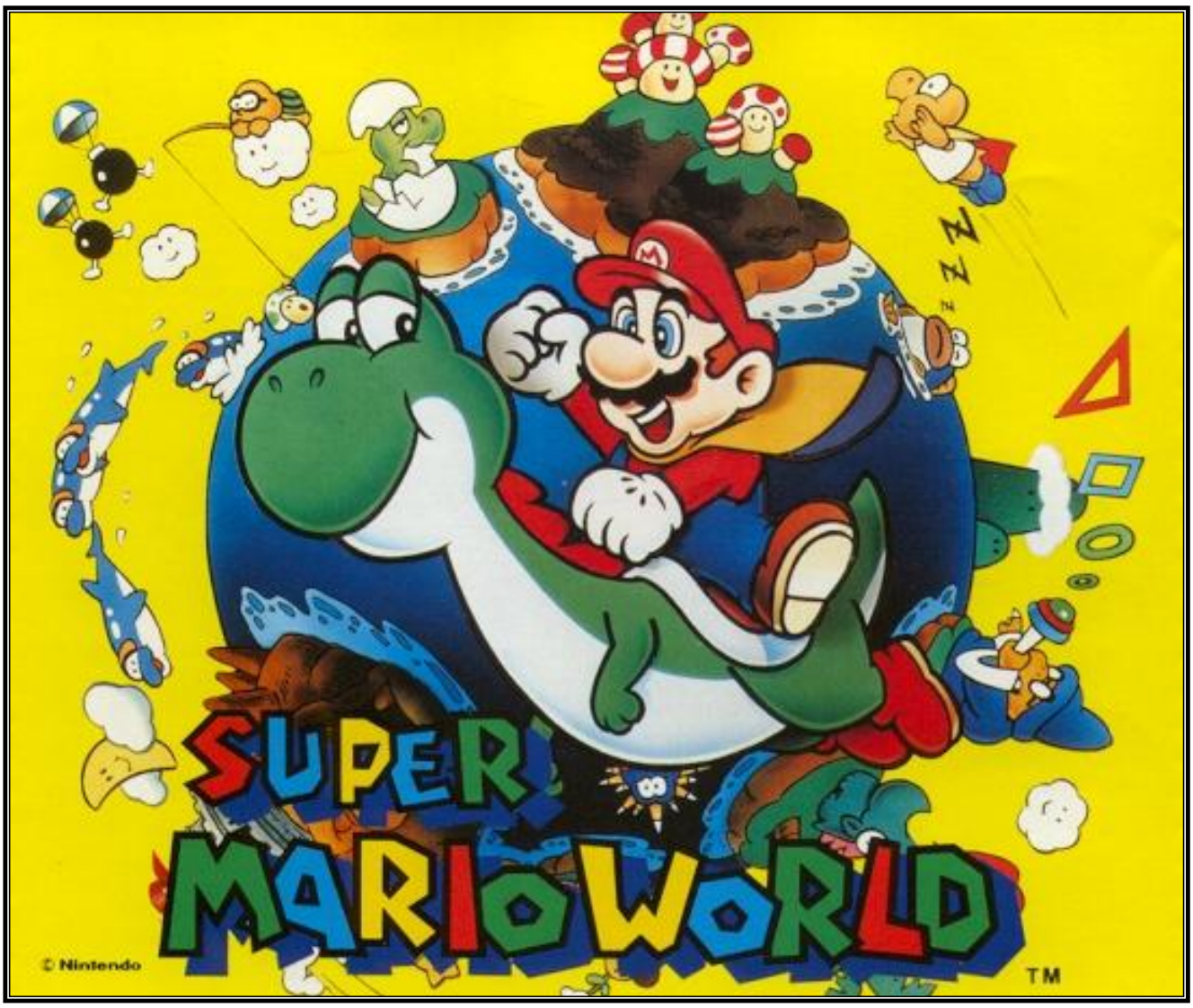
spe

2016

Fortaleza Reznor

anonimzwx



DYNAMIC Z TUTORIAL

Complete documentation of Dynamic Z V 3.0

Table of Contents

I. Introduction 1

II. Installation 2

III. Graphics Change 3

IV. Palette Change 6

V. Player Graphics and Palette Change 8

1. Graphic Change 8

2. Palette Change 10

VI. Sprites 12

1. Sprite Creation: General Stuffs. 12

Animation Tables 12

Frames Tables 14

How to use Tables and RAMs 18

2. Dynamic Sprites 20

How to make a Dynamic Sprite 21

Dynamic Sprite of 32x32 23

Dynamic Sprite of 48x48 25

Dynamic Sprite of 64x64 27

3. 50% More Mode 28

Dynamic Sprite of 80x80 29

4. Giant Dynamic Sprites 31

Giant Dynamic Sprites of 96x96 31

Giant Dynamic Sprites of 112x112 32
5. Semi-Dynamic Sprites 34
VII. Custom Player 36
VIII. Routines 40
IX. Compatibility 42
X. Other Content 43
XI. Credits 44

I. Introduction

Dynamic Z is a patch used to control SNES VRAM (Video RAM) and CGRAM (Color Graphic RAM) easily, with this you can make things like change graphics and palettes on the fly, use dynamic sprites and make changes on the player like power ups or your own player.

In this tutorial you will find how to use and create the different resources that this patch allows.

II. Installation

To install Dynamic Z you must follow the next steps:

1. Put DynamicZ.asm and header.asm on the same folder.
2. Open header.asm.
3. Go this line:

```
!Freespace = $3E8000
```

4. Use Slogger on the ROM to know a Free Space.
5. Replace the line of step 2 for the Free Space.
6. Insert the patch using Asar.

To use Dynamic Z on dynamic sprites follow the next steps:

1. After installation put a copy of header.asm in Sprite folder of sprite tool.

To use Dynamic Z on uber asm or other patches follow the next steps:

1. After installation put a copy of header.asm on the same folder of the uber asm or the patch.
2. Go to the top of the uber asm or the patch after:

```
header : lorom
```

3. Put the next line:

```
incsrc header.asm
```

III. Graphics Change

To change graphics you need the following RAMs:

- !GFXNumber (1 byte): Number of Graphic Changes that the patch is trying to make. Max Value = #$0A
- !GFXBnk (20 bytes): Bank of the resource.
- !GFXRec (20 bytes): RAM address of the resource.
- !GFXLenght (20 bytes): Size of the resource. Max Value = #$0800
    - 0x800 = half of GFX.
    - 0x400 = quarter of GFX.
    - 0x20 = A tile of 8x8.(0x40=2 tiles, 0x60=3 tiles, 0x80 = 4 tiles, etc)
- !GFXVram (20 bytes): Place on the VRAM where the resource will arrive.
    - #$0000 = First half of FG1.
    - #$0400 = Second half of FG1.
    - #$0800 = First half of FG2.
    - #$0C00 = Second half of FG2.
    - #$1000 = First half of BG1.
    - #$1400 = Second half of BG1.
    - #$1800 = First half of FG3.
    - #$1C00 = Second half of FG3.
    - #$2000 = First half of BG2.
    - #$2400 = Second half of BG2.
    - #$2800 = First half of BG3.
    - #$2C00 = Second half of BG3.
    - #$6000 = First half of SP1.
    - #$6400 = Second half of SP1.
    - #$6800 = First half of SP2.
    - #$6C00 = Second half of SP2.
    - #$7000 = First half of SP3.
    - #$7400 = Second half of SP3.
    - #$7800 = First half of SP4.
    - #$7C00 = Second half of SP4.

The Dynamic Z will try to make the maximum number of graphics changes per frame up to transfer 0x800 bytes.

To make a Graphic Change you must follow the steps:

- If !GFXNumber is #$0A or more then return.
- Put the double of the value of !GFXNumber on X register.
- Increase !GFXNumber in 1.
- Fill !GFXBnk,x, !GFXRec,x, !GFXLenght,x and !GFXLenght,x.
- Return.

For example I will transfer half of GFX to the first half of BG3:

```
GraphicChange:

        LDA !GFXNumber
        CMP #$0A
        BCC +
        RTS
+
        INC A
        STA !GFXNumber
        DEC A
        ASL
        TAX

        PHB
        PLA
        STA !GFXBnk,x
        LDA #$00
        STA !GFXBnk+$01,x

        REP #$20
        LDA GFXPointer
        STA !GFXRec,x

        LDA #$0800
        STA !GFXLenght,x

        LDA #$2800
        STA !GFXVram,x

        SEP #$20
        RTS

GFXPointer:
dw resource

resource:
incbin AnyGFXOf2kb.bin ;replace this for your GFX name
```

Another way is to insert your resource with a patch like this:

```
!FreeSpace = $308000

org !FreeSpace

db "ST","AR"
dw End-Start-$01
dw End-Start-$01^$FFFF

Start:
incbin AnyGFXOf2kb.bin ;replace this for your GFX name
End:
```

In this case the bank will be the first 2 digits on the FreeSpace and the resource will be the last 4 digits + 8.

```
GraphicChange:

        LDA !GFXNumber
        CMP #$0A
        BCC +
        RTS
+
        INC A
        STA !GFXNumber
        DEC A
        ASL
        TAX

        REP #$20
        LDA #$0030
        STA !GFXBnk,x

        LDA #$8008
        STA !GFXRec,x

        LDA #$0800
        STA !GFXLenght,x

        LDA #$2800
        STA !GFXVram,x

        SEP #$20
        RTS
```

If you are using a sprite to change palettes, remember to use PHX before the routine and PLX after routine.

IV. Palette Change

To change palettes you need the following RAMs:

- !paletteNumber (1 byte): How many colors will you change?
- !paletteDestiny (128 bytes): What color will you change?
- !paletteLow (128 bytes): Low byte of the new color.
- !paletteHigh (128 bytes): High byte of the new color.

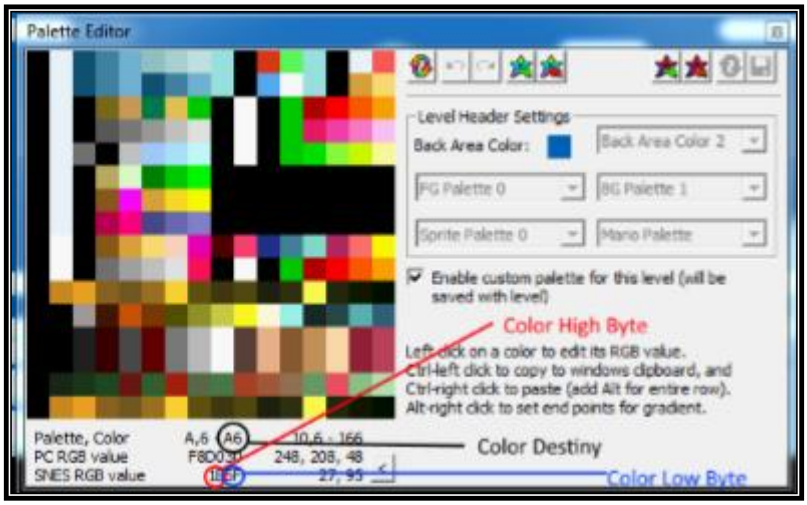
To change colors you should follow the next steps:

- If !paletteNumber + All new colors that you want to change are more than 80 then return.
- Add to !paletteNumber the number of colors that you want to change.
- Make a loop that start on !paletteNumber – 1 and end at 0 to fill !paletteDestiny, !paletteLow and !paletteHigh.

In code it is like this:

```
ColorDestiny:       db $xx,$xx,$xx,$xx,$xx,$xx… ;replace this table
ColorLowByte:       db $xx,$xx,$xx,$xx,$xx,$xx… ;replace this table
ColorHihByte:       db $xx,$xx,$xx,$xx,$xx,$xx… ;replace this table
PaletteChange:

       LDA !paletteNumber
       STA $00
       CLC
       ADC #$numberOfColors-1 ;putthe number of colors that you will change-1
       CMP #$80
       BCC +
       RTS
+
       STA !paletteNumber
       DEC A
       TAX
       LDY #$numberOfColors-1 ;putthe number of colors that you will change-1
.loop
       LDA ColorDestiny,y
       STA !paletteDestiny,x

       LDA ColorLowByte,y
       STA !paletteLow,x

       LDA ColorHihByte,y
       STA !paletteHigh,x

       DEY
       DEX
       BMI +
       CPX $00
       BCS .loop
+
       RTS
```

The destination, low byte and high byte you can obtain them on lunar magic. You only need to open the palette window and copy the value like on the next image.



If you are using a sprite to change palettes, remember to use PHX before the routine and PLX after routine.

V. Player Graphics and Palette Change

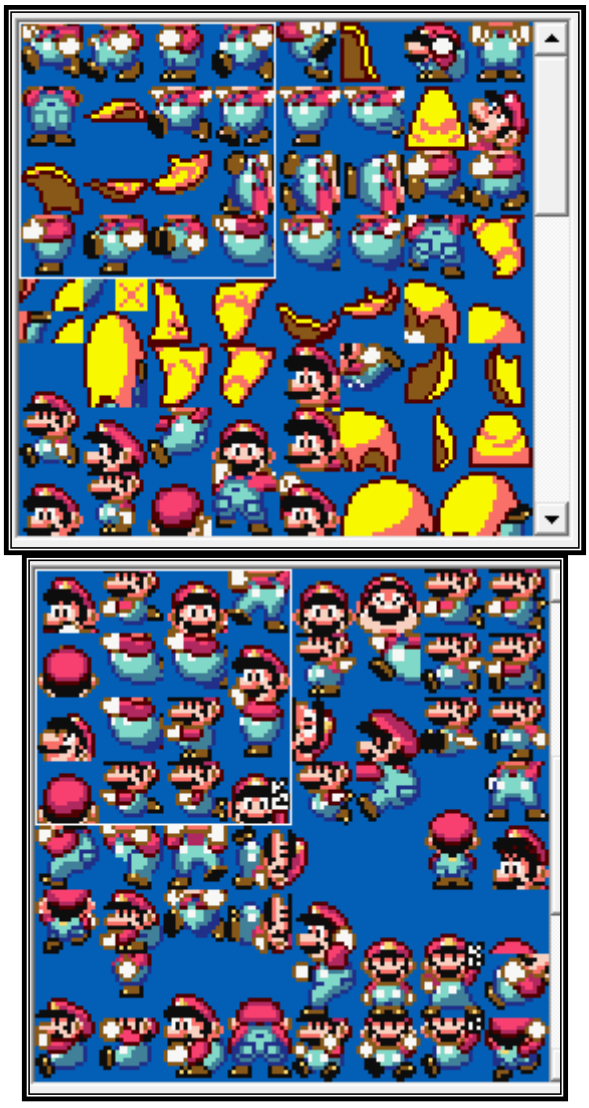
You can change the graphics and palette of the player, you can use this to make power ups or make a player that uses the same number of frame than Mario and the same size of Mario.

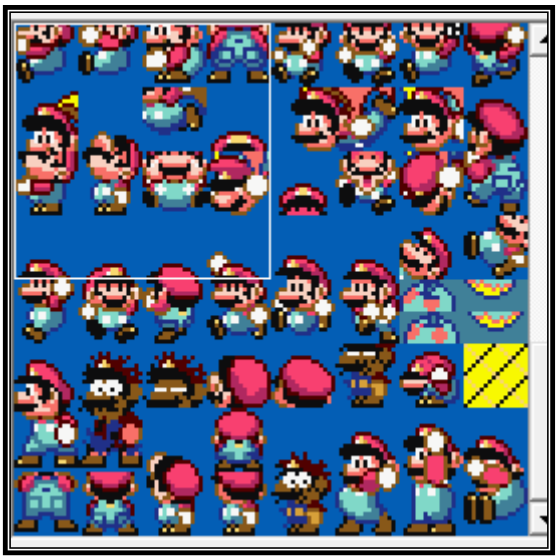
1. Graphic Change

To change player graphics is very similar to change other graphics, you need the following RAMs:

- !marioNormalCustomGFXOn (1 byte): if is #$01 enable the custom Mario graphics.
- !marioNormalCustomGFXBnk (1 byte): Bank of the new GFX.
- !marioNormalCustomGFXRec (2 bytes): ROM Address of the new GFX.

And you will need a GFX similar to GFX32:





Also you need to insert the graphics in the ROM with a patch like this:

```
!FreeSpace = $308000

org !FreeSpace

db "ST","AR"
dw End-Start-$01
dw End-Start-$01^$FFFF

Start:
incbin APlayerGFX.bin ;replace this for your GFX name
End:
```

Now you must follow these steps:

- Set !marioNormalCustomGFXOn in #$01
- Fill !marioNormalCustomGFXBnk and !marioNormalCustomGFXRec.

!marioNormalCustomGFXBnk will be the first 2 digits of the FreeSpace that you use on the patch and !marioNormalCustomGFXRec will be the last 4 digits of the FreeSpace + 8.

In code is like this:

```
ChangePlayerGraphics:
	LDA #$01
	STA !marioNormalCustomGFXOn

	LDA #$30
	STA !marioNormalCustomGFXBnk

	REP #$20
	LDA #$8008
	STA !marioNormalCustomGFXRec
	SEP #$20
	RTS
```

You must call this routine every frame the player changes its graphics.

2. Palette Change

To change player palette you need the following RAMs:

- !marioPal (1 byte): If it is #$01 enables the player palette change.
- !marioPalLow (15 bytes): Low byte of the new color.
- !marioPalHigh (15 bytes): High byte of the new color.

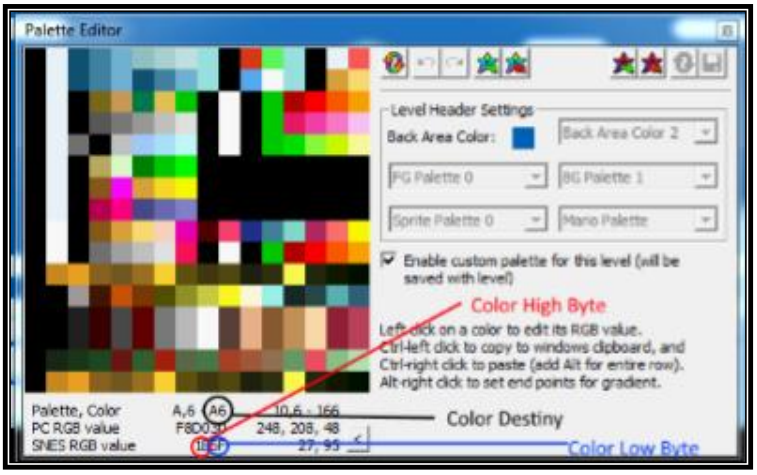
You need follow the steps:

- Set !marioPal to #$01.
- Use a loop to fill !marioPalLow,x and !marioPalHigh,x with the colors.

It is very similar to change a normal palette. In code is something like this:

```
;-------------86..87..88..89..8A..8B..8C..8D..8E..8f
;Fill these tables with the colors.
ColorLow: db $xx,$xx,$xx,$xx,$xx,$xx,$xx,$xx,$xx,$xx
ColorHih: db $xx,$xx,$xx,$xx,$xx,$xx,$xx,$xx,$xx,$xx

ChangeMarioPal:
       LDA #$01
       STA !marioPal

       LDX #$09
.loop
       LDA ColorLow,x
       STA !marioPalLow,x

       LDA ColorHih,x
       STA !marioPalHigh,x

       DEX
       BPL .loop
RTS
```

Like on the palette change chapter, to know what are the low byte and the high byte of a color you can use the next diagram:



If you are using a sprite to change the player palette, remember to use a PHX before routine and a PLX after routine.

Player Palette change only will change color 0x86, 0x87, 0x88, 0x89, 0x8A, 0x8B, 0x8C, 0x8D, 0x8E and 0x8F.

VI. Sprites

You can make or use 3 kinds of sprites with this patch, Dynamic Sprites, Semi-Dynamic Sprites and Giant Dynamic Sprites. If you want to know how to make or use each kind of sprite then Read each section.

1. Sprite Creation: General Stuffs.

Every kind of sprite that you can make with this patch, you can make it with Dyzen V3.0 or higher, or the templates in the Templates Folder.

If you use the templates or Dyzen you will find the following RAMs:

- !FramePointer: Is the ID of the current image that is showing your sprite.
- !AnPointer: Is the double of the ID of the current animation of your sprite.
- !AnFramePointer: is an index used to change between the different frames on the current animation. 0 is the first frame of the animation, 1 is the second frame of the animation, 2 is the third, and so on.
- !AnimationTimer: Is a timer used to know in how many SNES frames the current frame will change.
- !invisible: used to delay the sprite up to load its graphics on the VRAM, DON'T TOUCH THIS.
- !GlobalFlipper: Is a ram used to flip the sprite in X or Y.
    - #$00 = No flip.
    - #$01 = Only horizontal flip.
    - #$02 = Only vertical flip.
    - #$03 = Horizontal and vertical flip.
- !LocalFlipper: Used for flip into animations. DON'T TOUCH THIS.

And you will need to fill some tables. If you use Dyzen you don't need to fill the tables, but if you are using Templates you need it. I will show you how to fill every table on the template.

Animation Tables

Animation tables are:

- EndPositionAnim: It is the first position of the animation.
- AnimationsFrames: It has the IDs of the frame that have every animation.
- AnimationsNFr: It has the order of the next frame that will be used on the animation, It doesn't use IDs only the order.
- AnimationsTFr: It has the times in SNES frames that will be the next animation. Usually is 0x04.
- AnimationsFlips: It has information about the flipping of the frame.

I will explain this with an example. I have a sprite that has 2 animations, idle and walk.

Idle has 4 frames: Idle1, Idle2, Idle3 and Idle1 but flipped horizontally:

- Idle1: has the id #$00, the next frame will be Idle2 that is the second on the list of frames of the animation; uses 4 frames and is not flipped.
- Idle2: has the id #$04, the next frame will be Idle3 that is the third on the list of frames of the animation; uses 6 frames and is not flipped.
- Idle3: has the id #$02, the next frame will be a flipped Idle1 that is the fourth on the list of frames of the animation; uses 2 frames and is not flipped.
- Flipped Idle1: has the id #$00, the next frame will be Idle1 that is the first on the list of frames of the animation; uses 4 frames and is not flipped.

Then the tables for the Idle animation will be:

```
idleFrames: db $00,$04,$02,$00

idleNext: db $01,$02,$03,$00

idleTimes: db $04,$06,$02,$04

idleFlip: db $00,$00,$00,$01
```

Walk has 2 frames, walk1 and walk2:

- Walk1: has the id #$01, the next frame will be walk2 that is the second on the list of frames of the animation; uses 4 frames and is not flipped.
- Walk2: has id #$03, the next frame will be walk1 that is the first on the list of frames of the animation; uses 4 frames and is not flipped.

Then the tables for the walk animation will be:

```
walkFrames: db $01,$03

walkNext: db $01,$00

walkTimes: db $04,$04

walkFlip: db $00,$00
```

Now to fill EndPositionAnim you must know that the first animation is idle and idle uses 4 frames, the first value of EndPositionAnim is always #$0000, then you must add the number of frames of the animation to know the next value.

In this case EndPositionAnim will be:

```
EndPositionAnim: dw $0000,$0004
```

Because idle uses 4 frames. If we have a third animation then it is like this:

```
EndPositionAnim: dw $0000,$0004,$0006
```

Because idle uses 4 frames and walk uses 2 frames.

Then to fill all tables you will have this:

```
EndPositionAnim: dw $0000,$0004

AnimationsFrames:
idleFrames:
       db $00,$04,$02,$00
walkFrames:
       db $01,$03

AnimationsNFr:
idleNext:
       db $01,$02,$03,$00
walkNext:
       db $01,$00

AnimationsTFr:
idleTimes:
       db $04,$06,$02,$04
walkTimes:
       db $04,$04

AnimationsFlips:
idleFlip:
       db $00,$00,$00,$01
walkFlip:
       db $00,$00
```

Frames Tables

Frames Tables are:

- FramesTotalTiles: Number of tiles that have each frame.
- StartPositionFrames: Position of the last tile of each frame.
- EndPositionFrames: Position of the first tile of each frame.
- FramesXDisp: X Displacement relative to sprite X position of each tile of the frame.
- FramesYDisp: Y Displacement relative to sprite Y position of each tile of the frame.
- FramesPropertie: Properties of each tile of the frame. They use format YXPPCCCT.
- FramesTile: Tile number of each tile of the frame.

I will explain this with an example. I have 2 frames, walk1 and walk2.

Walk1 use 4 tiles:

- Tile 1 has X and Y Displacement 0, is the tile 0x80 and use propety 0x3E.
- Tile 2 has X = 0x10 and Y = 0, is the tile 0x82 and use propety 0x3C.
- Tile 3 has X = 0 and Y = 0x10, is the tile 0x60 and use propety 0x2C.
- Tile 4 has X = 0x10 and Y = 0x10, is the tile 0x62 and use propety 0x2E.

Then the tables for this table are:

```
walk1XDisp: db $00,$10,$00,$10

walk1YDisp: db $00,$00,$10,$10

walk1Properties: db $3E,$3C,$2C,$2E

walk1Tiles: db $80,$82,$60,$62
```

Walk2 use 2 tiles:

- Tile 1 has X and Y Displacement 0, is the tile 0xA0 and use propety 0x3E.
- Tile 2 has X = 0x10 and Y = 0, is the tile 0xA2 and use propety 0x3C.

Then the tables for this table are:

```
walk2XDisp: db $00,$10

walk2YDisp: db $00,$00

walk2Properties: db $3E,$3C

walk2Tiles: db $A0,$A2
```

Also if your sprite can be flipped horizontally or vertically, you must add others tables with the flip for each frame.

```
walk1XDispFlipX: db $10,$00,$10,$00

walk1YDispFlipX: db $00,$00,$10,$10

walk1PropertiesFlipX: db $7E,$7C,$6C,$6E

walk1TilesFlipX: db $80,$82,$60,$62

walk1XDispFlipY: db $00,$10,$00,$10

walk1YDispFlipY: db $10,$10,$00,$00

walk1PropertiesFlipY: db $BE,$BC,$AC,$AE

walk1TilesFlipY: db $80,$82,$60,$62

walk1XDispFlipXY: db $10,$00,$10,$00

walk1YDispFlipXY: db $10,$10,$00,$00

walk1PropertiesFlipXY: db $FE,$FC,$EC,$EE

walk1TilesFlipXY: db $80,$82,$60,$62
```

```
walk2XDispFlipX: db $10,$00

walk2YDispFlipX: db $00,$00

walk2PropertiesFlipX: db $7E,$7C

walk2TilesFlipX: db $A0,$A2

walk2XDispFlipY: db $00,$10

walk2YDispFlipY: db $00,$00

walk2PropertiesFlipY: db $BE,$BC

walk2TilesFlipY: db $A0,$A2

walk2XDispFlipXY: db $10,$00

walk2YDispFlipXY: db $00,$00

walk2PropertiesFlipXY: db $FE,$FC

walk2TilesFlipXY: db $A0,$A2
```

Then complete tables are:

```
FramesXDisp:
walk1XDisp:
       db $00,$10,$00,$10
walk2XDisp:
       db $00,$10
walk1XDispFlipX:
       db $10,$00,$10,$00
walk2XDispFlipX:
       db $10,$00
walk1XDispFlipY:
       db $00,$10,$00,$10
walk2XDispFlipY:
       db $00,$10
walk1XDispFlipXY:
       db $10,$00,$10,$00
walk2XDispFlipXY:
       db $10,$00

FramesYDisp:
walk1YDisp:
       db $00,$00,$10,$10
walk2YDisp:
       db $00,$00
walk1YDispFlipX:
       db $00,$00,$10,$10
walk2YDispFlipX:
       db $00,$00
walk1YDispFlipY:
       db $10,$10,$00,$00
walk2YDispFlipY:
       db $00,$00
walk1YDispFlipXY:
       db $10,$10,$00,$00
```

```
walk2YDispFlipXY:
	db $00,$00

FramesPropertie:
walk1Properties:
	db $3E,$3C,$2C,$2E
walk2Properties:
	db $3E,$3C
walk1PropertiesFlipX:
	db $7E,$7C,$6C,$6E
walk2PropertiesFlipX:
	db $7E,$7C
walk1PropertiesFlipY:
	db $BE,$BC,$AC,$AE
walk2PropertiesFlipY:
	db $BE,$BC
walk1PropertiesFlipXY:
	db $FE,$FC,$EC,$EE
walk2PropertiesFlipY:
	db $BE,$BC

FramesTile:
walk1Tiles:
	db $80,$82,$60,$62
walk2Tiles:
	db $A0,$A2
walk1TilesFlipX:
	db $80,$82,$60,$62
walk2TilesFlipX:
	db $A0,$A2
walk1TilesFlipY:
	db $80,$82,$60,$62
walk2TilesFlipY:
	db $A0,$A2
walk1TilesFlipXY:
	db $80,$82,$60,$62
walk2TilesFlipXY:
	db $A0,$A2
```

Now to fill FramesTotalTiles you must put the number of tiles -1 of every frame and its flips:

```
FramesTotalTiles:
	db $03,$01,$03,$01,$03,$01,$03,$01
```

To fill StartPositionFrames we must put the position of the last tile of every frame and its flips.

```
StartPositionFrames:
	dw $0003,$0005,$0009,$000B,$000F,$0011,$0015,$0017
```

To fill EndPositionFrameswe must put the position of the first tile of every frame and its flips.

```
EndPositionFrames:
	dw $0000,$0004,$0006,$000A,$000C,$0010,$0012,$0016
```

How to use Tables and RAMs

To change between the different animations you must follow these steps:

- If !AnimationTimer isn't #$00 then return.
- Set !AnFramePointer to #$00.
- Set !AnPointer with the double of the ID of the new animation.
- Set !FramePointer with the ID of the first frame of the new animation.
- Set !AnimationTimer with the number of frames that you want to delay the first frame of the new animation. It can only start in a non-zero, even number.

To know the value of !FramePointer you can see the table AnimationsFrames and search for the respective table and use the first value.

To know the value of !AnimationTimer you can see the table AnimationsTFr and search for the respective table and use the first value.

For example if I have this tables:

```
EndPositionAnim: dw $0000,$0004

AnimationsFrames:
idleFrames:
       db $00,$04,$02,$00
walkFrames:
       db $01,$03

AnimationsNFr:
idleNext:
       db $01,$02,$03,$00
walkNext:
       db $01,$00

AnimationsTFr:
idleTimes:
       db $04,$06,$02,$04
walkTimes:
       db $04,$04

AnimationsFlips:
idleFlip:
       db $00,$00,$00,$01
walkFlip:
       db $00,$00
```

The value for walk will be:

- !FramePointer = #$01
- !AnimationTimer = #$04
- !AnPointer = #$02

And the value for idle will be:

- !FramePointer = #$00
- !AnimationTimer = #$04
- !AnPointer = #$00

In code is like this:

```
ChangeAnimation:

	LDA !AnimationTimer
	BNE .ret

	STZ !AnFramePointer
	LDA #$firstFrameOfTheNewAnimation
	STA !FramePointer
	LDA #$TimeOfTheFirstFrameofTheNewAnimation
	STA !AnimationTimer
	LDA #$DoubleOfIdOfTheNewAnimation
	STA !AnPointer
.ret
	RTS
```

If you want to flip your sprite you only need to set !GlobalFlipper with the value that you want.

Also you can use these RAMs to make conditions, for example, if I want the animation #$03 at the end of the third frame to do something, I can do this:

```
CheckToDoSomething:
	LDA !AnimationTimer
	BNE .ret

	LDA !AnPointer
	CMP #$06
	BNE .ret

	LDA !AnFramePointer
	CMP #$02
	BNE .ret

	;Do Something

.ret
RTS
```

Or maybe you want that every time when a specific frame ends (for example frame #$0A), something happens:

```
CheckToDoSomething2:
	LDA !AnimationTimer
	BNE .ret

	LDA !FramePointer
	CMP #$0A
	BNE .ret

	;Do Something

.ret
RTS
```

In the case of Dynamic Sprites you will have a variable called !DynamicSwitch, to use the others variables is the same but when you want to change animation you must make this:

```
ChangeAnimation:

	LDA !DynamicSwitch
	BEQ .ret

	STZ !AnFramePointer
	LDA #$firstFrameOfTheNewAnimation
	STA !FramePointer
	LDA #$TimeOfTheFirstFrameofTheNewAnimation
	STA !AnimationTimer
	LDA #$DoubleOfIdOfTheNewAnimation
	STA !AnPointer

	JSR Graphics
.ret
	RTS
```

!DynamicSwitch will say you if in this frame the Dynamic Routine will happend, also you must recall Graphic Routine to avoid a graphic glitch.

2. Dynamic Sprites

A Dynamic Sprite is like any other sprite, the only different thing is that Dynamic Sprites load their own graphics to VRAM periodically allowing a lot of frames saving space on VRAM and using very smooth animations.

Dynamic Sprites can use all SP4, Dynamic Z will try to put every Dynamic Sprite at the first half of SP4 and, if the first half is completely used, then it will use the second half of SP4. If you use “50% more” mode then Dynamic Z will start to use the second half of SP3 and all SP4.

This patch allows 4 sizes of Dynamic Sprites, 32x32, 48x48, 64x64 and 80x80. You have a limited number of Dynamic Sprites that you can use on the screen, you can have maximum:

- 8 of 32x32.
- 4 of 48x48.
- 2 of 64x64.
- 2 of 32x32 and 3 of 48x48.
- 2 of 32x32,1 of 48x48 and 1 of 64x64.
- 4 of 32x32 and 2 of 48x48.
- 4 of 32x32 and 1 of 64x64.
- 6 of 32x32 and 1 of 48x48.
- 2 of 48x48 and 1 of 64x64.

If you use “50% more” mode the maximum is:

- 12 of 32x32
- 6 of 48x48
- 2 of 48x48 and 2 of 64x64*****
- 2 of 80x80
- 2 of 32x32 and 5 of 48x48
- 2 of 32x32, 3 of 48x48 and 1 of 64x64*****
- 2 of 32x32, 1 of 48x48 and 2 of 64x64*****
- 2 of 32x32, 2 of 48x48 and 1 of 80x80*****
- 2 of 32x32, 1 of 64x64 and 1 of 80x80*****
- 4 of 32x32 and 4 of 48x48
- 4 of 32x32, 2 of 48x48 and 1 of 64x64*****
- 4 of 32x32 and 2 of 64x64*****
- 4 of 32x32, 1 of 48x48 and 1 of 80x80*****
- 6 of 32x32 and 3 of 48x48
- 6 of 32x32, 1 of 48x48 and 1 of 64x64*****
- 6 of 32x32 and 1 of 80x80*****
- 8 of 32x32 and 2 of 48x48
- 8 of 32x32 and 1 of 64x64*****
- 10 of 32x32 and 1 of 48x48

*****This combination could not show a sprite of 64x64 or 80x80, try to use fewer sprites.

If you use more than the maximum, the patch will erase the sprite that can’t be loaded and this sprite could be reloaded again if the player enters the screen again and there are less Dynamic Sprites.

How to make a Dynamic Sprite

To make Dynamic Sprite first you need to insert a GFX on the ROM. To insert it you can use Direct Transfer or Indirect Transfer, you can use the templates of Direct Transfer of Indirect Transfer (look at the Templates folder).

If you use less than 32kb of graphics you should use Direct Transfer, If you are using it then you must find the next line:

```
resource:
incbin sprites\GFXName.bin
```

And replace GFXName by the name of your GFX. The GFX must be on the same folder of the sprite.

If you use more than 32kb you must use Indirect Transfer, if you are using it then you must use a patch like this:

```
!FreeSpace1 = $308000
!FreeSpace2 = $318000
!FreeSpace3 = $328000
!FreeSpace4 = $338000
;…

org !FreeSpace1

db "ST","AR"
dw End-Start-$01
dw End-Start-$01^$FFFF

Start:
incbin GFXName1.bin ;replace this for your GFX name
End:

org !FreeSpace2

db "ST","AR"
dw End-Start-$01
dw End-Start-$01^$FFFF


Start:
incbin GFXName2.bin ;replace this for your GFX name
End:
org !FreeSpace3

db "ST","AR"
dw End-Start-$01
dw End-Start-$01^$FFFF

Start:
incbin GFXName3.bin ;replace this for your GFX name
End:

org !FreeSpace4

db "ST","AR"
dw End-Start-$01
dw End-Start-$01^$FFFF

Start:
incbin GFXName4.bin ;replace this for your GFX name
End:
```

```
;…
```

Then you insert the patch on the ROM and then find these lines on the template:

```
!bnk0 = $30
!source0 = $8008

!bnk1 = $31
!source1 = $8008

!bnk2 = $32
!source2 = $8008

!bnk3 = $33
!source3 = $8008

!bnk4 = $34
!source4 = $8008

;.
;.
;.
```

And replace bankX with the first 2 digits of FreeSpaceX, and sourceX with the last 4 digits +8 of the FreeSpaceX.

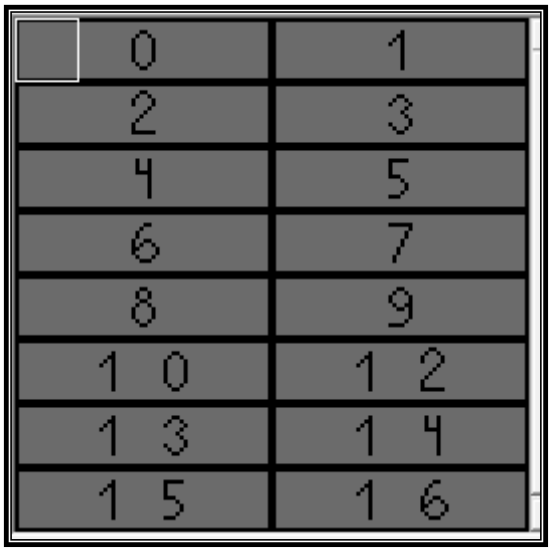
GFXs must be on the same folder of the patch.

Now you can't use any GFX, you need to sort the GFX depending on size of the sprite and fill a table.

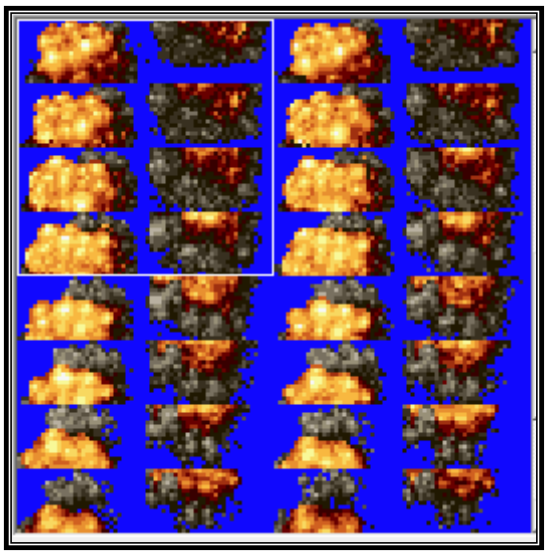
Dynamic Sprite of 32x32

Dynamic Sprites of 32x32 can only use tiles from SP1, SP2, the first half of SP3, and from SP4 only this tiles 0x80, 0x81, 0x82, 0x83, 0x84, 0x85 and 0x86.

The GFXs of Dynamic Sprites of 32x32 use this order:



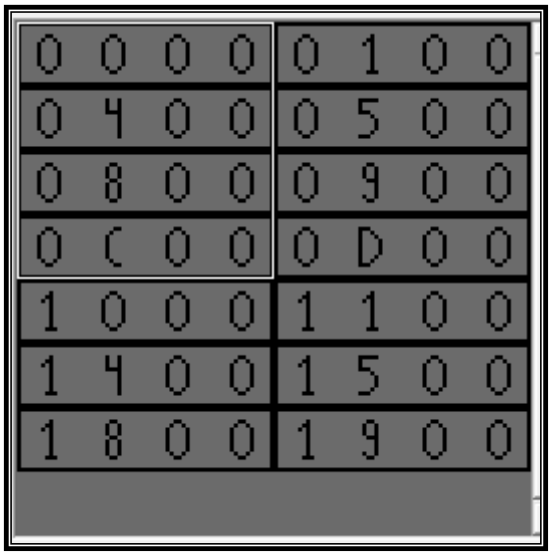
For example:



Then you must go to this line:

```
VramTable:
        dw
```

And fill it with these values:



For example:

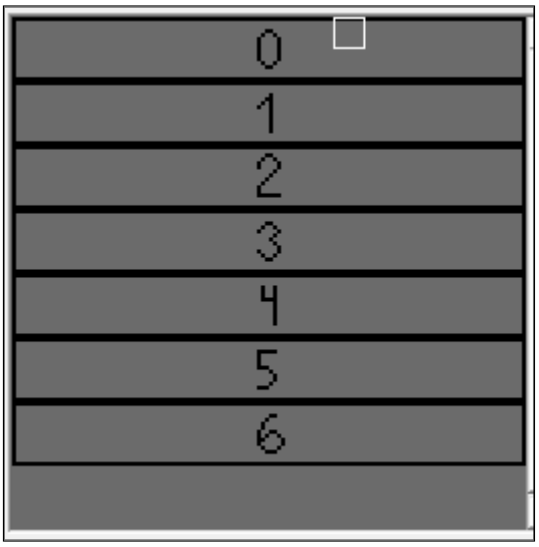
```
VramTable:
        dw $0000,$0100,$0400,$0500,$0800,$0900,$0C00,$0D00,$1000,$1100
        dw $1400,$1500,$1800,$1900,$1C00,$1D00,$2000,$2100,$2400,$2500
```

If there is a frame that doesn't use Dynamic Graphics then put $FFFF. Also if you are using Dyzen It will sort the GFX and make the table.

Dynamic Sprite of 48x48

Dynamic Sprites of 48x48 can only use tiles from SP1, SP2, the first half of SP3, and from SP4 only this tiles 0x80, 0x81, 0x82, 0x83, 0x84, 0x85, 0x86, 0x87, 0x88, 0x89, 0x8A, 0x8B, 0x8C, 0x8D and 0x8E.

The GFXs of Dynamic Sprites of 48x48 use this order:

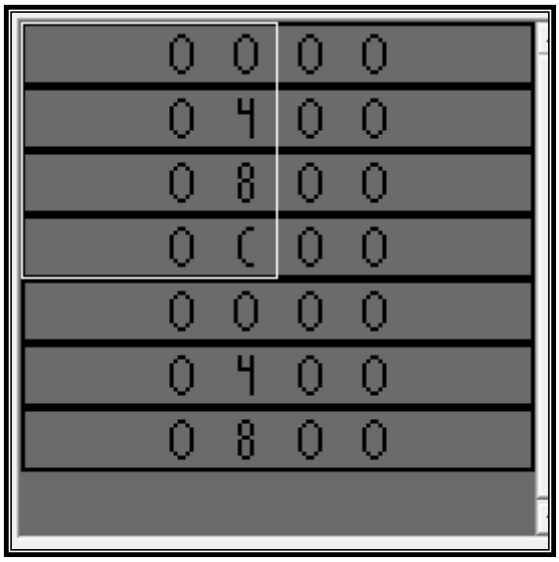


For example:

Then you must go to this line:

```
VramTable:
        dw
```

And fill it with these values:



For example:

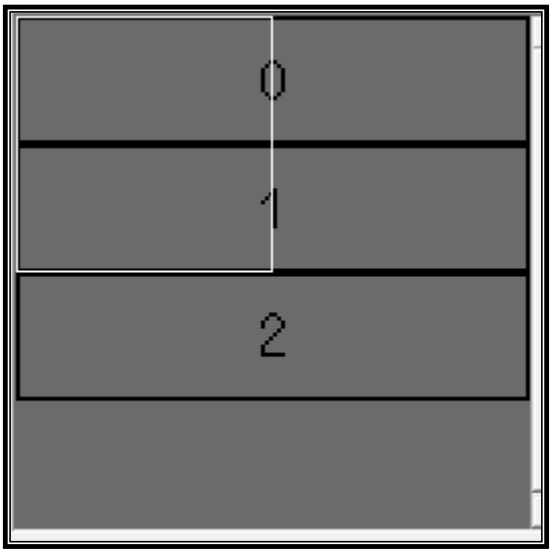
```
VramTable:
        dw $0000,$0400,$0800,$0C00,$1000,$1400,$1800,$1C00,$2000,$2400
        dw $2800,$2C00,$3000,$3400,$3800,$3C00,$4000,$4400,$4800,$4C00
```

If there is a frame that doesn't use Dynamic Graphics then put $FFFF. Also if you are using Dyzen It will sort the GFX and make the table.

Dynamic Sprite of 64x64

Dynamic Sprites of 64x64 can only use tiles from SP1, SP2, the first half of SP3, and the first half of SP4.

The GFXs of Dynamic Sprites of 64x64 use this order:

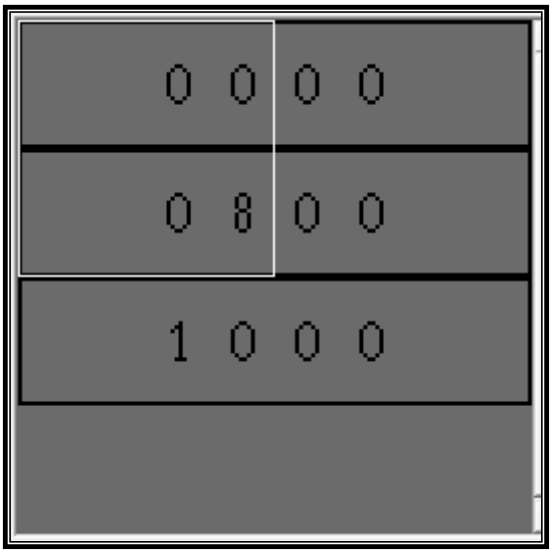


For example:

Then you must go to this line:

```
VramTable:
        dw
```

And fill it with these values:



For example:

```
VramTable:
	dw $0000,$0800,$1000,$1800,$2000,$2800,$3000,$3800
	dw $4000,$4800,$5000,$5800,$6000,$6800,$7000
```

If there is a frame that doesn't use Dynamic Graphics then put $FFFF. Also if you are using Dyzen It will sort the GFX and make the table.

3. 50% More Mode

This mode allows to use 50% more Dynamic Sprites and use Dynamic sprites of 80x80 and Giant Dynamic Sprites of 112x112, to activate it you only need to put this line:

```
LDA #$01
STA !mode50
```

In level_code of Level/uber asm, remember to put this line at the start of Level/uber asm:

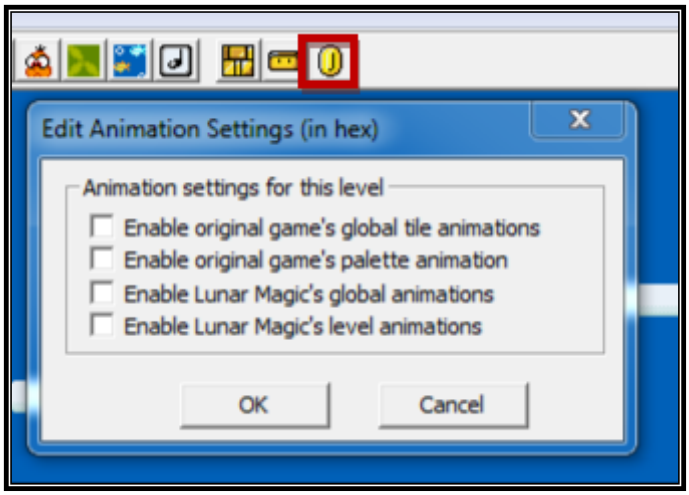
```
incsrc header.asm
```

And put a copy of header.asm on the same folder of Level/uber asm.

This mode will disable the following:

- Yoshi.
- Podoboos.
- Every SP1 DMA.
- Status Bar.

Also you must disable this on the level:

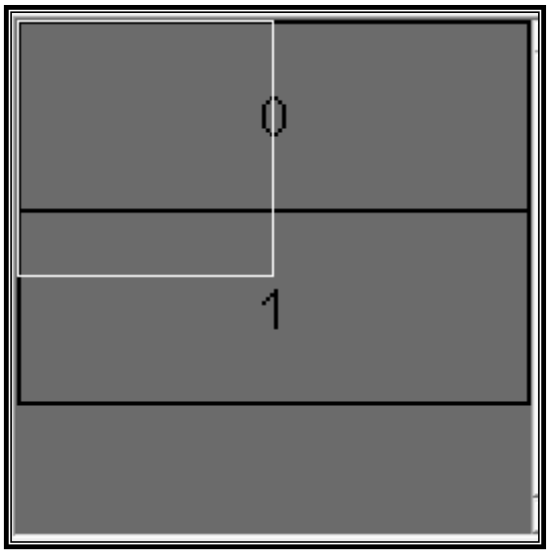


Dynamic Sprite of 80x80

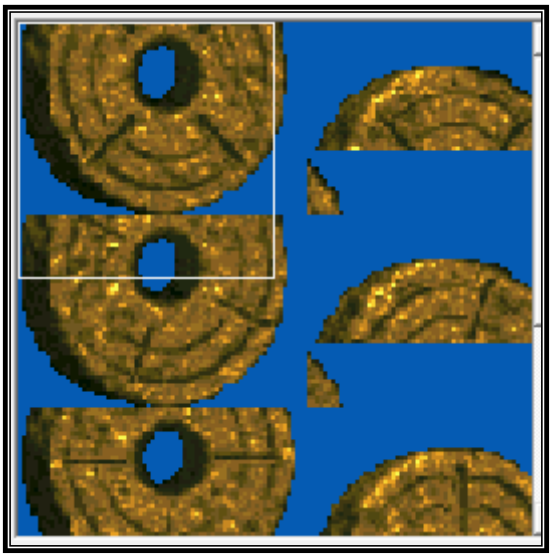
To use Dynamic Sprites of 80x80 you need to use mode 50% more. You can use Dyzen V3.2 or higher to make them.

Dynamic Sprites of 80x80 can only use tiles from SP1, SP2, the first half of SP3, the first half of SP4 and from the second half of SP4 only can use these tiles 0xC0, 0xC1, 0xC2, 0xC3, 0xC4, 0xC5, 0xC6, 0xC7, 0xC8, 0xC9, 0xCA, 0xCB, 0xCC, 0xCD and 0xCE.

The GFXs of Dynamic Sprites of 80x80 use this order:



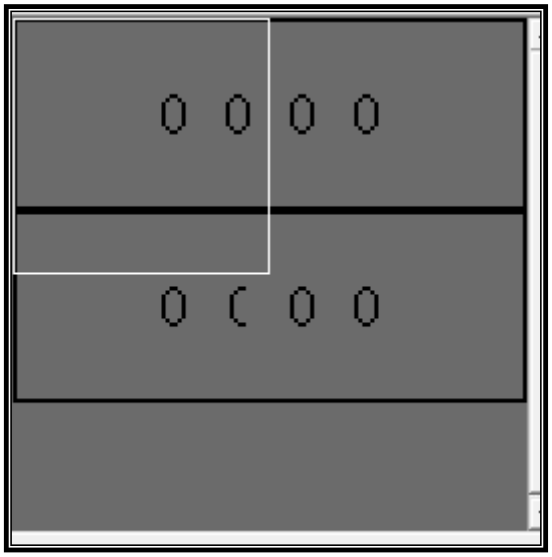
For example:



Then you must go to this line:

```
VramTable:
        dw
```

And fill it with these values:



For example:

```
VramTable:
        dw $0000,$0C00,$1800,$2400,$3000,$3C00,$4800,$5400,$6000
```

If there is a frame that doesn't use Dynamic Graphics then put $FFFF. Also if you are using Dyzen V3.2 or higher It will sort the GFX and make the table.

4. Giant Dynamic Sprites

You can make or use Dynamic sprites of 96x96 or 112x112 with Dynamic Z, You can have only 1 Giant Dynamic Sprite on the screen. Dynamic Sprites of 96x96 will use all SP2 and SP3 and Dynamic Sprites of 112x112 will use the second half of SP1, SP2, SP3 and the first half of SP4.

You can use Dyzen V3.2 or higher to make them.

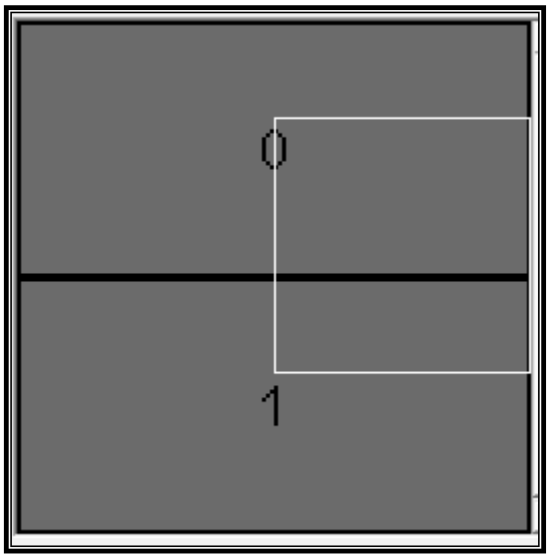
The way to make them is the same like a normal Dynamic Sprite.

Giant Dynamic Sprites of 96x96

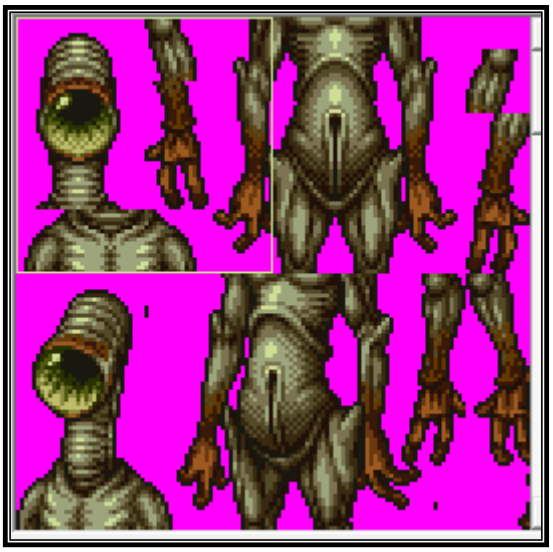
You can use Dyzen V3.2 or higher to make them.

Giant Dynamic Sprites of 96x96 only can use tiles in the second Half of SP1, SP2 and SP4.

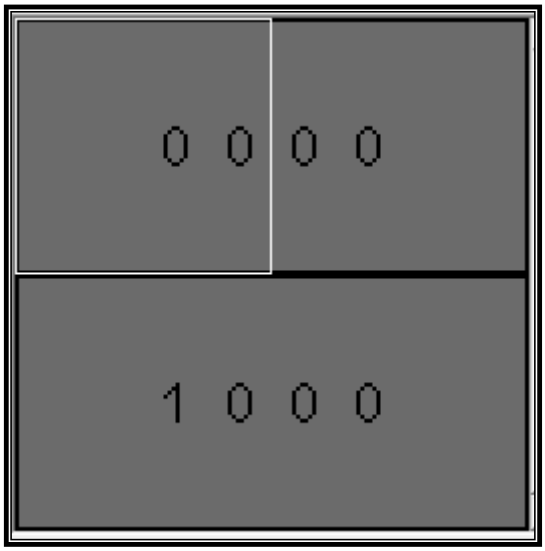
The GFXs of Dynamic Sprites of 96x96 use this order:



For example:



And fill it with these values:



For example:

```
VramTable:
        dw $0000,$1000,$2000,$3000,$4000,$5000,$6000
```

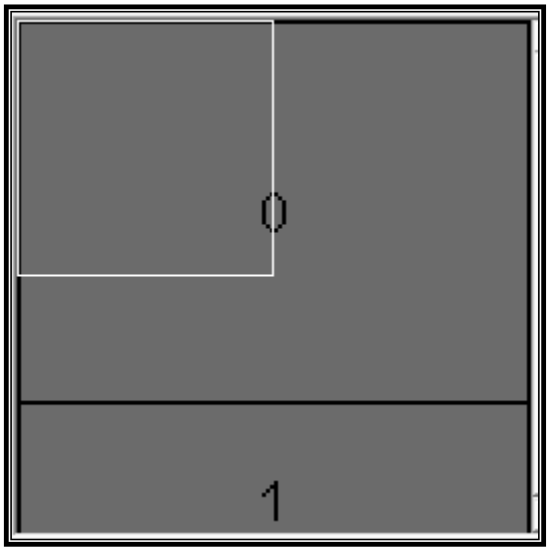
If there is a frame that doesn't use Dynamic Graphics then put $FFFF. Also if you are using Dyzen V3.2 or higher It will sort the GFX and make the table.

Giant Dynamic Sprites of 112x112

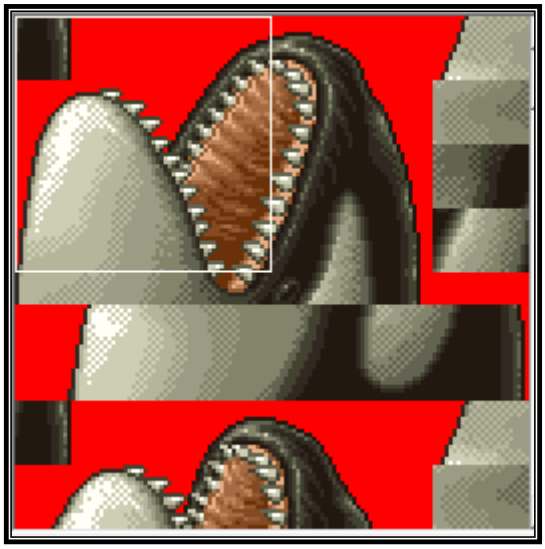
To use Giant Dynamic Sprites of 112x112 you need to use mode 50% more. You can use Dyzen V3.2 or higher to make them.

Giant Dynamic Sprites of 112x112 only can use tiles in the second Half of SP1, SP2 and the second Half of SP4.

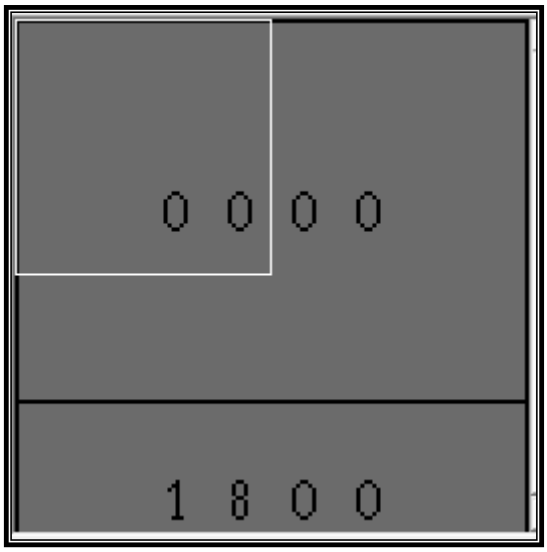
The GFXs of Dynamic Sprites of 112x112 use this order:



For example:



And fill it with these values:



For example:

```
VramTable:
	dw $0000,$1800,$3000,$4800,$6000
```

If there is a frame that doesn't use Dynamic Graphics then put $FFFF. Also if you are using Dyzen V3.2 or higher It will sort the GFX and make the table.

5. Semi-Dynamic Sprites

Semi-Dynamic Sprites are like any other normal sprite, but you don't need to put their graphics on the level. When you use a Semi-Dynamic Sprite it will load its entire graphics when it starts on the screen and will share graphics with any other copy of the sprite. They don't have limitation, so you can use 8 or 12 of any size. Now there are 3 kinds of Semi-Dynamic Sprites:

- Half GFX: it uses only a half of GFX.
- GFX: it uses a complete GFX.
- 2GFX: it uses 2 GFXs.

You can select what zone of the VRAM the sprite will occupy, using the extra property byte 1.

In the case of Half-GFX:

- Extra Property 0 = first half of SP2.
- Extra Property 1 = second half of SP2.
- Extra Property 2 = first half of SP3.
- Extra Property 3 = second half of SP3.

In the case of GFX:

- Extra Property 0 = SP2.
- Extra Property 1 = SP3.

2GFX always use all SP2 and SP3.

Semi-Dynamic Sprites only can use a few tiles, in the case of Half GFX they only can use tiles in SP1 and first half of SP2, in the case of GFX they only can use tiles in SP1 and SP2 and in the case of 2GFX they only can use tiles on SP1, SP2 and SP3.

To make a Semi-Dynamic Sprite you only need to go to this line:

```
resource:
incbin sprites\GFXName.bin
```

And replace GFXName for the name of your GFX. In the case of half GFX they must use a GFX of 2kb, in the case of GFX they must use a GFX of 4kb and in the case of 2GFX they must use a GFX of 8kb.

To use a Semi-Dynamic Sprite you only need to put the GFX in the same folder of the sprite and insert it like any other sprite, only remember the extra property stuff.

I recommend transferring the palette of the sprite when it starts on the level with the Palette Change system of the patch.

VII. Custom Player

You can make a player of 48x48 with Dynamic Z. This patch will only make the Dynamic Routine; you must make the graphic routine on a separate patch. If you use this, you can't use Yoshi or podoboos.

To make a Custom Player you need the following RAMs:

- !marioCustomGFXOn (1 byte): Activate custom player, must be set in #$01 every frame.
- !marioCustomGFXBnk (1 byte): Bank of the resource.
- !marioCustomGFXRec (2 bytes): RAM address of the resource.

To use this function you need to follow these steps:

- Set !marioCustomGFXOn in #$01.
- Fill !marioCustomGFXBnk and !marioCustomGFXRec.

To fill !marioCustomGFXBnk and !marioCustomGFXRec you have 2 ways.

First way is to use Direct Transfer. In this case your code should be like this:

```
Vram: dw $0000,$0400,$0800,$0C00,$1000,$1400,$1800,$1C00
      dw $2000,$2400,$2800,$2C00,$3000,$3400,$3800,$3C00
      dw $4000,$4400,$4800,$4C00,$5000,$5400,$5800,$5C00
      dw $6000,$6400,$6800,$6C00,$7000,$7400,$7800

PlayerDynamicRoutine:
      LDA #$01
      STA !marioCustomGFXOn

                          ;you will need to use a ram to know
      LDA !FramePointer   ;what is the current frame of the player
      ASL
      TAX

      PHB
      PLA
      STA !marioCustomGFXBnk

      REP #$20
      LDA GFXPointer
      CLC
      ADC Vram,x
      STA !marioCustomGFXRec
      SEP #$20

      RTS

GFXPointer:
dw resource

resource:
incbin playerGFX.bin ;replace this with your GFX name
```

This way is only for players that use 32kb of graphics or less. If you want to use more you need Indirect Transfer.

Second Way is to use Indirect Transfer, to do this you need a patch like this:

```
!FreeSpace1 = $308000
!FreeSpace2 = $318000
!FreeSpace3 = $328000
!FreeSpace4 = $338000
;…

org !FreeSpace1

db "ST","AR"
dw End-Start-$01
dw End-Start-$01^$FFFF

Start:
incbin GFXName1.bin ;replace this for your GFX name
End:

org !FreeSpace2

db "ST","AR"
dw End-Start-$01
dw End-Start-$01^$FFFF


Start:
incbin GFXName2.bin ;replace this for your GFX name
End:
org !FreeSpace3

db "ST","AR"
dw End-Start-$01
dw End-Start-$01^$FFFF

Start:
incbin GFXName3.bin ;replace this for your GFX name
End:

org !FreeSpace4

db "ST","AR"
dw End-Start-$01
dw End-Start-$01^$FFFF

Start:
incbin GFXName4.bin ;replace this for your GFX name
End:

;…
```

Then in code is something like this:

```
!Bank1 = $30
!Bank2 = $31
!Bank3 = $32
!Bank4 = $33

!Source1 = $8008
!Source2 = $8008
!Source3 = $8008
!Source4 = $8008


Banks: db !Bank1,!Bank1,!Bank1,!Bank1,!Bank1,!Bank1,!Bank1,!Bank1
       db !Bank1,!Bank1,!Bank1,!Bank1,!Bank1,!Bank1,!Bank1,!Bank1
       db !Bank1,!Bank1,!Bank1,!Bank1,!Bank1,!Bank1,!Bank1,!Bank1
       db !Bank1,!Bank1,!Bank1,!Bank1,!Bank1,!Bank1,!Bank1
       db !Bank2,!Bank2,!Bank2,!Bank2,!Bank2,!Bank2,!Bank2,!Bank2
       db !Bank2,!Bank2,!Bank2,!Bank2,!Bank2,!Bank2,!Bank2,!Bank2
       db !Bank2,!Bank2,!Bank2,!Bank2,!Bank2,!Bank2,!Bank2,!Bank2
       db !Bank2,!Bank2,!Bank2,!Bank2,!Bank2,!Bank2,!Bank2
       db !Bank3,!Bank3,!Bank3,!Bank3,!Bank3,!Bank3,!Bank3,!Bank3
       db !Bank3,!Bank3,!Bank3,!Bank3,!Bank3,!Bank3,!Bank3,!Bank3
       db !Bank3,!Bank3,!Bank3,!Bank3,!Bank3,!Bank3,!Bank3,!Bank3
       db !Bank3,!Bank3,!Bank3,!Bank3,!Bank3,!Bank3,!Bank3
       db !Bank4,!Bank4,!Bank4,!Bank4,!Bank4,!Bank4,!Bank4,!Bank4
       db !Bank4,!Bank4,!Bank4,!Bank4,!Bank4,!Bank4,!Bank4,!Bank4
       db !Bank4,!Bank4,!Bank4,!Bank4,!Bank4,!Bank4,!Bank4,!Bank4
       db !Bank4,!Bank4,!Bank4,!Bank4,!Bank4,!Bank4,!Bank4



Sources: dw !Source1,!Source1,!Source1,!Source1,!Source1,!Source1,!Source1,!Source1
       dw !Source1,!Source1,!Source1,!Source1,!Source1,!Source1,!Source1,!Source1
       dw !Source1,!Source1,!Source1,!Source1,!Source1,!Source1,!Source1,!Source1
       dw !Source1,!Source1,!Source1,!Source1,!Source1,!Source1,!Source1
       dw !Source2,!Source2,!Source2,!Source2,!Source2,!Source2,!Source2,!Source2
       dw !Source2,!Source2,!Source2,!Source2,!Source2,!Source2,!Source2,!Source2
       dw !Source2,!Source2,!Source2,!Source2,!Source2,!Source2,!Source2,!Source2
       dw !Source2,!Source2,!Source2,!Source2,!Source2,!Source2,!Source2
       dw !Source3,!Source3,!Source3,!Source3,!Source3,!Source3,!Source3,!Source3
       dw !Source3,!Source3,!Source3,!Source3,!Source3,!Source3,!Source3,!Source3
       dw !Source3,!Source3,!Source3,!Source3,!Source3,!Source3,!Source3,!Source3
       dw !Source3,!Source3,!Source3,!Source3,!Source3,!Source3,!Source3
       dw !Source4,!Source4,!Source4,!Source4,!Source4,!Source4,!Source4,!Source4
       dw !Source4,!Source4,!Source4,!Source4,!Source4,!Source4,!Source4,!Source4
       dw !Source4,!Source4,!Source4,!Source4,!Source4,!Source4,!Source4,!Source4
       dw !Source4,!Source4,!Source4,!Source4,!Source4,!Source4,!Source4

Vram: dw $0000,$0400,$0800,$0C00,$1000,$1400,$1800,$1C00
       dw $2000,$2400,$2800,$2C00,$3000,$3400,$3800,$3C00
       dw $4000,$4400,$4800,$4C00,$5000,$5400,$5800,$5C00
       dw $6000,$6400,$6800,$6C00,$7000,$7400,$7800
       dw $0000,$0400,$0800,$0C00,$1000,$1400,$1800,$1C00
       dw $2000,$2400,$2800,$2C00,$3000,$3400,$3800,$3C00
       dw $4000,$4400,$4800,$4C00,$5000,$5400,$5800,$5C00
       dw $6000,$6400,$6800,$6C00,$7000,$7400,$7800
       dw $0000,$0400,$0800,$0C00,$1000,$1400,$1800,$1C00
       dw $2000,$2400,$2800,$2C00,$3000,$3400,$3800,$3C00
```

```
        dw $4000,$4400,$4800,$4C00,$5000,$5400,$5800,$5C00
        dw $6000,$6400,$6800,$6C00,$7000,$7400,$7800
        dw $0000,$0400,$0800,$0C00,$1000,$1400,$1800,$1C00
        dw $2000,$2400,$2800,$2C00,$3000,$3400,$3800,$3C00
        dw $4000,$4400,$4800,$4C00,$5000,$5400,$5800,$5C00
        dw $6000,$6400,$6800,$6C00,$7000,$7400,$7800


PlayerDynamicRoutine:
        LDA #$01
        STA !marioCustomGFXOn

                            ;you will need to use a ram to know
        LDA !FramePointer    ;what is the current frame of the player
        TAX

        LDA Banks,x
        STA !marioCustomGFXBnk

        TXA
        ASL
        TAX

        REP #$20
        LDA Sources,x
        CLC
        ADC Vram,x
        STA !marioCustomGFXRec
        SEP #$20

        RTS
```

!BankX will be the first 2 digits of the !FreeSpaceX, and !SourceX will be the last 4 digits of the !FreeSpace +8.

This function will kill the original DMA routine of Mario and replace it for another DMA routine. I recommend when you change of frame then make the Dynamic Routine and then make the Graphic Routine. Also I recommend using mode 50% more if you use this function. Also you must call the Dynamic routine every SNES frame.

VIII. Routines

This is the list of routines that are included on the patch:

- !reserveNormalSlot80: Used to reserve a slot if you are using a normal sprite of 80x80.
- !reserveExtendedSlot80: Used to reserve a slot if you are using a extended sprite of 80x80.
- !reserveClusterSlot80: Used to reserve a slot if you are using a cluster sprite of 80x80. It will assume that the sprite index is in Y Register.
- !reserveNormalSlot64: Used to reserve a slot if you are using a normal sprite of 64x64.
- !reserveExtendedSlot64: Used to reserve a slot if you are using a extended sprite of 64x64.
- !reserveClusterSlot64: Used to reserve a slot if you are using a cluster sprite of 64x64. It will assume that the sprite index is in Y Register.
- !reserveNormalSlot48: Used to reserve a slot if you are using a normal sprite of 48x48.
- !reserveExtendedSlot48: Used to reserve a slot if you are using a extended sprite of 48x48.
- !reserveClusterSlot48: Used to reserve a slot if you are using a cluster sprite of 48x48. It will assume that the sprite index is in Y Register.
- !reserveNormalSlot32: Used to reserve a slot if you are using a normal sprite of 32x32.
- !reserveExtendedSlot32: Used to reserve a slot if you are using a extended sprite of 32x32.
- !reserveClusterSlot32: Used to reserve a slot if you are using a cluster sprite of 32x32. It will assume that the sprite index is in Y Register.
- !GET_DRAW_INFO: A version of Get_Draw_info that can be called with JSL. If the sprite is valid then $06 will be #$00 after this routine.
- !SUB_OFF_SCREEN_X0, X1,X2,X3,X4,X5,X6,X7: A version of Sub_Off_Screen that can be called with JSL.

- !DynamicRoutine80Start: Must be called for every sprite of 80x80 at the start of the Dynamic Routine. It must have the value of the Dynamic slot on !tileRelativePositionNormal,x, !tileRelativePositionExtended,x or !tileRelativePositionCluster,x in Y register before routine.

- !DynamicRoutine64Start: Must be called for every sprite of 64x64 at the start of the Dynamic Routine. It must have the value of the Dynamic slot on !tileRelativePositionNormal,x, !tileRelativePositionExtended,x or !tileRelativePositionCluster,x in Y register before routine.

- !DynamicRoutine48Start: Must be called for every sprite of 48x48 at the start of the Dynamic Routine. It must have the value of the Dynamic slot on !tileRelativePositionNormal,x, !tileRelativePositionExtended,x or !tileRelativePositionCluster,x in Y register before routine.

- !DynamicRoutine32Start: Must be called for every sprite of 32x32 at the start of the Dynamic Routine. It must have the value of the Dynamic slot on !tileRelativePositionNormal,x, !tileRelativePositionExtended,x or !tileRelativePositionCluster,x in Y register before routine.

- !Ping80: Used to know if the current Dynamic Sprite of 80x80 exists on the screen.

- !Ping64: Used to know if the current Dynamic Sprite of 64x64 exists on the screen. !Ping80: Used to know if the current Dynamic Sprite of 80x80 exists on the screen.

- !Ping48: Used to know if the current Dynamic Sprite of 48x48 exists on the screen.

- !Ping32: Used to know if the current Dynamic Sprite of 32x32 exists on the screen.

Every reserve routine will save the value of the Dynamic Slot in the RAM addresses !tileRelativePositionNormal,x, !tileRelativePositionExtended,x or !tileRelativePositionCluster,x.

IX. Compatibility

This patch is compatible with every old Dynamic sprite, but if you use an old Dynamic sprite on the screen, you can't use a New Dynamic sprite (i.e. made with Dynamic Z) on the same screen. Also you must not use DSX with this patch.

This patch doesn't have compatibility with any other patch that use org $00A300, like marioDMAer, 32x32 Player or LX's custom power ups. Also this patch doesn't have compatibility with the **status bar hijack of uber.asm** or other patches that use org $008E1A, org $008DAC or org $008292. To use this patch with uber asm you must deactivate the status bar hijack.

X. Other Content

Dynamic Z optimizes the Dynamic routine of Mario. The original Dynamic routine of Mario loads the graphics of Yoshi, Podoboos or the cape every frame even if they aren't used in the level. Also it loads the graphics every frame even if the graphics were already loaded. Dynamic Z's Dynamic routine of Mario, only loads the graphics of Yoshi, Podoboos or cape if they exist in the level and doesn't load the graphics if they were already loaded.

Also Dynamic Z's Sprites only load the graphics if they need to be loaded, whereas DSX's Dynamic Sprites loads their graphics all frames even if they were already loaded.

XI. Credits

Dynamic Z is a patch made by **anonimzwx**. If you have questions, send a PM to **anonimzwx**.

Special Thanks:	Masterlink DiscoTheBat FortalezaReznor LX5