on screen can use the same pattern but it is not necessary for them to be the same color.

# *Example: Smash game*

| Character graphic: Ball | | | | | | | | Color | Pattern |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 40 | 3C |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 40 | 7E |
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 47 | 9F |
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 47 | 9F |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 40 | FF |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 40 | FF |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 40 | 7E |
| 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 40 | 3C |

- We use transparent, blue and cyan for the ball.

# *Example: Smash game*

| Paddle graphic: left | | | | | | | | Color | Pattern |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 80 | 7F |
| 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 89 | C3 |
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 89 | 9F |
| 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 89 | BF |
| 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 86 | FD |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 86 | F1 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 86 | C1 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 80 | 7F |

| Paddle graphic: block | | | | | | | | Color | Pattern |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | E0 | FF |
| | | | | | | | 1 | EF | 01 |
| 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | E7 | FB |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | E7 | 82 |
| 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | E7 | D8 |
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | E7 | 0A |
| 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | E7 | DF |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | E0 | FF |

| Paddle graphic: right | | | | | | | | Color | Pattern |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 80 | FE |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 89 | 83 |
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 89 | 9F |
| 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 89 | BF |
| 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 86 | FD |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 86 | F1 |
| 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 86 | C3 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 80 | FE |

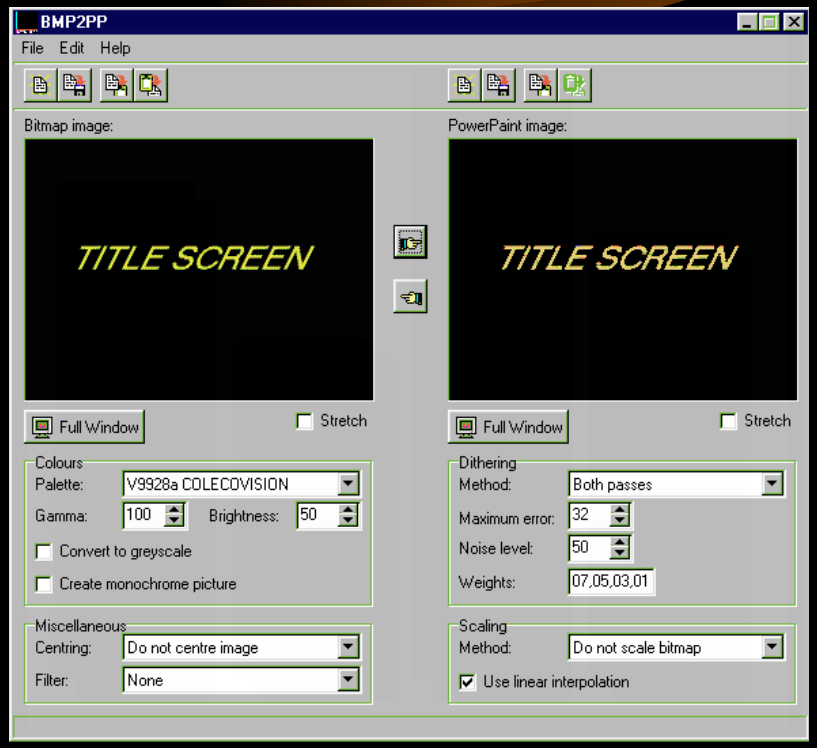
- We use pink and red for the extreme left and right of the paddle.
- We use grey, white and cyan for the middle part of the paddle.

# *Graphics tools*

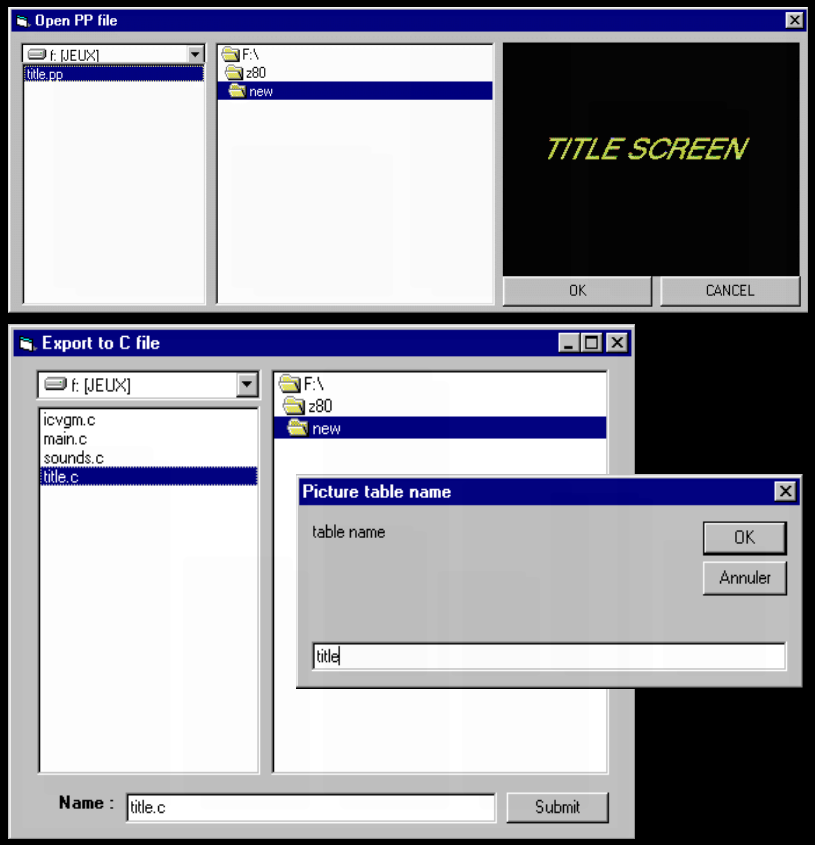
BMP2PP

by Marcel de Kogel

- Convert BMP files into PP (PowerPaint) files.
- Can use clipboard.



# *Graphics tools*




PP2C

by Daniel Bienvenu

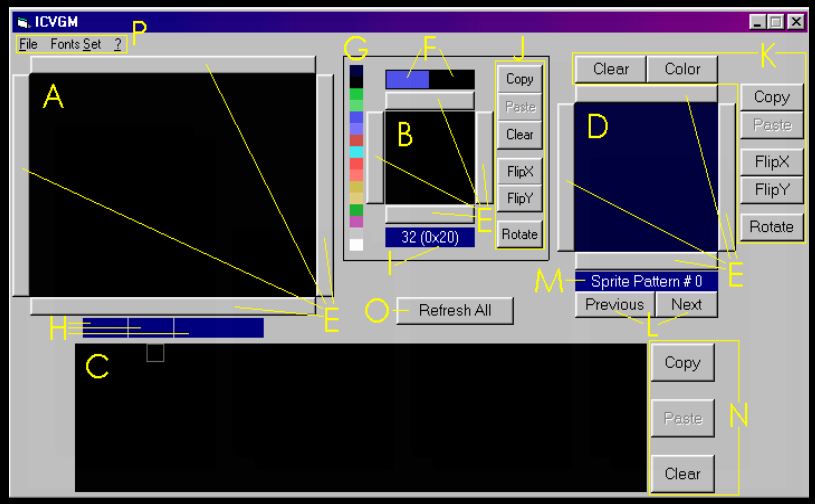
- Convert PP files into an array of HEX codes with compression (RLE).
- Generate a C file.

# *Graphics tools*

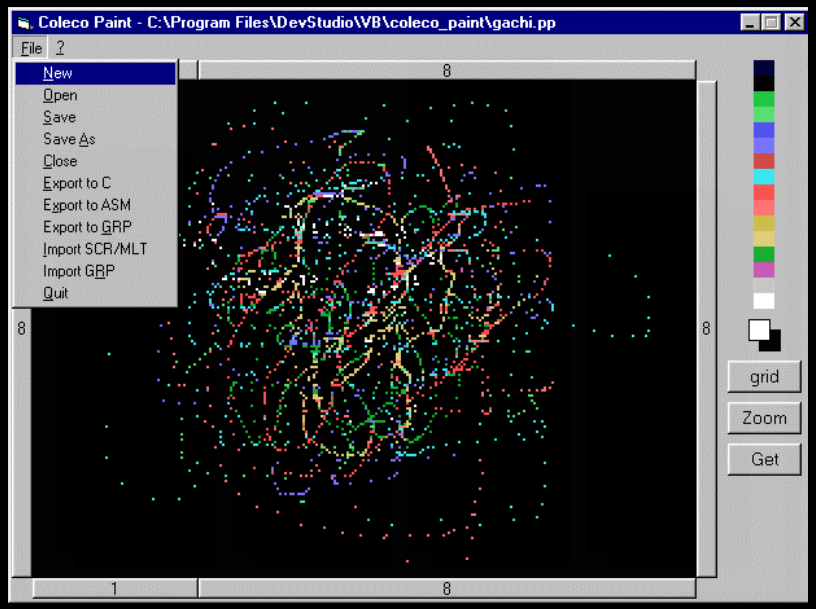
I.C.V.G.M.

by Daniel Bienvenu

- Create and edit a character set and sprites.



# *Graphics tools*




CVPAINT (beta)

by Daniel Bienvenu

- Load & Save PP files.
- Import ZX Spectrum pictures.
- Edit pixels within the limits of screen mode 2.
- Generate a C file.

# *Sounds*

There are four sound channels

- 3 tone channels: frequency + attenuation
- 1 noise channel: control + attenuation

# *Sounds*

## Tone channels

- The frequency is encoded in two bytes with the control registers
- Frenquency (in Hz) = 3 579 545 Hz / n
- n is the encoded value in two bytes
- The attenuation is encoded in 4 bits:
- (0000) = loud, (1111) = silence

# *Sounds*

## Noise channel

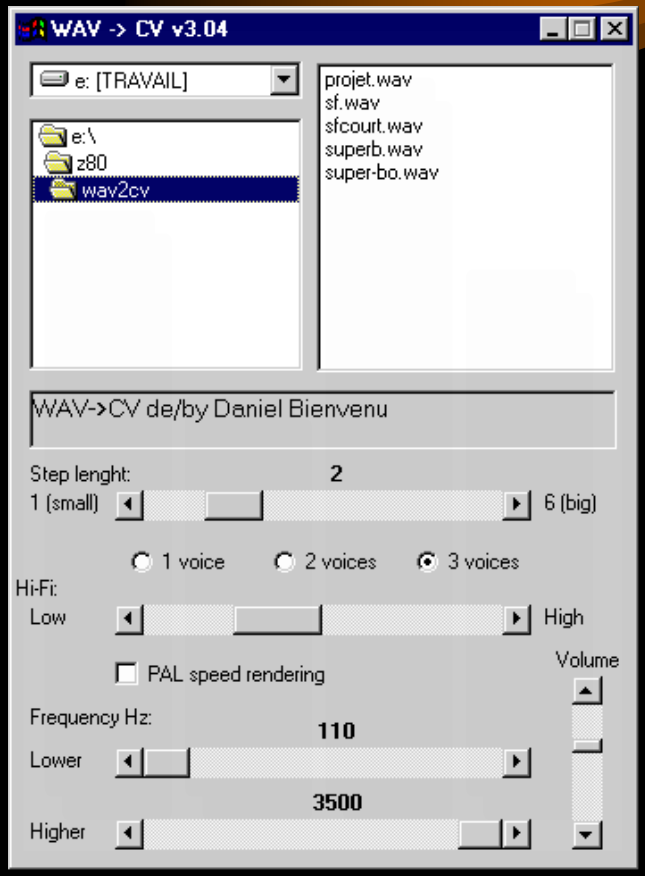
- There are only three frequencies available.
- To play more frequencies, the noise channel can use the tone channel 3 output.
- There are two modes to play noise: white and periodic.
- The attenuation works the same as the one for the tone channels (in 4 bits).

# *Sound tools*

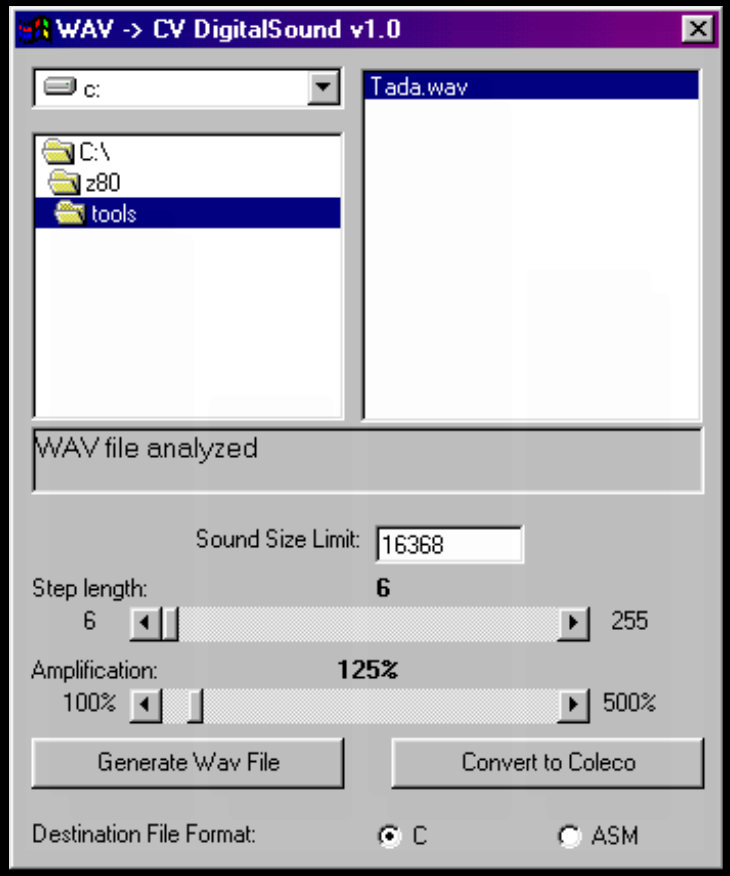
WAV2CV

by Daniel Bienvenu

- Convert mono WAV files into hex codes for the sound routines in the Coleco library.
- This tool uses the Fast Fourrier Transform



# *Sound tools*

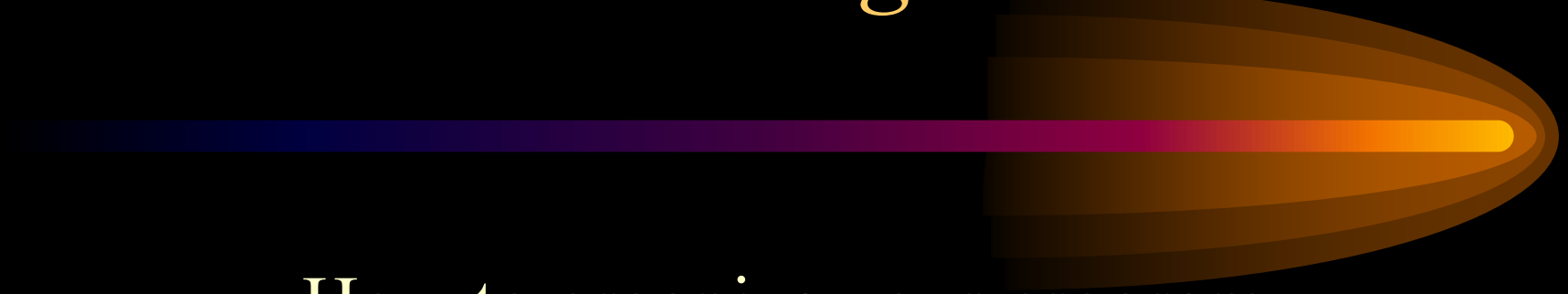



WAV2CVDS

by Daniel Bienvenu

- Convert mono WAV files into digital sound to be used with DSound library.
- Generate a C file.

# *Program Structure*



How to organize your program

# *How to organise your program:*

- Before starting to write your code, you can use paper to write and to draw what your game project will be.
- To help you, the following slides will show you a few things you can do to organize your idea on paper.

# *Storyboard*

- Necessary for big videogame projects, a storyboard is a set of pictures you draw on paper to get an idea of what your game will be.
- For simple videogames, a storyboard might be only one picture to represent the authors ' vision of the game.

# *Videogame parts*

- A videogame is divided into three (3) parts: opening, game engine, ending.
- Sometimes, a videogame has a story or a small animation to introduce the game.
- Sometimes, a game can show different endings.

# *Screens*

- A videogame project is divided into many screens: company logo(s), opening, title screen, menu, options, game screen(s), ending(s) and credits.
- The most important screens are: title screen, menu and game screen(s).

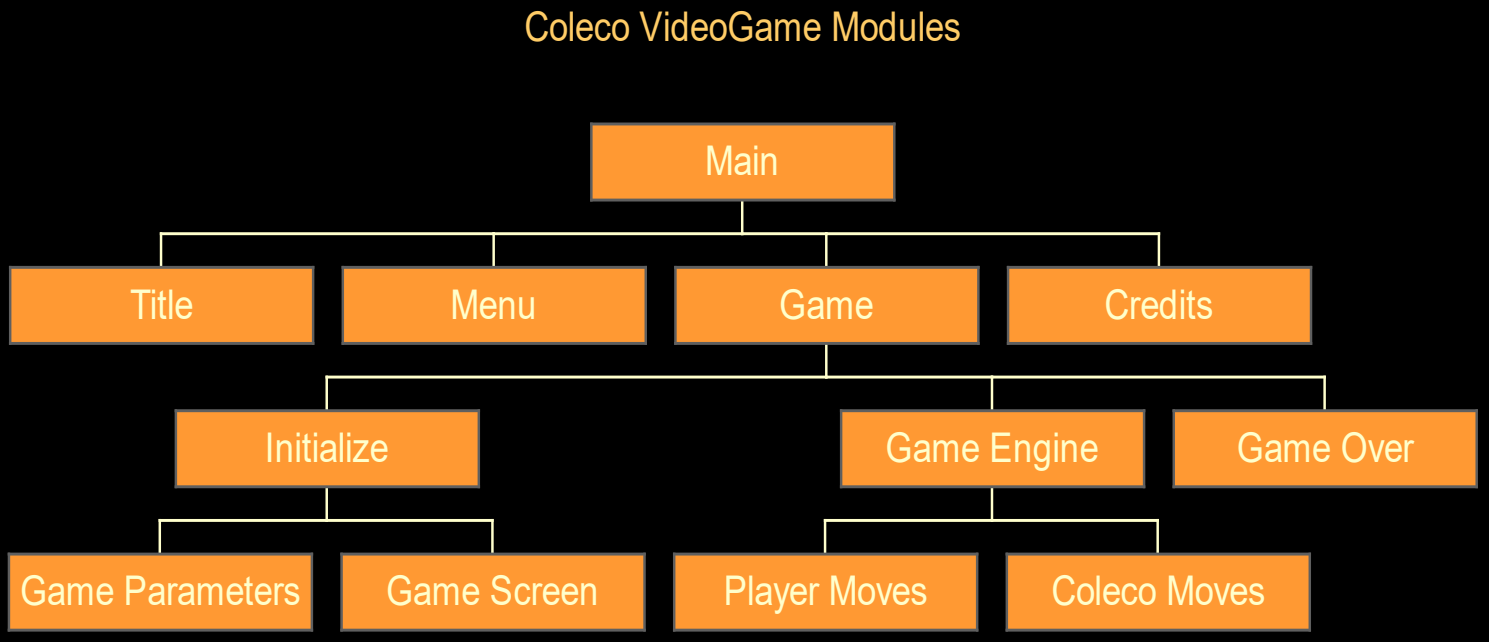
# *Important Screens*

- The title screen is the main entrance of the game: it should be professional.
- The menu screen is the best way to let the player modify the game parameters. It must be simple to use and undertstand.
- The game screens are what the gamer will use and remember most. To make a good impression, make them attractive.

# *Modules*

- A module is only a term I personally use to designate « a section of code » to do « a particular job » in the program.
- A module may need external data.
- A module can be in another C file.
- Generaly, a module is coded in a single routine.

# *Modules Sample Diagram*

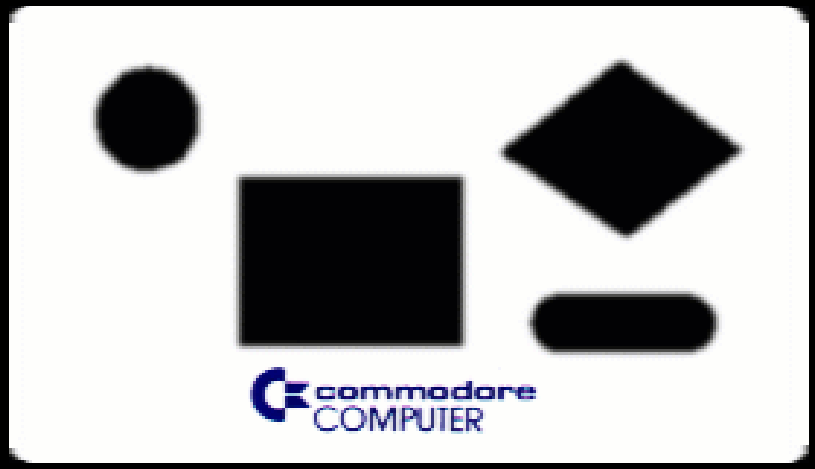


# *Flow Charts*

- A flow chart is a diagram which represents the normal execution of (a part of ) your code.
- This particular diagram uses a specific formalism with symbols like rectangles, rounded rectangles, circles and more.
- Using flow charts to represent the modules of your code is a good idea.

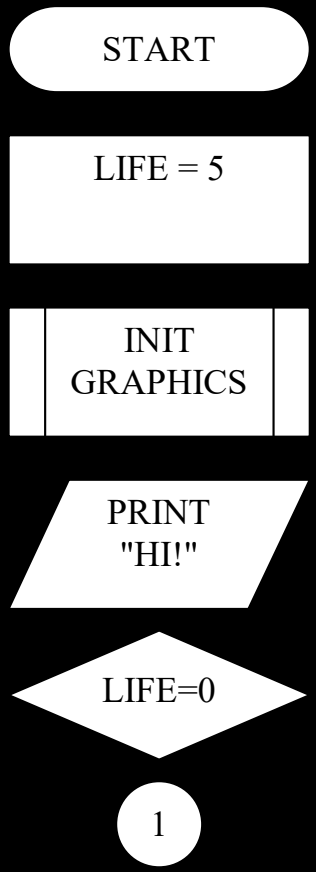
# *Flow Charts*

In the past, we used this kind of stencil to facilitate drawing flow charts on paper.




Now, with the computers, we use software to do exactly the same thing.

# *Flow Charts*




These symbols are the most common ones.

They are simple geometric graphics with text.

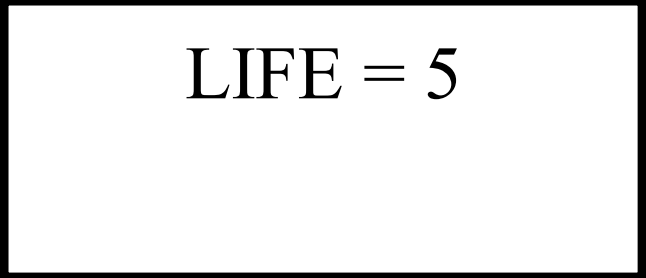
To see the execution path, the convention is to use arrows between symbols.

# *Flow Chart Symbols*

- This symbol is used to specify where your code starts and finishes.

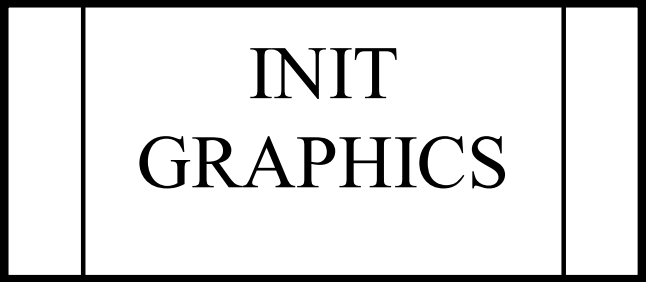
# *Flow Chart Symbols*

- This symbol is used to do a mathematical operation or to initialize a variable.



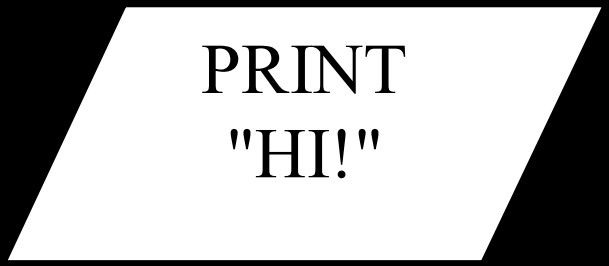
# *Flow Chart symbols*

- This symbol is used to avoid using too many symbols in your diagram. It can represent a module you call.



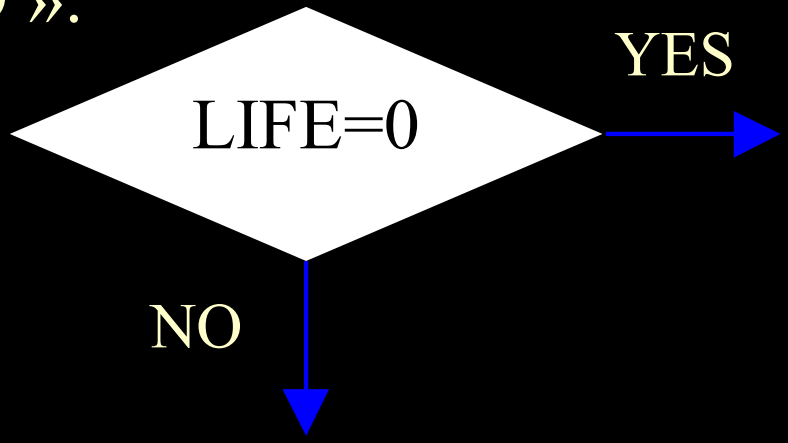
# *Flow Chart Symbols*

- This symbol is used for the output instructions. It may represent one or more lines of code in your program.



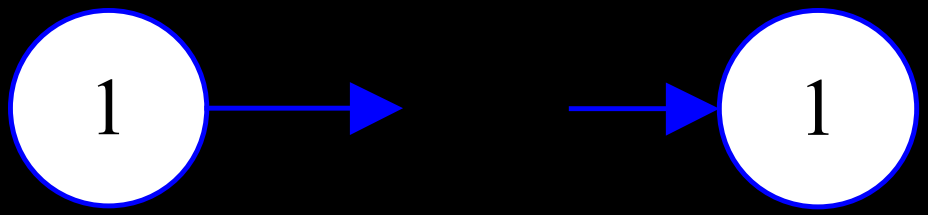
# *Flow Chart Symbols*

- This symbol is used for conditions. After this symbol, two arrows most be added: one for « YES », the other for « NO ».



# *Flow Chart Symbols*

- And this little symbol represents a label. You can use labels to mark where a jump is possible.




Label Go to Label

# *List of variables*

- Before doing a detailed flow chart, you must do a list of variables the game needs.
- Example: the variables for a bouncing ball on screen could be its coordinate and its orientation + speed (simply the incrementation in X and Y axes).
- A list of variables helps you to not omit important variables.


Between the idea and the code

# *What is an algorithm?*

- It ’s a representation of your code in a natural language.
- It uses keywords called « pseudo-code » to refer to the instructions you will need to write in the code.
- A flow chart is a graphical version of an algorithm.

# *Algorithm*

- An algorithm is a series of orders you give to the computer to be executed.
- To help your imagination, this is a sample algorithm to order a robot to buy apples at the market.

« Go outside. Turn Left. Walk to next corner. Turn left again. Walk 10 feet. Open door. Walk 3 feet. Buy apples. Come back. »

# *Pseudo-Code*

- There is no convention for pseudo-code but it must be something in natural language which represents instructions.

Ex.: « for each y position from 0 to 10 »

- When the time comes to write your code based on the algorithm, converting keywords to instructions is easy.

Ex.: « for (y=0;y<=10;y++) »

# *Programming in C*

General information about the
C programming language

# *Warning!*

- First of all, I need to specify that the C language I ’m talking here is not the C++ or C# language but the standard ANSI C.
- The following slides are only a very small introduction to the C programming. For more information, read books about C language.

# *C programming*

- A C file structure is simple if you understand this: what you need must be previously given. That is to say, no forward declarations
- At the top, you have the libraries, the constants, the headers of external routines, tables and global variables.
- At the bottom, you have routines and then routines who need others routines to run.

# *C programming*

```
#include<coleco.h>

#define potatoes 10

extern byte title[];

byte numbers[]={1,2,3};

byte numbers[3];
```

- Libraries are included in your C file with the instruction « #include »
- The constants are defined in your C file with the instruction « #define »

# *C programming*

```
#include<coleco.h>

#define potatoes 10

extern byte title[];

byte numbers[]={1,2,3};

byte numbers[3];
```

- The tables which are in another C file are included with the instruction « extern »

# *C programming*

```
#include<coleco.h>

#define potatoes 10

extern byte title[];

byte numbers[]={1,2,3};

byte numbers[3];
```

- The first array of bytes named ‘numbers’ here, is a typical example of a ROM table.
- The second array of bytes named ‘numbers’ here is a typical example of a RAM table.

# *Data types*

- char : character or signed short integer in one byte.
- byte : unsigned short integer in one byte.
- int : signed integer in two bytes.
- unsigned : unsigned integer in two bytes.
- [] : array
- char [] : array of char or string

# *Operators*

- Arithmetic operators: +, -, *, /, % (modulus)
- Increment ++ and Decrement --
- Bitewise operators: << (shift left), >> (shift right), & (and), | (or), ^ (x-or), ! (not)
- Combined operators: <variable> = <variable><operator><expression> become <variable><operator>=<expression>. Ex.: a=a+2 become a+=2

# *Operators*

- Relational operators: > (greater than), < (less than), == (equal), >= (greater than or equal), <= (less than or equal), != (not equal)
- Logical operators: && (and), || (or), ! (not)

# *If Statement*

- Simple if

```
if (value) statement1;
```

- if... else

```
if (value) statement1;
else statement2;
```

- A statement can be inside brakets: { and }

# *Loops*

There are many kinds of loops.

- for loop

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

- while loop

```
while (x<10) {}
```

- do... while loop

```
do {} while (x<10)
```

# *Functions*

- A function may return a value or not (void).
- A function may need parameters to execute.
- A function header syntax is:

<return type> function_name (<data type> parameter#1, <data type> parameter#2, ...)

int substract (int a, int b)

void main(void)

# *Functions*

- A function core is written in braces: { and }
- At the top we have the temporary variables used within the scope of a particular function.
- After the temporary variables, we add instructions (operations and command lines)
- When needed, we use braces again to group many instructions together.

# *Function Sample*

```
byte sum(byte l)
{
 byte j;
 byte k;

 k=0;
 for(j=1;j<=l;j++)
 {
   k += j;
 }
 return k;
}
```

- This routine needs a byte value and returns a byte value.
- The temporary variables j and k are declared at the top of the routine.
- And finally, the operations.

# *I know…*

It ’s not enough information to start programming in C. You ’ll have to read books to learn C language.

Now, let ’s talk about Coleco programming

# *Coleco programming in C*

What you need to do a new Coleco project:

- the coleco library
- an nmi routine (empty or not)
- a main routine
- and a lot of routines for your own purposes
- You can use more libraries
- You can use more than one C file

# *Coleco programming in C*

The execution steps must be something like:

- Initialize the video display (VDP registers and Video memory)
- Initialize global variables
- Start doing operations and commands
- Use a loop for the game engine
- Use variables to output things on screen
- Use a delay to slowdown the execution

# *Libraries*

The Coleco library by Marcel de Kogel is the SDK to program new Coleco projects in C.

Getput1 library is a toolbox with a lot of useful routines. This library uses coleco library routines.

But, because we have no time, I suggest to read the Coleco programming documentation to find more information about these libraries.

# *Compile and Test*

How to use CCI software in the Coleco development kit?

# *Step by step*

- Create a project directory as a sub-directory of the compiler.
- Save all your C files in your project directory.
- Copy CCI software in your project directory.
- Run CCI to compile and link your project.

# *Example*

We will do now a new Coleco project together.

Follow the instructions in the next slides to experience the joy of compiling a new Coleco project.

I hope you have un-archived the Coleco development kit on your PC.

Open Notepad. Good luck!

# *Sample code: « Hello World »*

- This code is the starting point for all new ColecoVision projects in C.
- #include <coleco.h> is the command to include the Coleco library by Marcel de Kogel in your project.

```
#include <coleco.h>

/* EMPTY NMI ROUTINE */


/* EMPTY MAIN ROUTINE*/
```

The nmi routine is needed to use the Coleco library… even if you don't use it.

The main routine is the starting point of the program.

# *Sample code: « Hello World »*

```
#include <coleco.h>
#include <getput1.h>

/* EMPTY NMI ROUTINE */


/* EMPTY MAIN ROUTINE*/
```

- Getput1 library is a toolkit with a lot of graphic routines and more.
- This library helps you to simplify code by grouping commonly used functions.
- You must use it.

# *Sample code: « Hello World »*

```
#include <coleco.h>
#include <getput1.h>

/* EMPTY NMI ROUTINE */


/* MAIN ROUTINE*/
```

- You need to initialize the VDP registers to setup the screen.
- Fortunately, Getput1 helps us. This library contains a routine named screen_mode_2_text.

# *Sample code: « Hello World »*

```
#include <coleco.h>
#include <getput1.h>

/* EMPTY NMI ROUTINE */


/* MAIN ROUTINE*/
```

- Now, you need to initialize the characters set in video memory.
- Again, the Getput1 library helps with another routine named upload_default_ascii.

# *Sample code: « Hello World »*

```
#include <coleco.h>
#include <getput1.h>

/* EMPTY NMI ROUTINE */


/* MAIN ROUTINE*/
```

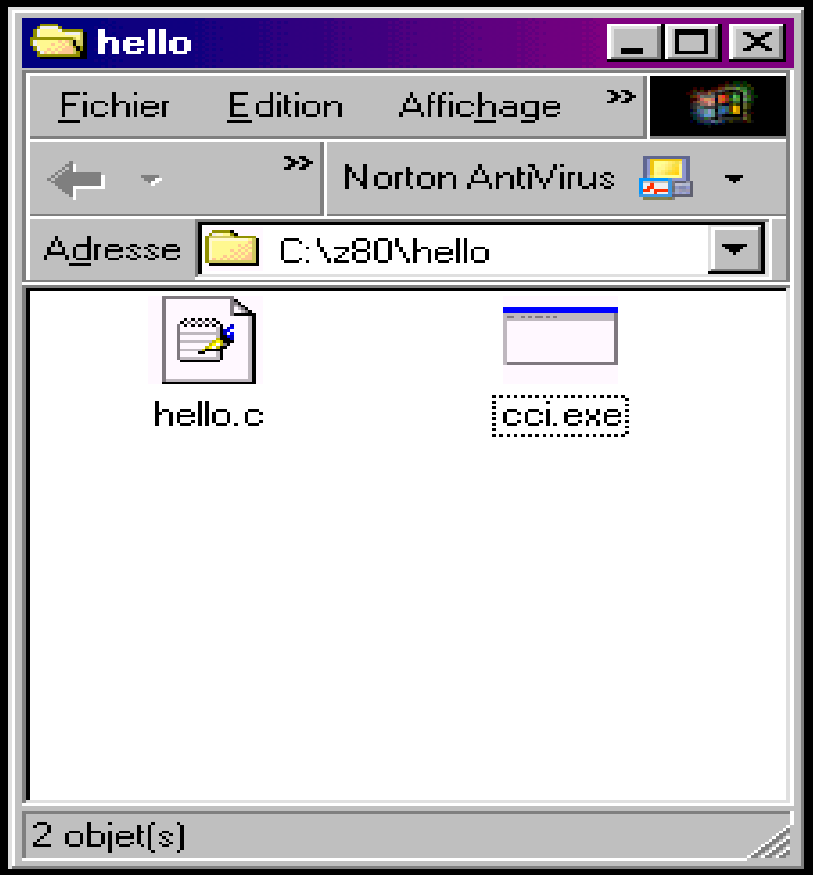
- Now, you are ready to print on screen.
- Use cls to clear screen
- Use center_string to center a string on screen.
- Use print_at to print on screen at a particular coordinate.

# *Sample code: « Hello World »*

```
#include <coleco.h>
#include <getput1.h>

/* EMPTY NMI ROUTINE */



/* MAIN ROUTINE*/



infiniteloop:
        infiniteloop
```

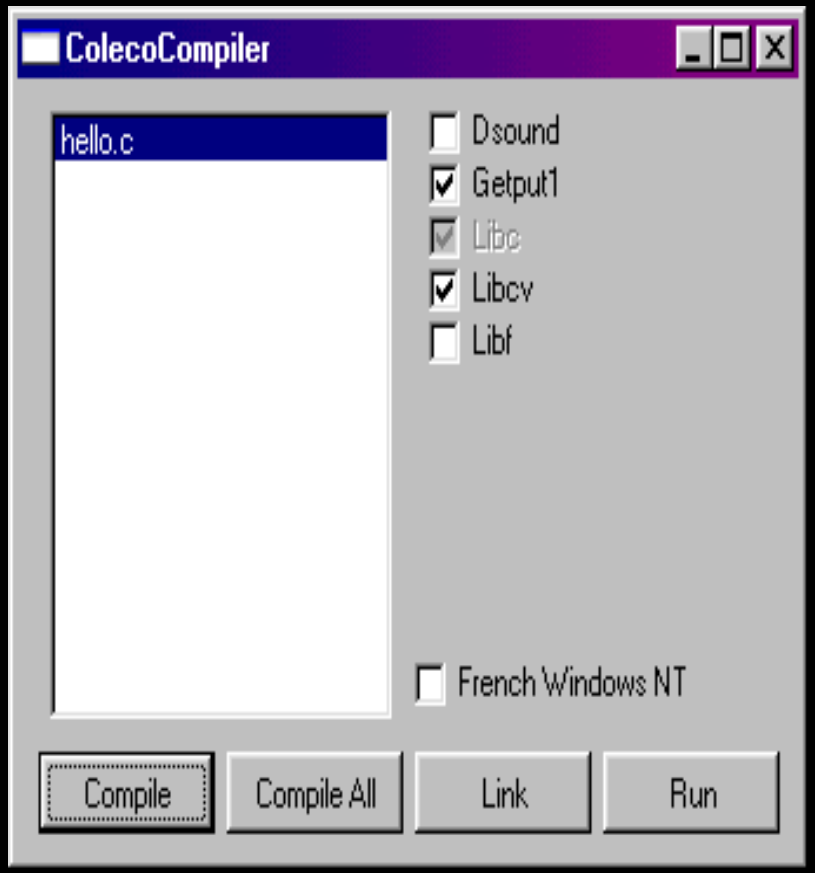
- Finaly, you must suspend normal execution to let the user see the messages.
- Use delay to suspend normal execution for a specified amount of time.
- Use pause to wait for a fire button.
- Use an infinite loop

# *Sample code: « Hello World »*



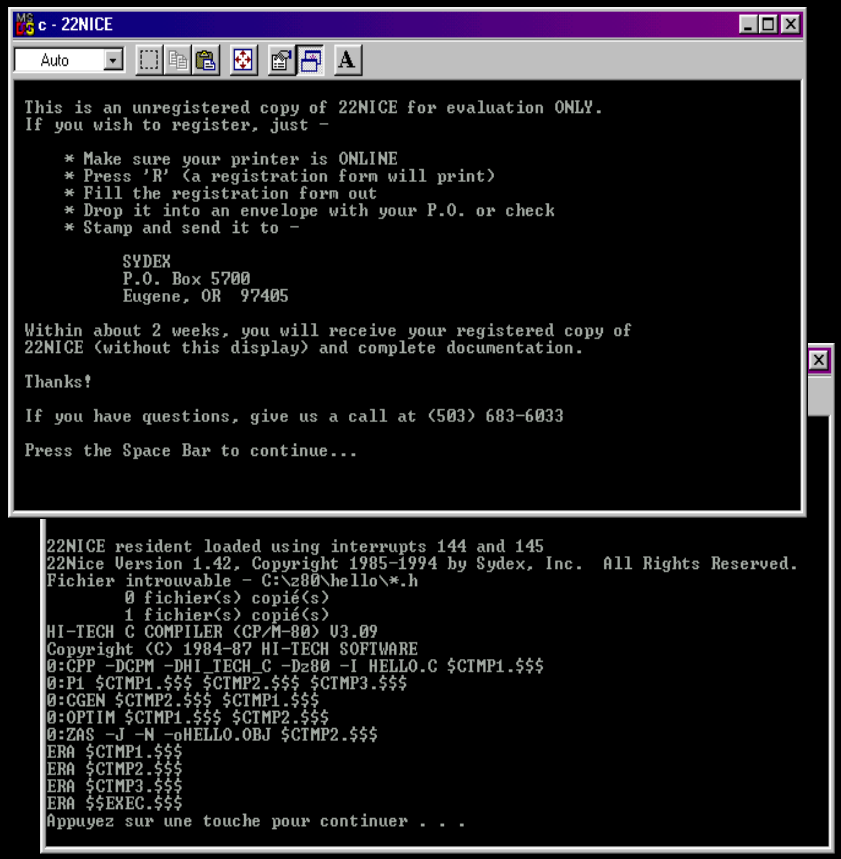

- Create a sub-directory in the compiler directory. This new directory is your project directory.
- Save your code as a C file in your project directory and add the CCI software.

# *Sample code: « Hello World »*



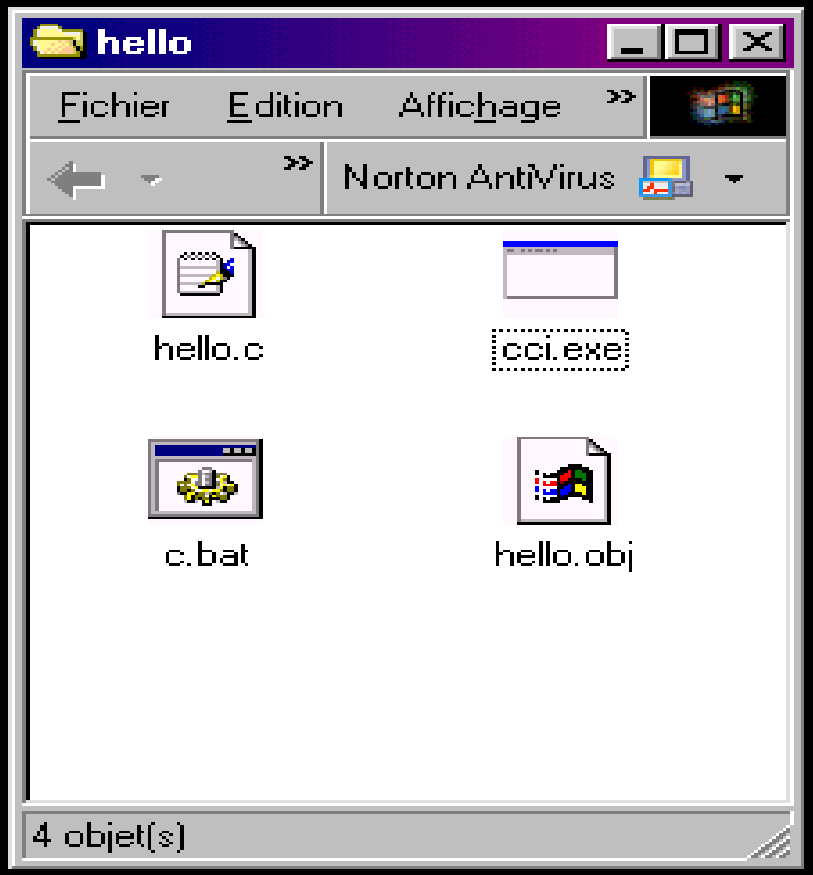

- Check Getput1 checkbox to let CCI use this library.
- Select your C file in the filelist box and click on Compile.

# *Sample code: « Hello World »*



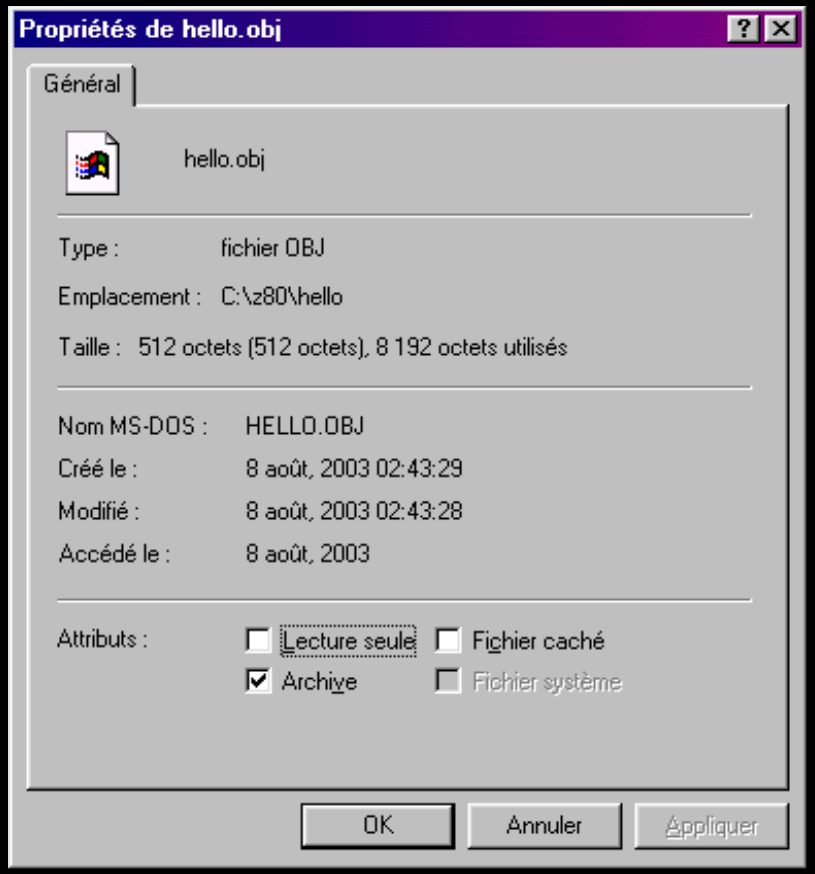

- A popup DOS window appear with 22NICE running.
- After pressing space bar, the compiler will start.
- After compiling your code, the DOS window waits. Close this window.

# *Sample code: « Hello World »*



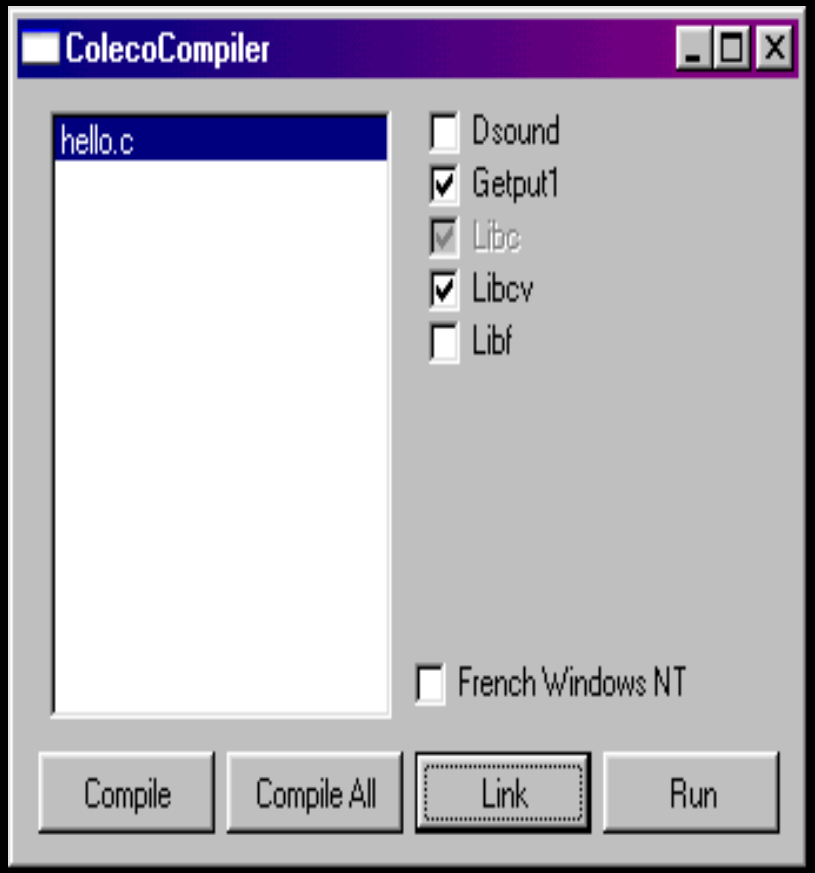

- CCI created a batch file named « c.bat »
- The compiler generated an object file named « hello.obj »

# *Sample code: « Hello World »*



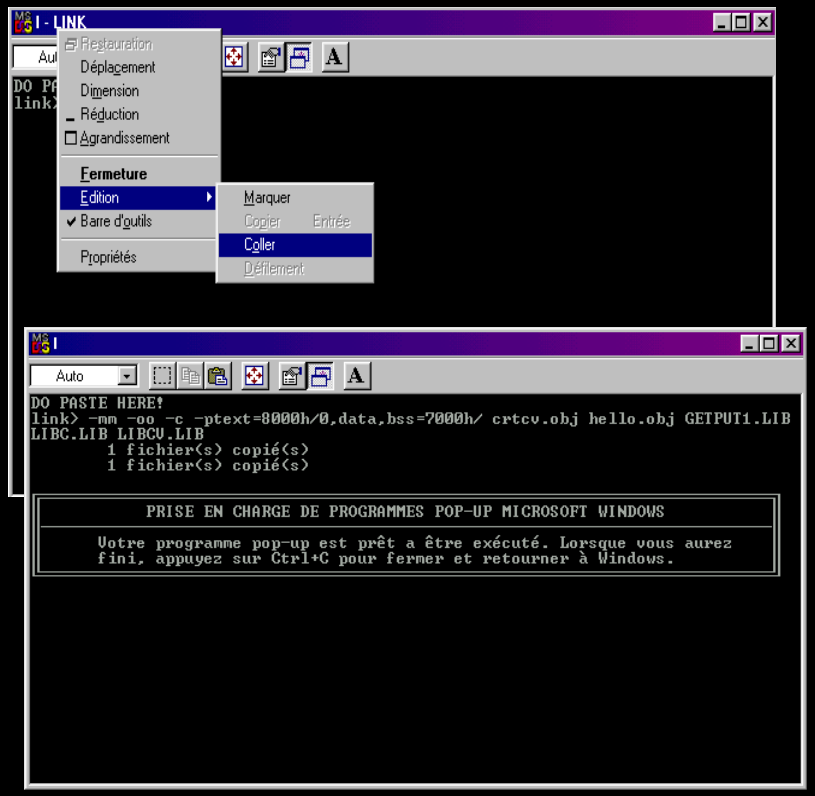

- Before using the linker, you must verify the filesize of the objets files.
- 512 bytes… it ’s very small but it ’s not zero.

# *Sample code: « Hello World »*




- Now, it ’s time to use the linker.
- Click on the Link button of the CCI software.

# *Sample code: « Hello World »*

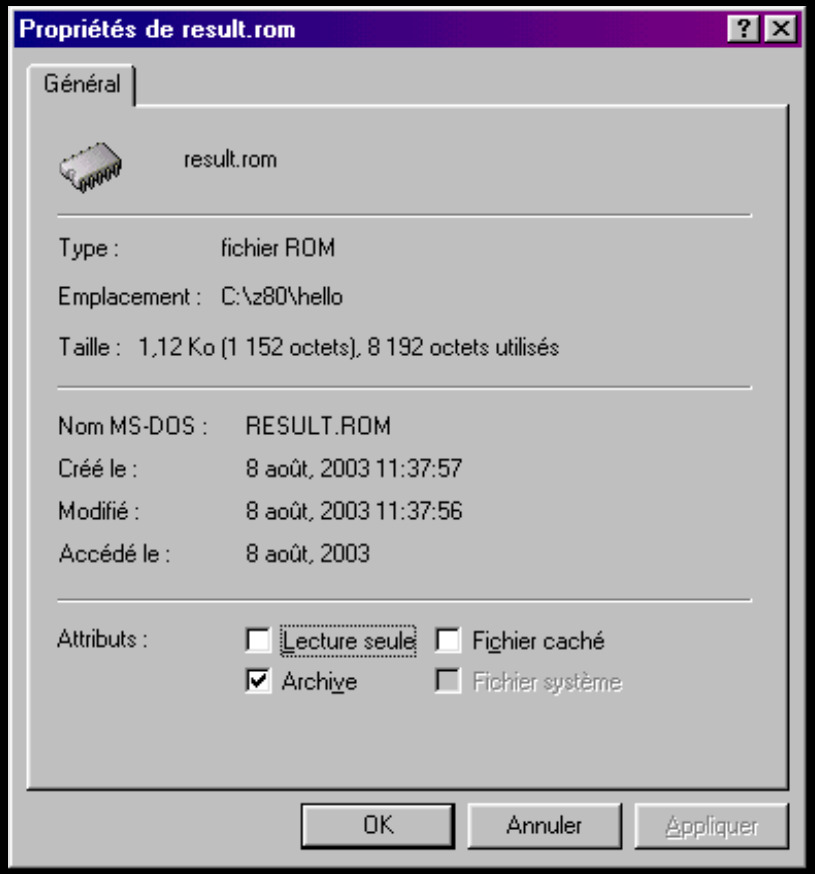



- 22NICE starts again.
- The linker is waiting for instructions.
- CCI will have copied the instructions to the clipboard.
- Do a « paste » in the DOS window.

# *Sample code: « Hello World »*


- CCI created a batch file named « l.bat »
- The linker generated two files: « map.txt » with the memory map information, and « result.rom » which is the rom file we wanted to create.

# *Sample code: « Hello World »*




- If Windows closed your DOS window, you may not see if the linker was able to link right the object files together with the libraries.
- To be sure, check the filesize of the rom file.

# *Sample code: « Hello World »*


- If you didn ’t close the DOS windows, do it now.
- Click on the « Run » button to start VirtualColeco.
- Well, it looks like you did it! :)

# *It 's working! :)*

- Now, you know how to use CCI. It 's time for you to program a bigger project.
- Note: A good way to do a project is to compile and link your code each time you update it.


Be prepared for the next:
New ColecoVision Programming
presentation