MPAGD MSX suite (only MSX1 specs supported)

README.md

MSX AGD Suite

AGD suite for converting, compiling and running AGD games on an MSX machine. Right now, the generated games will work in all MSX machine types; but only MSX1 features are supported (except color palettes to some extent).

Installation

Simply copy the files into a directory and make sure you have an environment variable (MSXEMUPATH) pointing to the home directory of your OpenMSX installation. You can of course use any other MSX emulator but you'll be forced to tune-up the actual build script to use their own command line parameters. As stated below, some specific models have been configured in the build script and thus, the exisiting OpenMSX installation must support them; you're free to change them to your favourite OpenMSX configured setups.

ZX Spectrum AGD programs conversion

The use of **MPAGD suite v0.7.5** (https://jonathan-cauldwell.itch.io/multi-platform-arcade-game-designer) is recommended. There's also a convert tool in this suite but; right now it's not in a working status or supported.

Building AGD programs

Usage: build <AGD file> [-?|-h] [-a] [-t <SC2 file>] [-q <SC2 file>] [-c <KB Size>|-d <KB size>] [-x <MSX type>]

Where:

<AGD file> AGD source file without .agd extension (1)

-?|-h This help -a Enables adventure mode -t <SC2 file> Loads a title screen -q <SC2 file> Loads a marquee screen to be used by te game (no video initialization at boot) -d <KB size> Disk (RAM) distribution: Valid KB sizes are: 16,32,48 (default:32) -c <KB size> Cartridge (ROM) distribution: Valid KB sizes are: 16,32 (default:32) -x <MSX type> Launch emulation after successful build

```
MSX types (-x)
------------------
0: None
1: MSX1 (Sony HB-75P) (Default)
2: MSX2 (Philips NMS-8245)
3: MSX2+ (Panasonic FS-A1WX)
4: TurboR (Panasonic FS-A1GT)
```

(1) Default are disk (DSK) distribution, 32KB setup and all the rest of options disabled except emulation set for MSX1

Examples:

```
>build oceano
```

Builds oceano.agd, creates a DSK and launches a MSX1 emulation.

```
>build oceano -c
```

Builds oceano.agd, creates a ROM and launches a MSX1 emulation.

```
>build foggy -a -x 2
```

Builds foggy.agd, enables adventure mode, creates a DSK and launches a MSX2 emulation.

```
>build dcr11 -t TITLE.SC2 -x 4
```

Builds dcr11.agd, adds a loading screen, creates a DSK and launches a TurboR emulation).

The distribution files (DSK or ROM files) are usually located in the OpenMSX home directory if any has been found, if not, the home directory of the AGD suite will be used instead.

Differences between ZX & MSX compilers

The AGD language as MPAGD v0.7.3 defined it is fully supported but to benefit from MSX features some changes have been made to the language syntax:

```
DEFINEBLOCK
```

Now 16 values are needed (excluding the block type) to define the shape of a block: first 8 values are the pattern, the next 8 are color attribute values, one for each row.

```
DEFINESPRITE
```

No changes to the amount of bytes used; but the arrangement is different to the one found in the ZX. The first value is the number of sprite frames; about the patterns, if you think in a 16x16 sprite divided in 8x8 areas:

AB CD

The values must be defined following order ACBD, first the 8 bytes from area A, next the 8 bytes from C and so on for each sprite and sprite frame...

```
DEFINEOBJECT
```

They're not sprites now!. In the MSX engine, objects are like block characters but with 16x16 size and can only be placed in coordinates multiple of 8 (they'll shown garbage if not). This is reflected in the attribute bytes that they're only 3 now: room, Y coordinate, X coordinate, they loose the ZX color attribute:

DEFINEOBJECT <room> <y> <x>

Given that, the bytes used are 64 and following the above sprite example they're ordered as ABCD, the first 32 bytes being the pattern bytes and the next 32 bytes the color attributes of each 8x8 area.

```
SPRITEPOSITION
```

One attribute is added: color of the sprite. The syntax is:

SPRITEPOSITION <type> <image> <color> <y> <x>

```
COLOUR
```

The only parameter accepted is used to define both foreground and background color. You can use the formula FG*256+BG to calculate the value or use the more straightforward hexadecimal numbers (as suggested in the following section).

```
SPRITEINK, INK, PAPER & BORDER
```

Now the value can be 0..15. Remember that 0 is transparent!.

```
DEFINEPALETTE
```

If the MSX machine supports color palettes this feature is enabled. The default palette if you don't use the DEFINEPALETTE command is the following:

0x000,0x000,0x611,0x733,0x117,0x327,0x151,0x627 0x171,0x373,0x661,0x664,0x411,0x265,0x555,0x777

The 16 values (one color palette definition each) are defined using the GRB scheme.

An example of use is the following:

DEFINEPALETTE $000 $000 $611 $733 $117 $327 $151 $627 $171 $373 $661 $664 $411 $265 $555 $777

```
PALETTE
```

As the above function, it'll only work in MSX2 or higher machines (as a matter of fact, a MSX1 machine with a V9938 can also work). It's used as follows:

PALETTE <color number> <color byte definition 1> <color definition byte 2>

Where:

color byte definition 1: 0 R2 R1 R0 0 B2 B1 B0 (defines red and blue color values). color byte definition 2: 0 0 0 0 0 G2 G1 G0 (defines green value).

```
DEFINEMUSIC
```

The MSX AGD parser has an integrated music & sfx framework based in the well-known PT3 song format and ayFX sounds. You can create or use a existing PT3 file in your program doing the following:

1. Prepare the PT3 archive trimming their first 100 bytes. You can accomplish this with the provided "trim" tool:

   trim myfavesong.pt3 100

2. Include the resulting file using DEFINEMUSIC command:

   DEFINEMUSIC "myfavesong.pt3"

You can define until 10 different musics or jingles, numbered from 1 to 10 in the same order they're being added.

```
MUSIC
```

Once included in your program with the define command, you can invoke the replayer using MUSIC command:

MUSIC <0|1-10> <0|1>

The first argument plays the nth defined music from 1 to 10. Value 0 stops the playing. The second argument mean if you want to endlessly loop the music (1) or play it only once (0), eg.: a "Game Over" jingle.

```
SOUND
```

Works the same as the already existing in AGD. It's used as follows:

SOUND <sound number>

The sound number represents the sound position inside an ayFX soundbank minus one; so if the bank has sounds 1-5, you'll use number from 1 to 5 to play them. Number 0 mutes the sfx, it's an alias of SILENCE command.

Only one soundbank is used, named "sfx.afb" that must be located in "AGDsources" directory. As for now, this can't be changed. This ready-made sound bank has the following sounds:

1. Jumping
2. Explosion
3. Shoot
4. Pickup
5. Event
6. Laser
7. Jet

ayFXEdit (https://shiru.untergrund.net/software.shtml) can be used to edit and create these banks (including the default one, of course). It's recommended to save it without sound names, you'll save some useless bytes.

```
DEFINECONTROLS
```

Now they're the following as default (note that "default" means no DEFINECONTROLS is added to the code):

- Movement: Cursor keys
- Fire 1: Space
- Fire 2: "M"
- Fire 3: "N"
- Option 1: "1"
- Option 2: "2"
- Option 3: "3"
- Option 4: "4"

When a joystick is used (1 or 2), both fire buttons are used and the third button is still mapped to the keyboard (joymega is still not supported).

```
SPRITESOFF
```

Hides all sprites, no parameters are used. They'll remain hidden until you set again their coordinates (eg.: entering a new screen or manually).

```
STOP key
```

It can't be programatically controlled; but it always works pausing the game (music still plays) until a new STOP key press releases the pause.

TIPS & BITS

Color conversion

When converting a ZX AGD source file, start looking for attribute color uses. You must adapt them to MSX color scheme.

Remember that a ZX color attribute byte always comes as follows:

```
7 6  5  4  3  2  1  0
F B P2 P2 P1 I0 I1 I0
```

- Bit 7 if set indicates the colour flashes between the foreground (ink) and background (paper) colours.
- Bit 6 if set indicates the colours are rendered bright.
- Bits 5 to 3 contain the paper (background) colour 0..7
- Bits 2 to 0 contain the ink (foreground) colour 0..7

The MSX one must be represented as follows:

```
 7  6  5  4  3  2  1  0
I3 I2 I1 I0 P3 P2 P1 P0
```

- Bits 0 to 3 contain the paper (background) colour 0..15
- Bits 4 to 7 contain the ink (foreground) colour 0..15

Hexadecimal numbers are supported so it's usually the easiest way to represent a color attribute, eg.: COLOUR $F1 (ink 15 & paper 1)

Engine FPS

The FPS (frames per second not first person shooter :-P) of the MSX engine double the number of the generated by the ZX engine (50fps against 25fps); so you must be extremely careful with variables used as timers (to animate sprites or timed actions) that are increment every "loop" or frame. You can usually find them in events MAINLOOP1 and MAINLOOP2 and once identified you must probably double it's values. Of course you need also to look for them in the rest of the source (usually as "IF variable = value" expressions) and evaluate if they need to be adapted. As the ZX engine moved the sprites 2px at a time, now they're only moved 1px every executed move command so the action is smoother. As an undesired effect, now it's hard to place a sprite in position (eg.: a main character under a stair).

Sprite colors

SPRITEINK commands are not needed...or mostly not. Sprites now have a color attribute defined in SPRITEPOSITION commands so they don't need to dinamically being assigned. They could be useful if you want to change sprite colors during the game or color a newly created sprite with SPAWN.

Text colors

Use COLOUR instead INK & PAPER when possible. It's faster and uses less RAM.

Locating cursor

Use AT command instead LINE & COLUMN. It's faster and uses less RAM. After a printing command has been executed, the COLUMN is always updated but the LINE remains the same (if you haven't reached the rightmost edge of the screen) so if you want to print more texts think if it's enough to update only a variable, LINE or COLUMN, this way you can keep saving more bytes and gaining again a little speed.

Beware MAINLOOPs

AVOID use of printing commands in MAINLOOP events (eg.: scoreboards). Of course yo must need to print something but look for alternatives or ways to reduce its use to a minimum, don't keep printing every frame the same text over and over again. The MSX color printing of texts is heavy.

Waiting for a key

WAITKEY right now doesn't returns the key pressed.

TICKER routine workaround

TICKER routine only updates every frame when a loop is completed. If you're in an infinite loop (eg.: in a WHILE/ENDWHILE conditional loop) and you want to see the TICKER being updated, you'll need to invoke a DELAY inside the loop. Use "DELAY 1" and it's enough...

TICKER with colors

TICKER routine doesn't update at all the color table; but you can setup the scrolling area before the ticker routine starts. Print anything you want with the same size of the scroll area and the desired colors, the scroll will be automagically colored!. Use blocks and you can even get some nice multicolor scrolling.

MSX Turbo CPUs

The MSX turbo enabled machines, some MSX2+ and MSX2 machines, automatically activate this feature. The TurboR machines also use the R800 ROM mode by default and thus are the fastest machines (all the events code is processed much faster), this can have influence over the pitch of the beeper commands.

MSX color palettes

In MSX2 and higher machines, the palette routine works so you can define your own color palette.

Sprite planes priorities

The order used when defining the screen sprites also determines the priority plane it uses. First SPRITEPOSITIONS are placed in higher priority planes than the last ones, this can change if you use REMOVE & SPAWN commands.

Particles

Try to not to abuse also of particles, several particle sources can easily overhead the frame process time causing slowdowns to some extent. The dynamic v-syncing will help to minimize this but definitely, use them wisely.

Extra tools

The MSX suite comes with some extra tools I've added:

- FontConverter; Java executable that allows to convert a 64c font file (Commodore 64) to a chr AGD loadable font.
- ZXAttributeConverter: Simple tool to convert a ZX color attribute value to its MSX1 equivalent.
- trim: Script to trim ending bytes of a file. Already explained above.

Acknowledgements

Sjasm - XL2S Entertainment (http://www.xl2s.tk/) ONEDRIVE v1.0 - Adriano Camargo Rodrigues da Cunha (http://www.alsoftware.com.br/adrianpage/), sadly his page is not available anymore...

wrdsk v1.6 - Arnold Metselaar (http://www.math.utwente.nl/~metselaa/msx/diskutil.html). Also not available but you can find a mirror here: (https://github.com/clach04/msx_diskutil)

split - from CoreUtils for Windows suite (http://gnuwin32.sourceforge.net/packages/coreutils.htm)