Tetris NES to SRS

Chase Dillard & Atreyu McLewin

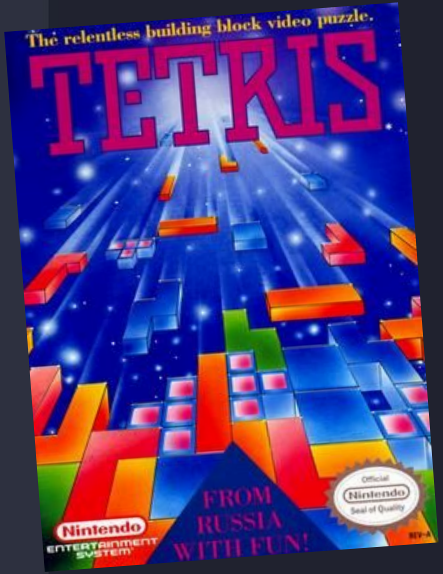
with special thanks to ROM-Hackers Kirjava, Kitaru, HydrantDude, and others!

github.com/GingerDeity/TetrisNES

Project Goals & Introduction

Tetris for the NES is a great game, BUT!

…it released in 1989, two years before the Standard Rotation System!


The relentless building block video puzzle.
TETRIS
FROM RUSSIA WITH FUN!
Official Nintendo Seal of Quality
Nintendo
ENTERTAINMENT SYSTEM

Our Goal? Add in modern Tetris mechanics!

- Add in missing rotation states
- Add in wall-kicking mechanics!

Project Approach

1) ***Modify logic as LITTLE as possible***

   AKA, use as much of original code as possible

2) ***Three environments, one for development, one for translating, and one for playtesting***
   - *Ghidra = translation & commenting*
   - *VSCode + GitHub = development & modifying*
   - *Mesen = playtesting & debugging*

Development Environment Tools

- *Disassembler (nesgodisasm)*
- *Assembler (CC65)*
- *Decompiler (Ghidra)*
- *Emulator (Mesen)*

cc65

- the 6502 C compiler

GHIDRA

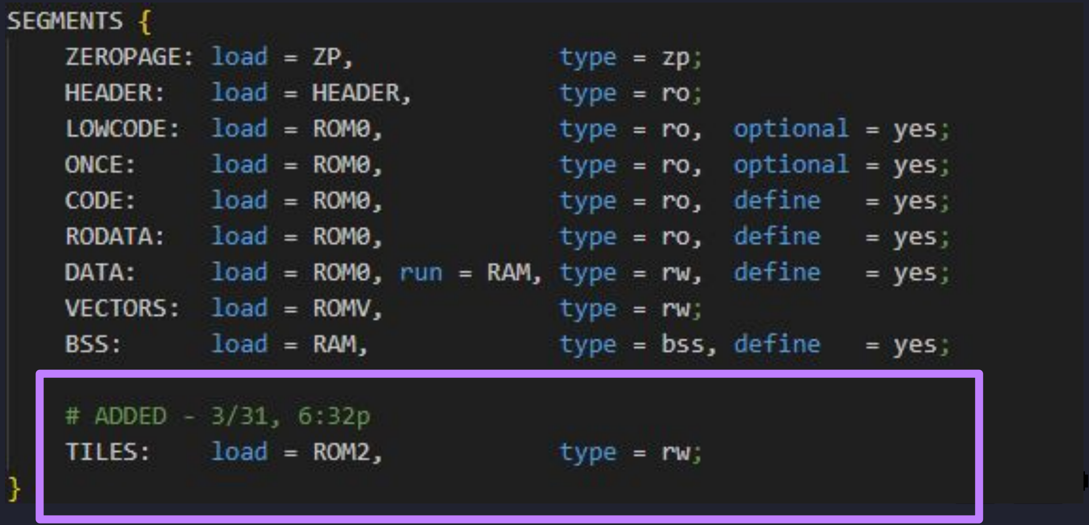
General Challenges & Solutions

1) Modifying NES memory is HARD!

Just disassembling & immediately reassembling a perfectly fine ROM was giving us glitched screens!

Solution?

Create our own config file and add a segment, this allowed us to actually START testing



```
SEGMENTS {
    ZEROPAGE: load = ZP,                  type = zp;
    HEADER:   load = HEADER,              type = ro;
    LOWCODE:  load = ROM0,                type = ro,  optional = yes;
    ONCE:     load = ROM0,                type = ro,  optional = yes;
    CODE:     load = ROM0,                type = ro,  define   = yes;
    RODATA:   load = ROM0,                type = ro,  define   = yes;
    DATA:     load = ROM0, run = RAM,     type = rw,  define   = yes;
    VECTORS:  load = ROMV,                type = rw;
    BSS:      load = RAM,                 type = bss, define   = yes;

    # ADDED - 3/31, 6:32p
    TILES:    load = ROM2,                type = rw;
}
```

General Challenges & Solutions

2) Modifying NES memory is STILL HARD!

Can't just add new functions and data in the middle of the ROM, it misaligns everything!

Solution?

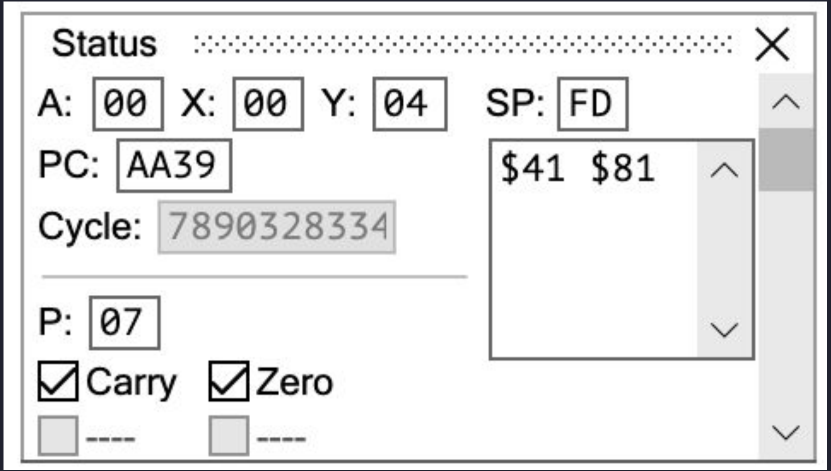
Add new data and functions to the ***end*** *of the ROM, remove padding bytes, and change references*

Misaligned…

General Challenges & Solutions

3) The NES is extremely limiting

- 8-bit processor
- 2 general purpose registers
- An accumulator register
- Stack is 256 bytes



```
Status
A: 00  X: 00  Y: 04    SP: FD
PC: AA39                $41 $81
Cycle: 7890328334
P: 07
[✓] Carry  [✓] Zero
[ ] ----   [ ] ----
```

Missing States

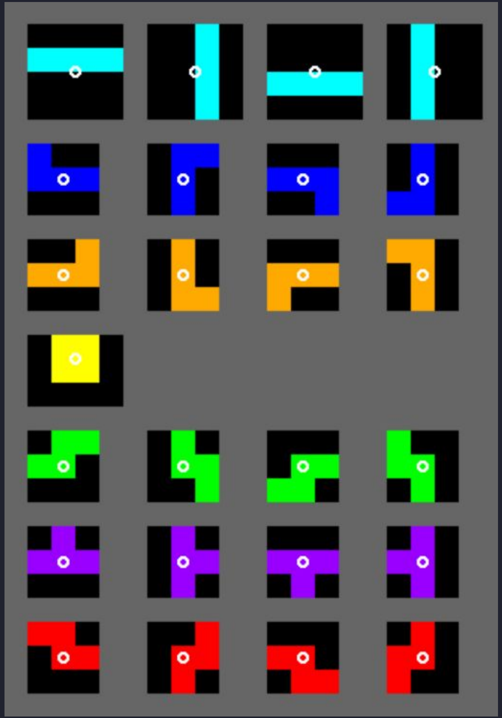
Missing States

Three Parts to Rotating Pieces in Tetris…

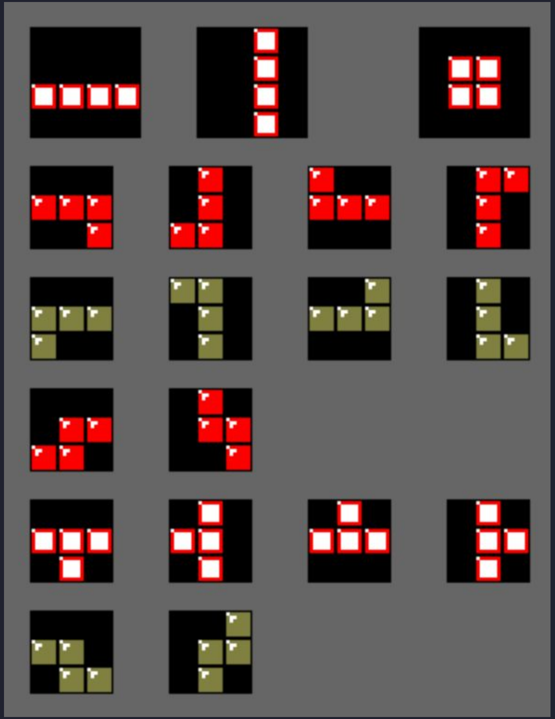
1) Find correct rotation state for that piece
2) Mapping that state to screen coordinates!
3) Spawn in the correct piece

Missing Rotation States

SRS

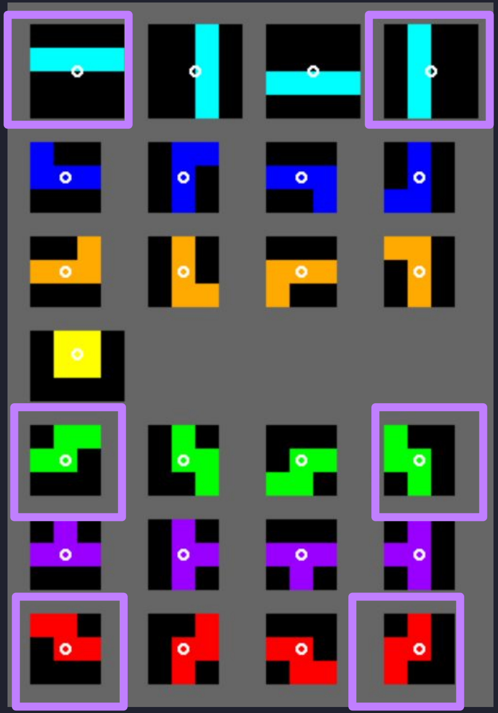


NES

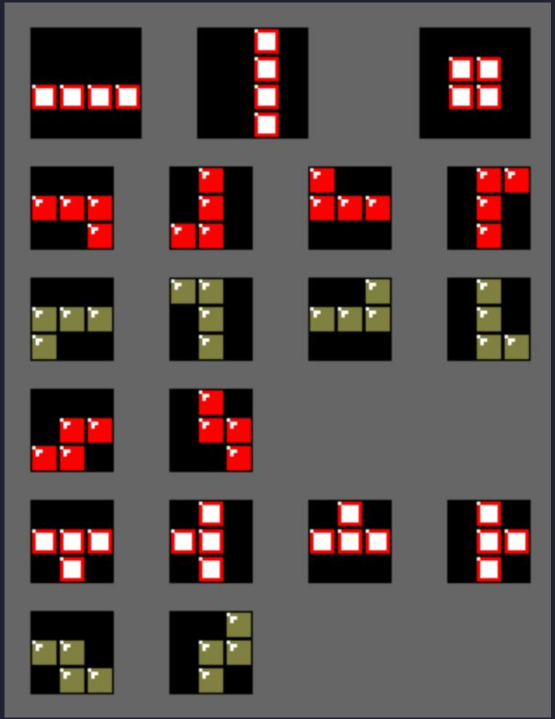


Missing Rotation States

SRS



NES

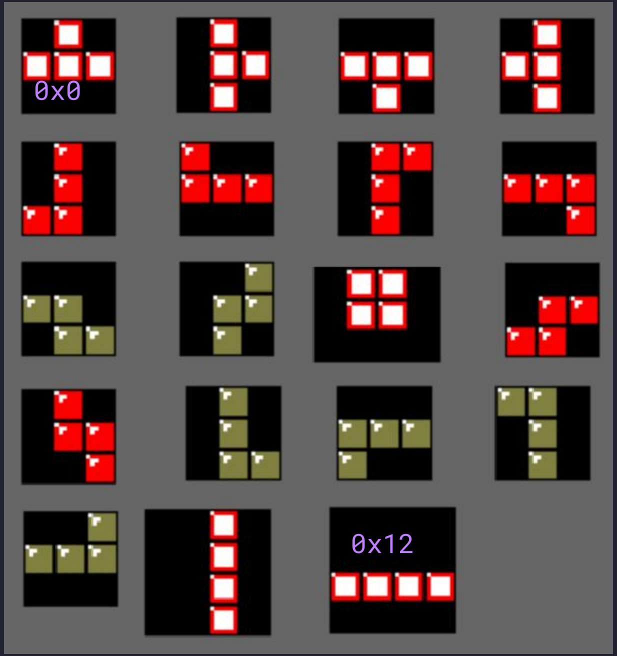


Missing:

- *2/4 I-pieces*
- *2/4 S-pieces*
- *2/4 Z-pieces*

Old Rotation States

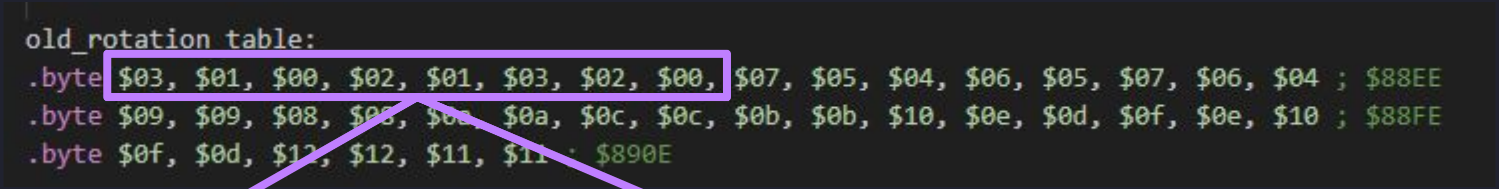
1 byte per rotation state (0x0-0x12)



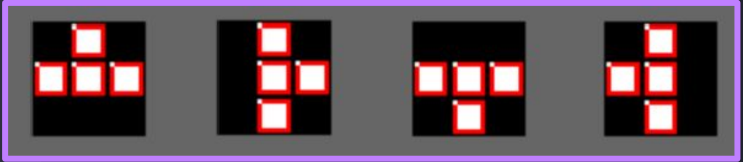
Increments from top-left…
(T-Block up)
to bottom right
(I-Block horizontal)

Old Rotation Table

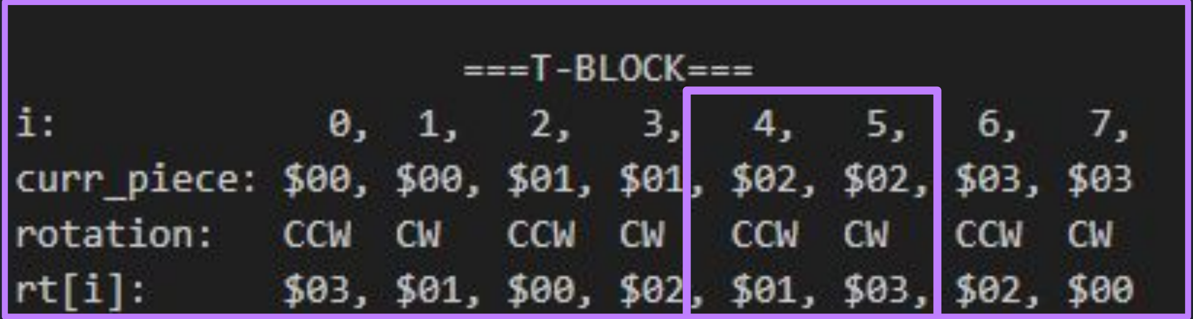
Maps button presses to state changes!



```
old_rotation table:
.byte $03, $01, $00, $02, $01, $03, $02, $00, $07, $05, $04, $06, $05, $07, $06, $04 ; $88EE
.byte $09, $09, $08, $08, $0a, $0a, $0c, $0c, $0b, $0b, $10, $0e, $0d, $0f, $0e, $10 ; $88FE
.byte $0f, $0d, $12, $12, $11, $11 ; $890E
```

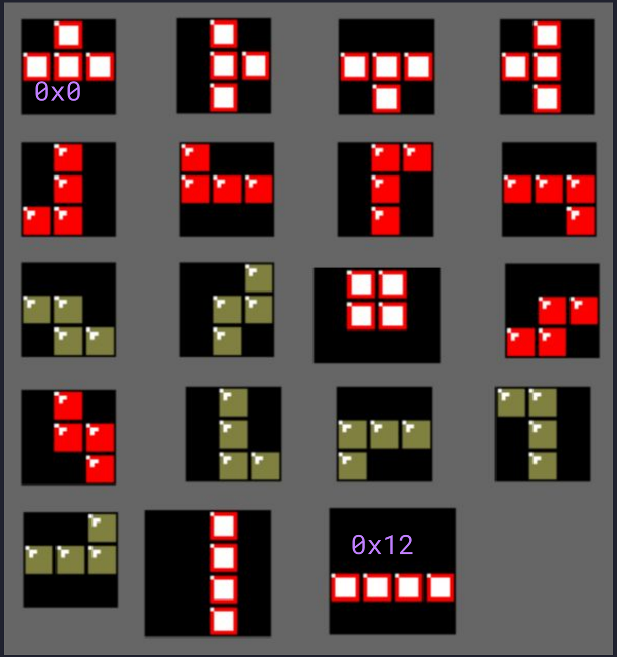


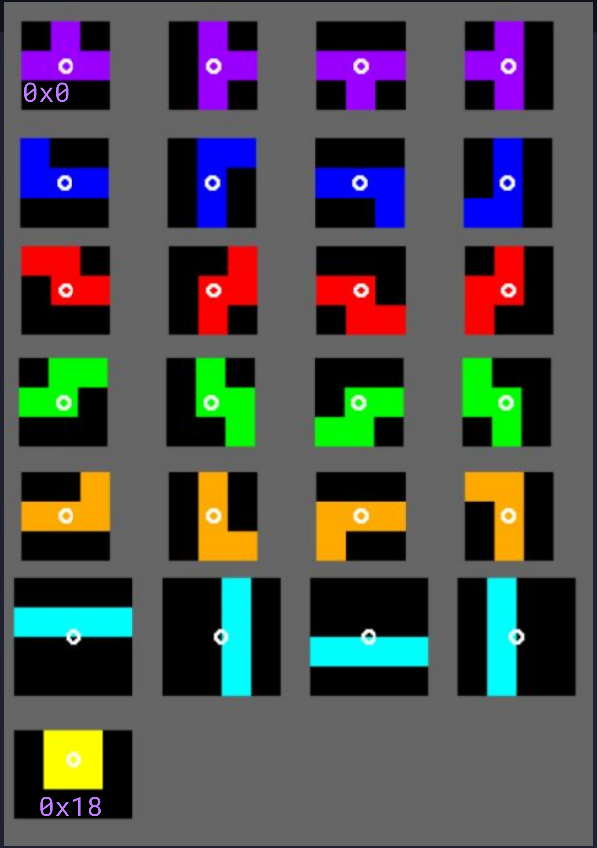
0x0 0x1 0x2 0x3



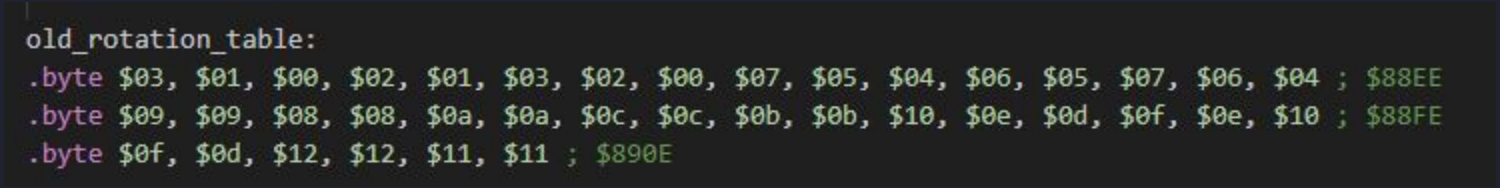
```
                        ===T-BLOCK===
i:             0,   1,   2,   3,   4,   5,   6,   7,
curr_piece: $00, $00, $01, $01, $02, $02, $03, $03
rotation:    CCW  CW   CCW  CW   CCW  CW   CCW  CW
rt[i]:      $03, $01, $00, $02, $01, $03, $02, $00
```

New Rotation States





New Rotation Table

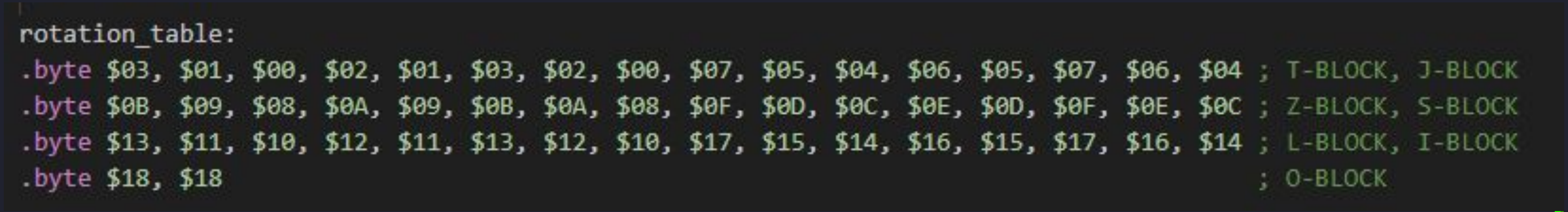


```
old_rotation_table:
.byte $03, $01, $00, $02, $01, $03, $02, $00, $07, $05, $04, $06, $05, $07, $06, $04 ; $88EE
.byte $09, $09, $08, $08, $0a, $0a, $0c, $0c, $0b, $0b, $10, $0e, $0d, $0f, $0e, $10 ; $88FE
.byte $0f, $0d, $12, $12, $11, $11 ; $890E
```

38 bytes

50 bytes

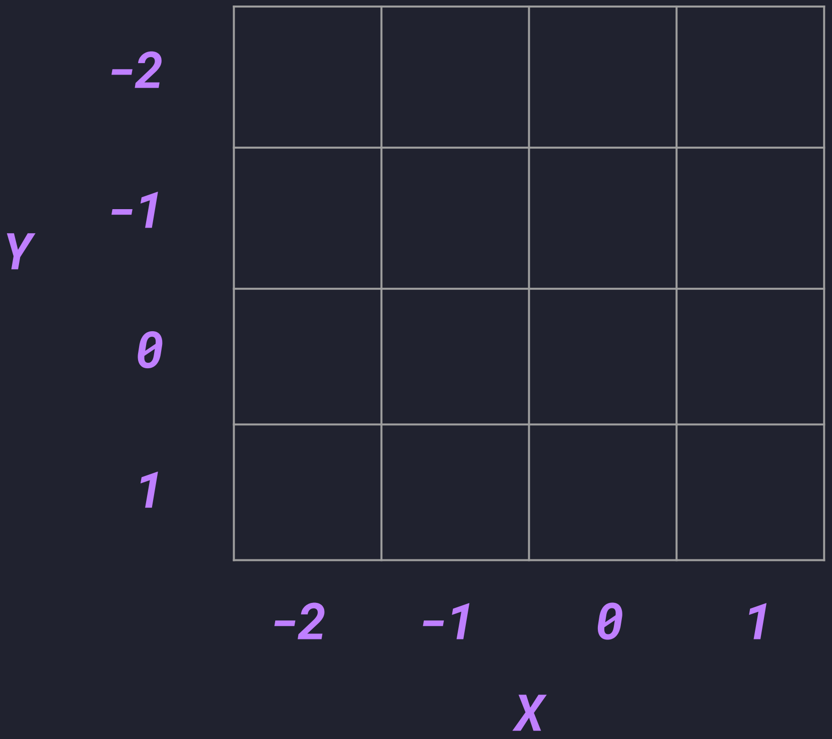
~32% size increase!

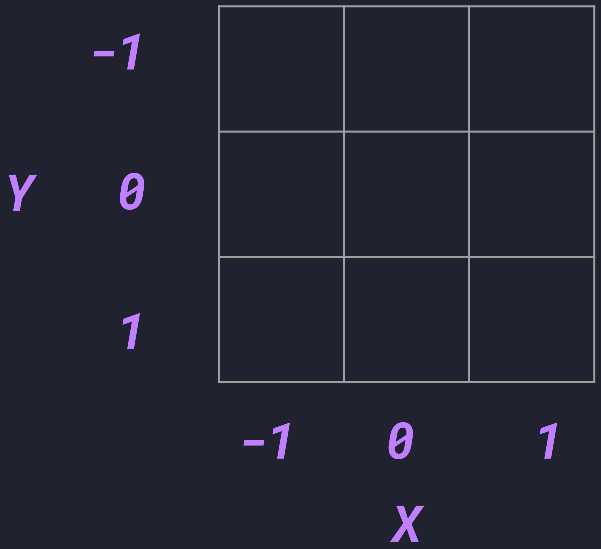


```
rotation_table:
.byte $03, $01, $00, $02, $01, $03, $02, $00, $07, $05, $04, $06, $05, $07, $06, $04 ; T-BLOCK, J-BLOCK
.byte $0B, $09, $08, $0A, $09, $0B, $0A, $08, $0F, $0D, $0C, $0E, $0D, $0F, $0E, $0C ; Z-BLOCK, S-BLOCK
.byte $13, $11, $10, $12, $11, $13, $12, $10, $17, $15, $14, $16, $15, $17, $16, $14 ; L-BLOCK, I-BLOCK
.byte $18, $18                                                                       ; O-BLOCK
```

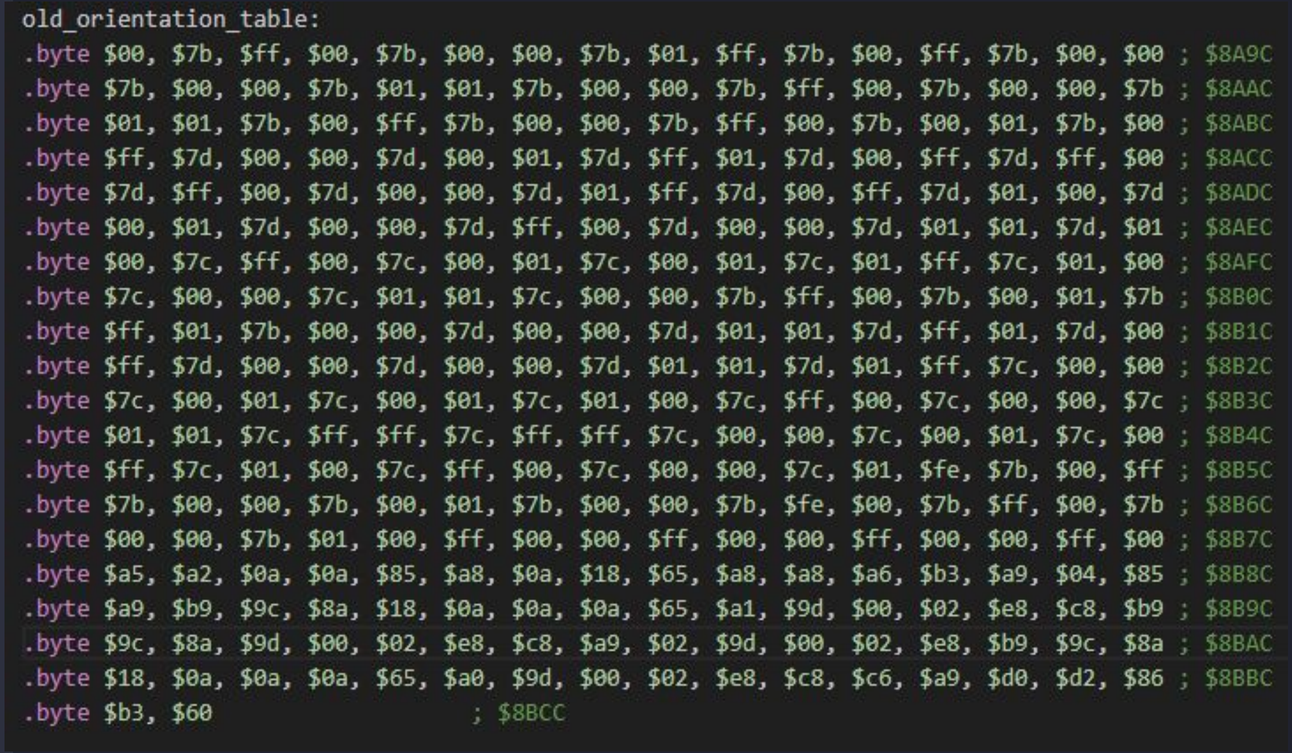
Graphics

Mapping to Graphics





Old Graphics Table



```
old_orientation_table:
.byte $00, $7b, $ff, $00, $7b, $00, $00, $7b, $01, $ff, $7b, $00, $ff, $7b, $00, $00 ; $8A9C
.byte $7b, $00, $00, $7b, $01, $01, $7b, $00, $00, $7b, $ff, $00, $7b, $00, $00, $7b ; $8AAC
.byte $01, $01, $7b, $00, $ff, $7b, $00, $00, $7b, $ff, $00, $7b, $00, $01, $7b, $00 ; $8ABC
.byte $ff, $7d, $00, $00, $7d, $00, $01, $7d, $ff, $01, $7d, $00, $ff, $7d, $ff, $00 ; $8ACC
.byte $7d, $ff, $00, $7d, $00, $00, $7d, $01, $ff, $7d, $00, $ff, $7d, $01, $00, $7d ; $8ADC
.byte $00, $01, $7d, $00, $00, $7d, $ff, $00, $7d, $00, $00, $7d, $01, $01, $7d, $01 ; $8AEC
.byte $00, $7c, $ff, $00, $7c, $00, $01, $7c, $00, $01, $7c, $01, $ff, $7c, $01, $00 ; $8AFC
.byte $7c, $00, $00, $7c, $01, $01, $7c, $00, $00, $7b, $ff, $00, $7b, $00, $01, $7b ; $8B0C
.byte $ff, $01, $7b, $00, $00, $7d, $00, $00, $7d, $01, $01, $7d, $ff, $01, $7d, $00 ; $8B1C
.byte $ff, $7d, $00, $00, $7d, $00, $00, $7d, $01, $01, $7d, $01, $ff, $7c, $00, $00 ; $8B2C
.byte $7c, $00, $01, $7c, $00, $01, $7c, $01, $00, $7c, $ff, $00, $7c, $00, $00, $7c ; $8B3C
.byte $01, $01, $7c, $ff, $ff, $7c, $ff, $ff, $7c, $00, $00, $7c, $00, $01, $7c, $00 ; $8B4C
.byte $ff, $7c, $01, $00, $7c, $ff, $00, $7c, $00, $00, $7c, $01, $fe, $7b, $00, $ff ; $8B5C
.byte $7b, $00, $00, $7b, $00, $01, $7b, $00, $00, $7b, $fe, $00, $7b, $ff, $00, $7b ; $8B6C
.byte $00, $00, $7b, $01, $00, $ff, $00, $00, $ff, $00, $00, $ff, $00, $00, $ff, $00 ; $8B7C
.byte $a5, $a2, $0a, $0a, $85, $a8, $0a, $18, $65, $a8, $a8, $a6, $b3, $a9, $04, $85 ; $8B8C
.byte $a9, $b9, $9c, $8a, $18, $0a, $0a, $0a, $65, $a1, $9d, $00, $02, $e8, $c8, $b9 ; $8B9C
.byte $9c, $8a, $9d, $00, $02, $e8, $c8, $a9, $02, $9d, $00, $02, $e8, $b9, $9c, $8a ; $8BAC
.byte $18, $0a, $0a, $0a, $65, $a0, $9d, $00, $02, $e8, $c8, $c6, $a9, $d0, $d2, $86 ; $8BBC
.byte $b3, $60                              ; $8BCC
```

4 sets per Tetromino:

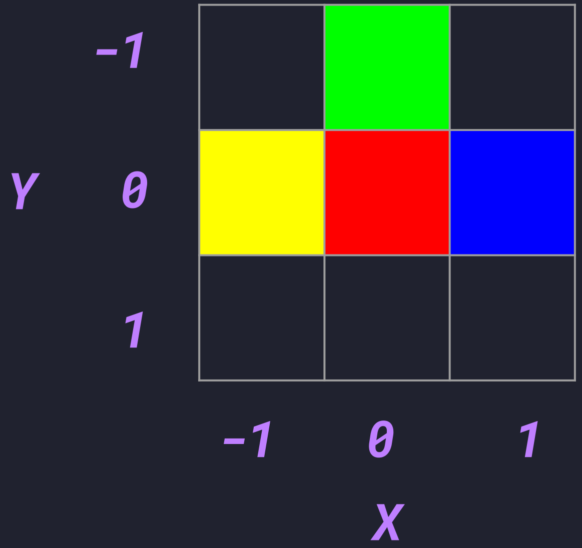
- 1 per block

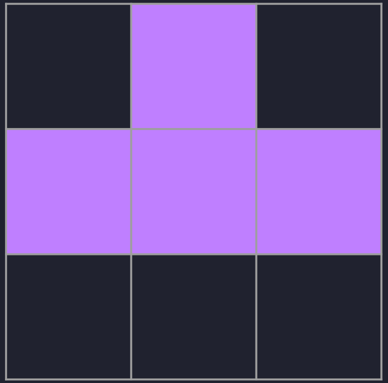
Each set:

- (Y, TILE, X)

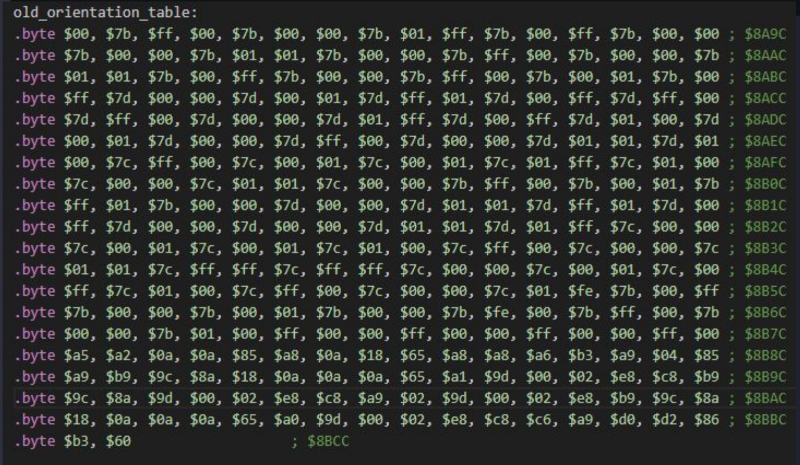
Old Graphics Table

```
T-Tetromino Up:
    (Y, TILE, X)
    (0, 123, -1)
    (0, 123, 0)
    (0, 123, 1)
    (-1, 123, 0)
```





New Graphics Table

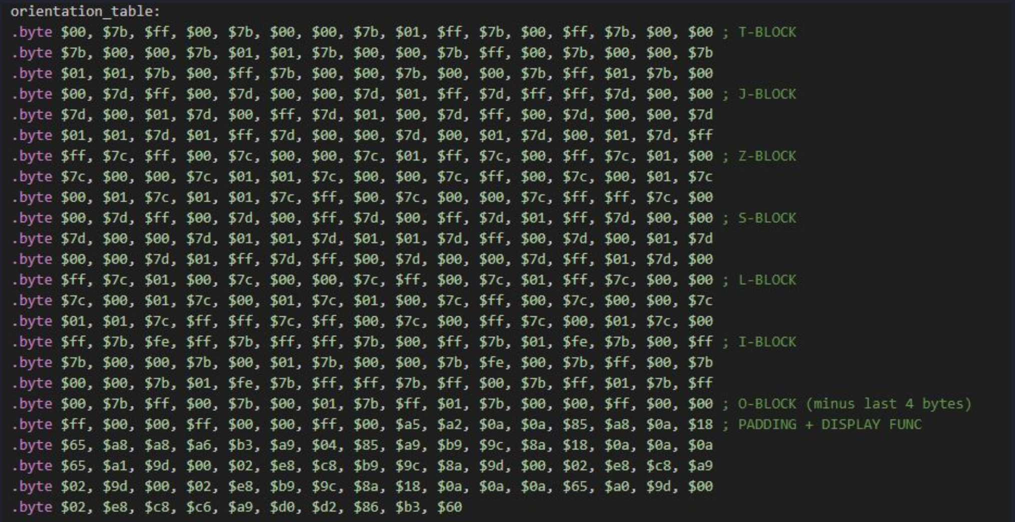


```
old_orientation_table:
.byte $00, $7b, $ff, $00, $7b, $00, $00, $7b, $01, $ff, $7b, $00, $ff, $7b, $00, $00 ; $8A9C
.byte $7b, $00, $00, $7b, $01, $01, $7b, $00, $00, $7b, $ff, $00, $7b, $00, $00, $7b ; $8AAC
.byte $01, $01, $7b, $00, $ff, $7b, $00, $00, $7b, $ff, $00, $7b, $00, $01, $7b, $00 ; $8ABC
.byte $ff, $7d, $00, $00, $7d, $00, $01, $7d, $ff, $01, $7d, $00, $ff, $7d, $ff, $00 ; $8ACC
.byte $7d, $ff, $00, $7d, $00, $00, $7d, $01, $ff, $7d, $00, $ff, $7d, $01, $00, $7d ; $8ADC
.byte $00, $01, $7d, $00, $00, $7d, $ff, $00, $7d, $00, $00, $7d, $01, $01, $7d, $01 ; $8AEC
.byte $00, $7c, $ff, $00, $7c, $00, $01, $7c, $00, $01, $7c, $01, $ff, $7c, $01, $00 ; $8AFC
.byte $7c, $00, $00, $7c, $01, $01, $7c, $00, $00, $7b, $ff, $00, $7b, $00, $01, $7b ; $8B0C
.byte $ff, $01, $7b, $00, $00, $7d, $00, $00, $7d, $01, $01, $7d, $ff, $01, $7d, $00 ; $8B1C
.byte $ff, $7d, $00, $00, $7d, $00, $00, $7d, $01, $01, $7d, $01, $ff, $7c, $00, $00 ; $8B2C
.byte $7c, $00, $01, $7c, $00, $01, $7c, $01, $00, $7c, $ff, $00, $7c, $00, $00, $7c ; $8B3C
.byte $01, $01, $7c, $ff, $ff, $7c, $ff, $ff, $7c, $00, $00, $7c, $00, $01, $7c, $00 ; $8B4C
.byte $ff, $7c, $01, $00, $7c, $ff, $00, $7c, $00, $00, $7c, $01, $fe, $7b, $00, $ff ; $8B5C
.byte $7b, $00, $00, $7b, $00, $01, $7b, $00, $00, $7b, $fe, $00, $7b, $ff, $00, $7b ; $8B6C
.byte $00, $00, $7b, $01, $00, $ff, $00, $00, $ff, $00, $00, $ff, $00, $00, $ff, $00 ; $8B7C
.byte $a5, $a2, $0a, $0a, $85, $a8, $0a, $18, $65, $a8, $a8, $a6, $b3, $a9, $04, $85 ; $8B8C
.byte $a9, $b9, $9c, $8a, $18, $0a, $0a, $0a, $65, $a1, $9d, $00, $02, $e8, $c8, $b9 ; $8B9C
.byte $9c, $8a, $9d, $00, $02, $e8, $c8, $a9, $02, $9d, $00, $02, $e8, $b9, $9c, $8a ; $8BAC
.byte $18, $0a, $0a, $0a, $65, $a0, $9d, $00, $02, $e8, $c8, $c6, $a9, $d0, $d2, $86 ; $8BBC
.byte $b3, $60                            ; $8BCC
```

306 bytes

378 bytes

~23% size increase!



```
orientation_table:
.byte $00, $7b, $ff, $00, $7b, $00, $00, $7b, $01, $ff, $7b, $00, $ff, $7b, $00, $00 ; T-BLOCK
.byte $7b, $00, $00, $7b, $01, $01, $7b, $00, $00, $7b, $ff, $00, $7b, $00, $00, $7b
.byte $01, $01, $7b, $00, $ff, $7b, $00, $00, $7b, $00, $00, $7b, $ff, $01, $7b, $00
.byte $00, $7d, $ff, $00, $7d, $00, $00, $7d, $01, $ff, $7d, $ff, $ff, $7d, $00, $00 ; J-BLOCK
.byte $7d, $00, $01, $7d, $00, $ff, $7d, $01, $00, $7d, $ff, $00, $7d, $00, $00, $7d
.byte $01, $01, $7d, $01, $ff, $7d, $00, $00, $7d, $00, $01, $7d, $00, $01, $7d, $ff
.byte $ff, $7c, $ff, $00, $7c, $00, $00, $7c, $01, $ff, $7c, $00, $ff, $7c, $01, $00 ; Z-BLOCK
.byte $7c, $00, $00, $7c, $01, $01, $7c, $00, $00, $7c, $ff, $00, $7c, $00, $01, $7c
.byte $00, $01, $7c, $01, $01, $7c, $ff, $00, $7c, $00, $00, $7c, $ff, $ff, $7c, $00
.byte $00, $7d, $ff, $00, $7d, $00, $ff, $7d, $00, $ff, $7d, $01, $ff, $7d, $00, $00 ; S-BLOCK
.byte $7d, $00, $00, $7d, $01, $01, $7d, $01, $01, $7d, $ff, $00, $7d, $00, $01, $7d
.byte $00, $00, $7d, $01, $ff, $7d, $ff, $00, $7d, $00, $00, $7d, $ff, $01, $7d, $00
.byte $ff, $7c, $01, $00, $7c, $00, $00, $7c, $ff, $00, $7c, $01, $ff, $7c, $00, $00 ; L-BLOCK
.byte $7c, $00, $01, $7c, $00, $01, $7c, $01, $00, $7c, $ff, $00, $7c, $00, $00, $7c
.byte $01, $01, $7c, $ff, $ff, $7c, $ff, $00, $7c, $00, $ff, $7c, $00, $01, $7c, $00
.byte $ff, $7b, $fe, $ff, $7b, $ff, $ff, $7b, $00, $ff, $7b, $01, $fe, $7b, $00, $ff ; I-BLOCK
.byte $7b, $00, $00, $7b, $00, $01, $7b, $00, $00, $7b, $fe, $00, $7b, $ff, $00, $7b
.byte $00, $00, $7b, $01, $fe, $7b, $ff, $ff, $7b, $ff, $00, $7b, $ff, $01, $7b, $ff
.byte $00, $7b, $ff, $00, $7b, $00, $01, $7b, $ff, $01, $7b, $00, $00, $ff, $00, $00 ; O-BLOCK (minus last 4 bytes)
.byte $ff, $00, $00, $ff, $00, $00, $ff, $00, $a5, $a2, $0a, $0a, $85, $a8, $0a, $18 ; PADDING + DISPLAY FUNC
.byte $65, $a8, $a8, $a6, $b3, $a9, $04, $85, $a9, $b9, $9c, $8a, $18, $0a, $0a, $0a
.byte $65, $a1, $9d, $00, $02, $e8, $c8, $b9, $9c, $8a, $9d, $00, $02, $e8, $c8, $a9
.byte $02, $9d, $00, $02, $e8, $b9, $9c, $8a, $18, $0a, $0a, $0a, $65, $a0, $9d, $00
.byte $02, $e8, $c8, $c6, $a9, $d0, $d2, $86, $b3, $60
```

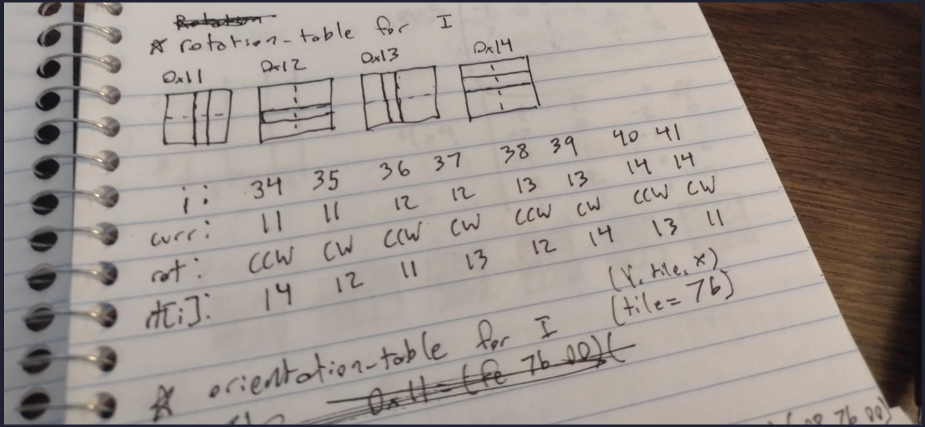
States & Graphics: Challenges & Solutions

1) Tables are relative to each other in terms of block ordering...

This meant that any changes to the layout of *one* table meant it had to be reflected to the other.

Solution?

Just be careful and keep copious records, try things by hand!



~~Rotation~~
☆ rotation-table for I

0x11
0x12
0x13
0x14

i:	34	35	36	37	38	39	40	41
curr:	11	11	12	12	13	13	14	14
rot:	CCW	CW	CCW	CW	CCW	CW	CCW	CW
rt[i]:	14	12	11	13	12	14	13	11

(Y, tile, x)
(tile = 76)

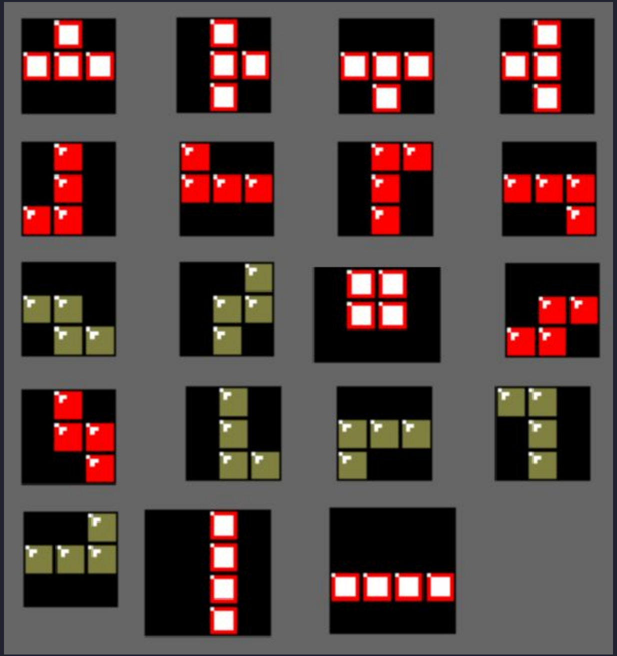
☆ orientation-table for I
~~0x11 = (fe 76 00)(~~

States & Graphics: Challenges & Solutions

2) O-Block was ANNOYING!

See how it shifts the picture? That was happening in the code too, it meant easily typing between rows and columns was VERY tedious & easily led to errors

Solution? Though tedious, moving it to the END of BOTH tables was best

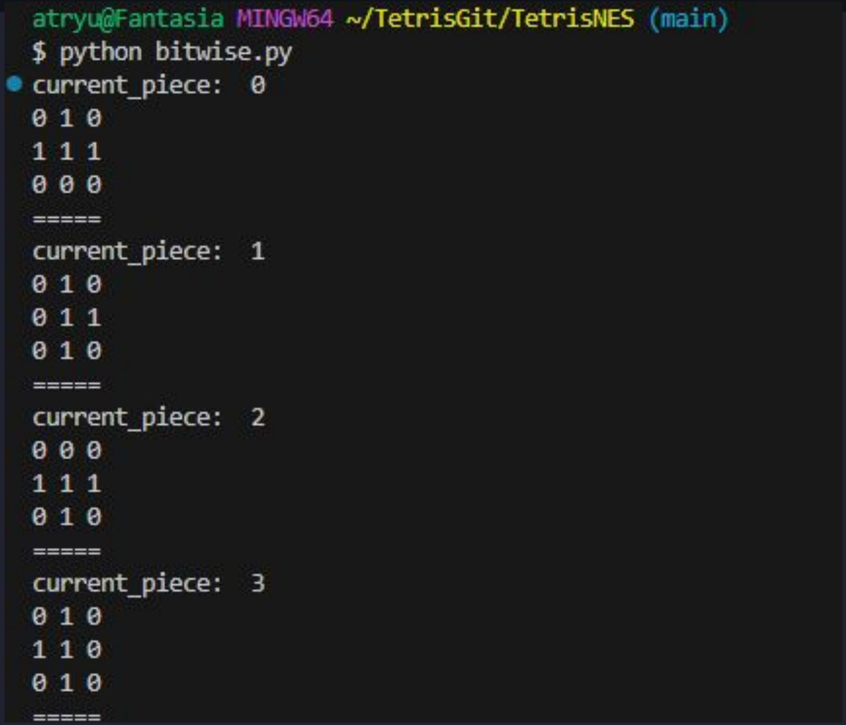


States & Graphics: Challenges & Solutions

3) Very difficult to read graphics tables

Reading incorrectly as (X,TILE,Y) was often, also very difficult to see if the new positions were correct until we playtested!

Solution? Python code that translates bytes into displays



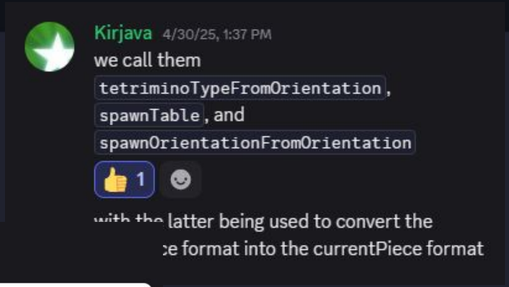
```
atryu@Fantasia MINGW64 ~/TetrisGit/TetrisNES (main)
$ python bitwise.py
current_piece:  0
0 1 0
1 1 1
0 0 0
=====
current_piece:  1
0 1 0
0 1 1
0 1 0
=====
current_piece:  2
0 0 0
1 1 1
0 1 0
=====
current_piece:  3
0 1 0
1 1 0
0 1 0
=====
```

States & Graphics: Challenges & Solutions

4) Decompiled assembly is just hard to read

Many moving pieces + no NES dev. experience meant lots of dead-ends

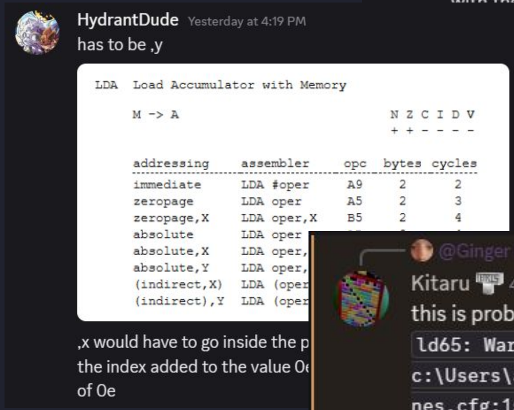
Solution? ROM Hackers who helped point us in the right way! Also helped with earlier memory problems



Kirjava 4/30/25, 1:37 PM
we call them `tetriminoTypeFromOrientation`, `spawnTable`, and `spawnOrientationFromOrientation`

👍 1

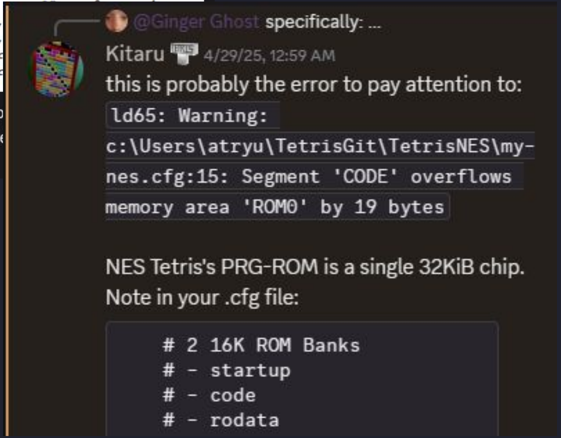
with the latter being used to convert the
ce format into the currentPiece format



HydrantDude Yesterday at 4:19 PM
has to be ,y

```
LDA  Load Accumulator with Memory

     M -> A                                   N Z C I D V
                                              + + - - - -

     addressing      assembler      opc  bytes  cycles
     --------------------------------------------------
     immediate       LDA #oper      A9    2      2
     zeropage        LDA oper       A5    2      3
     zeropage,X      LDA oper,X     B5    2      4
     absolute        LDA oper
     absolute,X      LDA oper,
     absolute,Y      LDA oper,
     (indirect,X)    LDA (oper
     (indirect),Y    LDA (oper
```

,x would have to go inside the p
the index added to the value 0e
of 0e



@Ginger Ghost specifically: ...
Kitaru 4/29/25, 12:59 AM
this is probably the error to pay attention to:
`ld65: Warning: c:\Users\atryu\TetrisGit\TetrisNES\my-nes.cfg:15: Segment 'CODE' overflows memory area 'ROM0' by 19 bytes`

NES Tetris's PRG-ROM is a single 32KiB chip.
Note in your .cfg file:

```
    # 2 16K ROM Banks
    # - startup
    # - code
    # - rodata
```

Spawning

Spawning the new piece

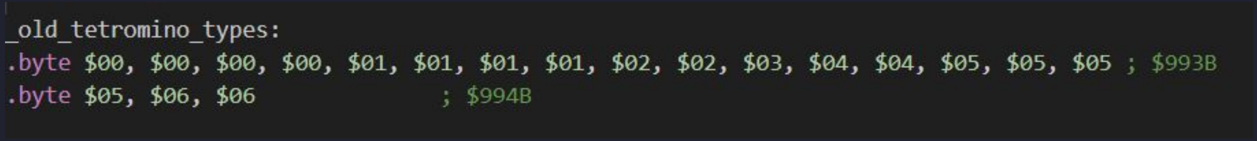
Game now knows that

a) There are more rotation states!
b) How to draw them!

Final part?

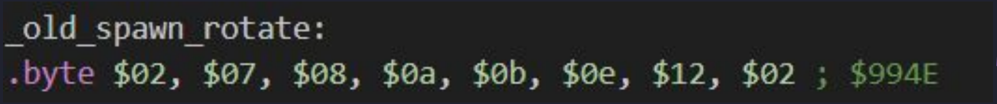
c) How do we spawn them in?

Tables used for Spawning



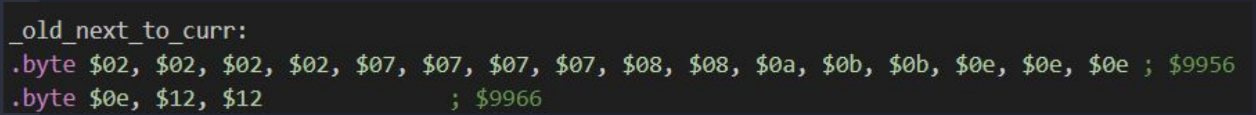
```
_old_tetromino_types:
.byte $00, $00, $00, $00, $01, $01, $01, $01, $02, $02, $03, $04, $04, $05, $05, $05 ; $993B
.byte $05, $06, $06                  ; $994B
```

Categorizes IDs into 7 tetrominoes



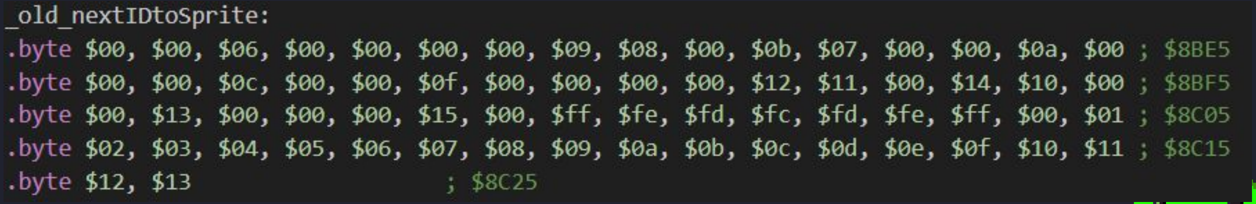
```
_old_spawn_rotate:
.byte $02, $07, $08, $0a, $0b, $0e, $12, $02 ; $994E
```

Spawn rotation for each tetromino type



```
_old_next_to_curr:
.byte $02, $02, $02, $02, $07, $07, $07, $07, $08, $08, $0a, $0b, $0b, $0e, $0e, $0e ; $9956
.byte $0e, $12, $12                  ; $9966
```

Spawn rotation for each ID

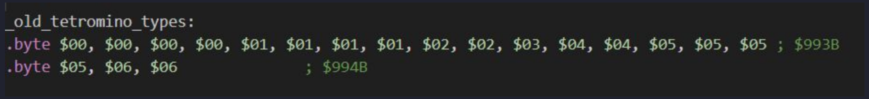


```
_old_nextIDtoSprite:
.byte $00, $00, $06, $00, $00, $00, $00, $09, $08, $00, $0b, $07, $00, $00, $0a, $00 ; $8BE5
.byte $00, $00, $0c, $00, $00, $0f, $00, $00, $00, $00, $12, $11, $00, $14, $10, $00 ; $8BF5
.byte $00, $13, $00, $00, $00, $15, $00, $ff, $fe, $fd, $fc, $fd, $fe, $ff, $00, $01 ; $8C05
.byte $02, $03, $04, $05, $06, $07, $08, $09, $0a, $0b, $0c, $0d, $0e, $0f, $10, $11 ; $8C15
.byte $12, $13                       ; $8C25
```

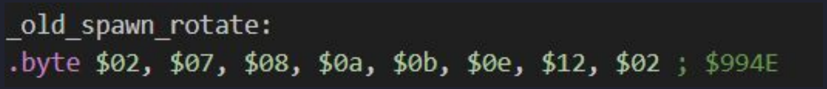
Sprite ID for next window ($00 = don't display)

Tables used for Spawning

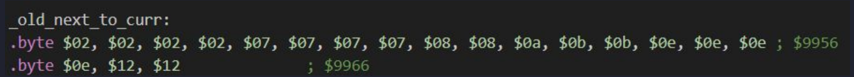
Before



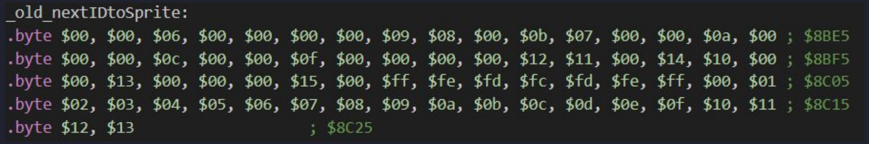
```
_old_tetromino_types:
.byte $00, $00, $00, $00, $01, $01, $01, $01, $02, $02, $03, $04, $04, $05, $05, $05 ; $993B
.byte $05, $06, $06                            ; $994B
```



```
_old_spawn_rotate:
.byte $02, $07, $08, $0a, $0b, $0e, $12, $02 ; $994E
```

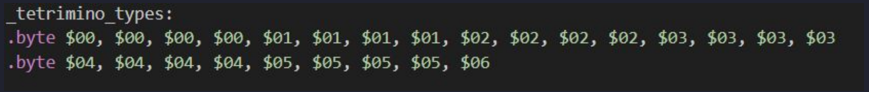


```
_old_next_to_curr:
.byte $02, $02, $02, $02, $07, $07, $07, $07, $08, $08, $0a, $0b, $0b, $0e, $0e, $0e ; $9956
.byte $0e, $12, $12                            ; $9966
```

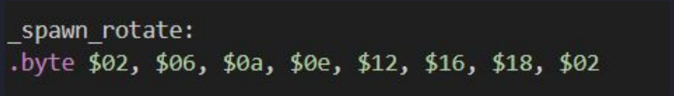


```
_old_nextIDtoSprite:
.byte $00, $00, $06, $00, $00, $00, $00, $09, $08, $00, $0b, $07, $00, $00, $0a, $00 ; $8BE5
.byte $00, $00, $0c, $00, $00, $0f, $00, $00, $00, $00, $12, $11, $00, $14, $10, $00 ; $8BF5
.byte $00, $13, $00, $00, $00, $15, $00, $ff, $fe, $fd, $fc, $fd, $fe, $ff, $00, $01 ; $8C05
.byte $02, $03, $04, $05, $06, $07, $08, $09, $0a, $0b, $0c, $0d, $0e, $0f, $10, $11 ; $8C15
.byte $12, $13                                 ; $8C25
```

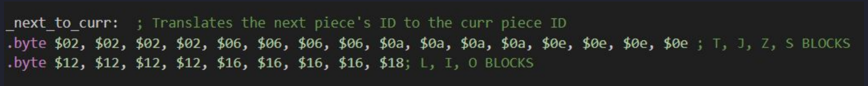
After



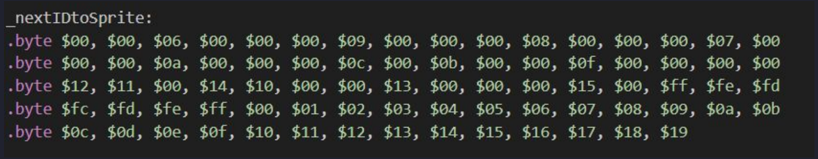
```
_tetrimino_types:
.byte $00, $00, $00, $00, $01, $01, $01, $01, $02, $02, $02, $02, $03, $03, $03, $03
.byte $04, $04, $04, $04, $05, $05, $05, $05, $06
```



```
_spawn_rotate:
.byte $02, $06, $0a, $0e, $12, $16, $18, $02
```



```
_next_to_curr:  ; Translates the next piece's ID to the curr piece ID
.byte $02, $02, $02, $02, $06, $06, $06, $06, $0a, $0a, $0a, $0a, $0e, $0e, $0e, $0e ; T, J, Z, S BLOCKS
.byte $12, $12, $12, $12, $16, $16, $16, $16, $18; L, I, O BLOCKS
```



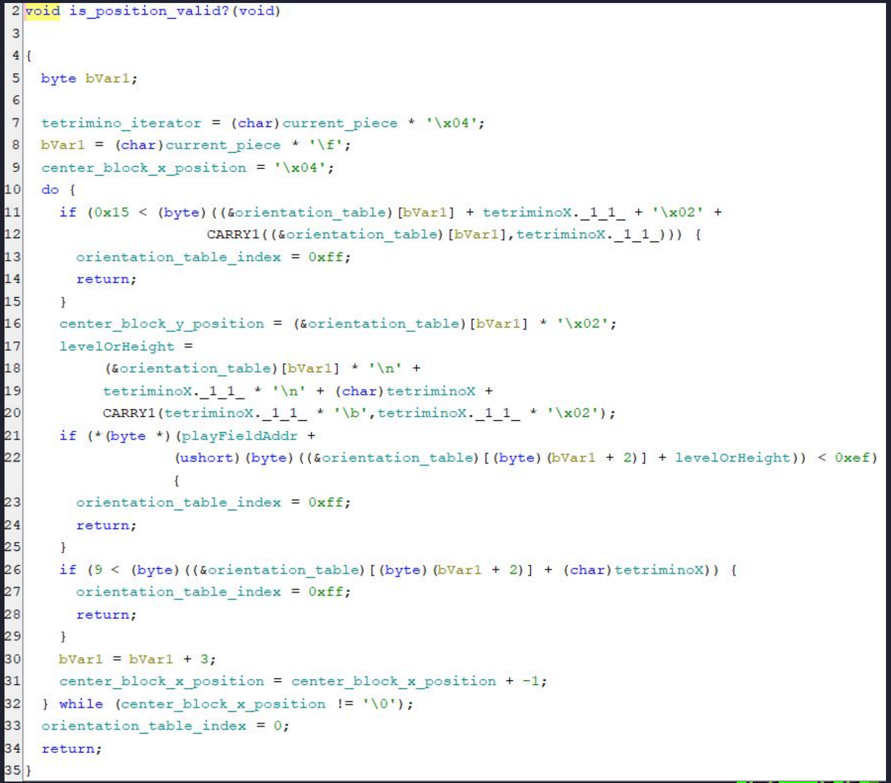
```
_nextIDtoSprite:
.byte $00, $00, $06, $00, $00, $00, $09, $00, $00, $00, $08, $00, $00, $00, $07, $00
.byte $00, $00, $0a, $00, $00, $00, $0c, $00, $0b, $00, $00, $0f, $00, $00, $00, $00
.byte $12, $11, $00, $14, $10, $00, $00, $13, $00, $00, $00, $15, $00, $ff, $fe, $fd
.byte $fc, $fd, $fe, $ff, $00, $01, $02, $03, $04, $05, $06, $07, $08, $09, $0a, $0b
.byte $0c, $0d, $0e, $0f, $10, $11, $12, $13, $14, $15, $16, $17, $18, $19
```

We’re done right??

Spawning: The Challenge

is_position_valid()

- *Checks if we can move piece to a position*
- *Iterates over all 4 blocks in a piece using orientation_table*
- *Checks each (X, Y) in orientation_table*

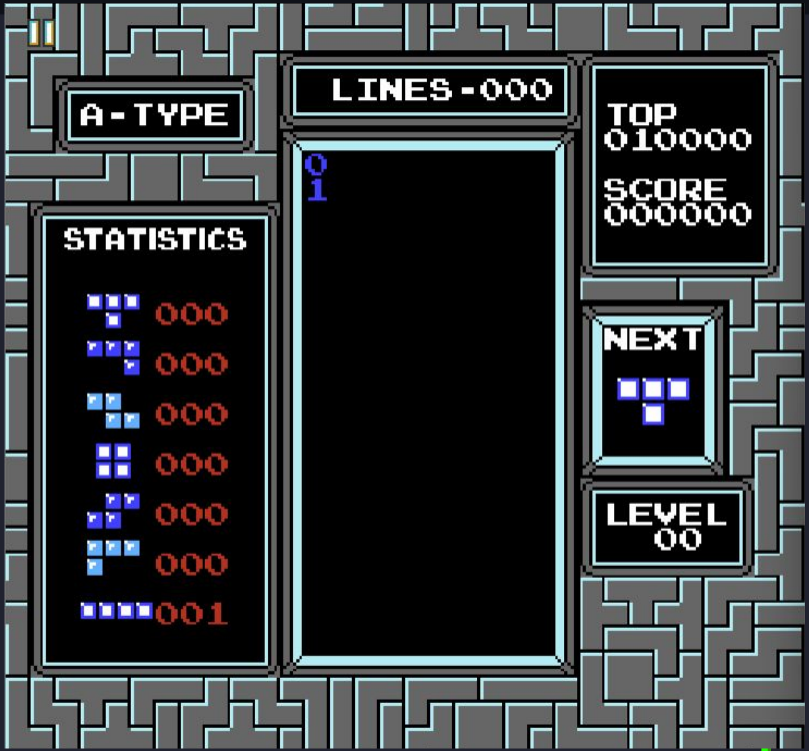


```
void is_position_valid?(void)


{
  byte bVar1;

  tetrimino_iterator = (char)current_piece * '\x04';
  bVar1 = (char)current_piece * '\f';
  center_block_x_position = '\x04';
  do {
    if (0x15 < (byte)((&orientation_table)[bVar1] + tetriminoX._1_1_ + '\x02' +
                     CARRY1((&orientation_table)[bVar1],tetriminoX._1_1_))) {
      orientation_table_index = 0xff;
      return;
    }
    center_block_y_position = (&orientation_table)[bVar1] * '\x02';
    levelOrHeight =
         (&orientation_table)[bVar1] * '\n' +
         tetriminoX._1_1_ * '\n' + (char)tetriminoX +
         CARRY1(tetriminoX._1_1_ * '\b',tetriminoX._1_1_ * '\x02');
    if (*(byte *)(playFieldAddr +
                 (ushort)(byte)((&orientation_table)[(byte)(bVar1 + 2)] + levelOrHeight)) < 0xef)
                 {
      orientation_table_index = 0xff;
      return;
    }
    if (9 < (byte)((&orientation_table)[(byte)(bVar1 + 2)] + (char)tetriminoX)) {
      orientation_table_index = 0xff;
      return;
    }
    bVar1 = bVar1 + 3;
    center_block_x_position = center_block_x_position + -1;
  } while (center_block_x_position != '\0');
  orientation_table_index = 0;
  return;
}
```

Spawning: The Challenge

`is_position_valid()`

- *Uses byte as index into table*
- *Table now has >255 bytes of data!!*
- *Overflows lead to game ejecting us from level when spawning pieces with IDs > 0x14*



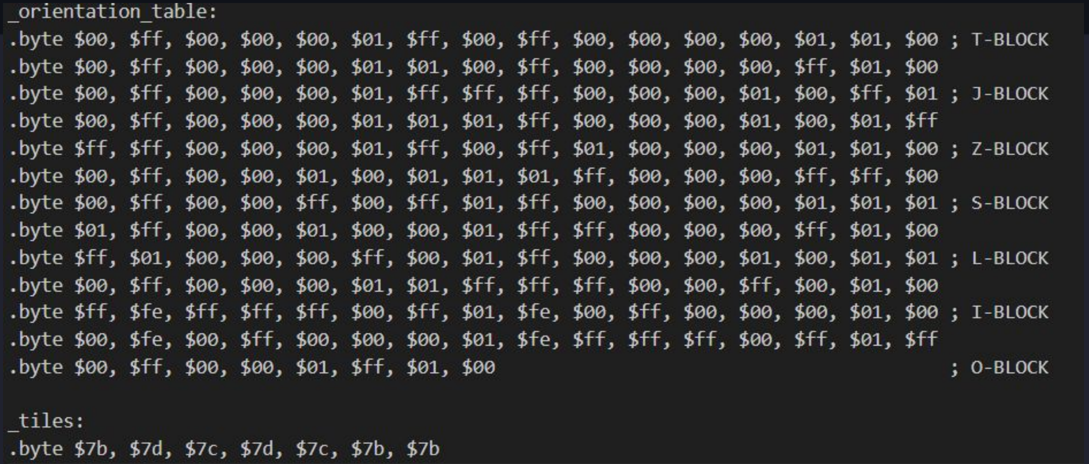
Spawning: Attempted Solutions

New graphics table

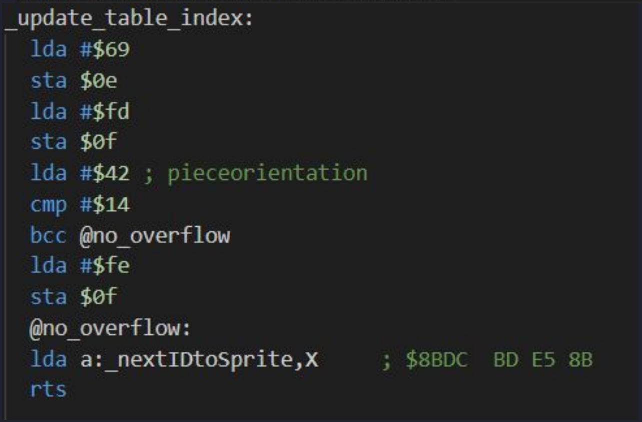
- Removes repeat sprite ID bytes
- Saves enough data!

Use offsets

- Pull from &table+255 if ID > 0x14
- Much simpler!



```
_orientation_table:
.byte $00, $ff, $00, $00, $00, $01, $ff, $00, $ff, $00, $00, $00, $00, $01, $01, $00 ; T-BLOCK
.byte $00, $ff, $00, $00, $00, $01, $01, $00, $ff, $00, $00, $00, $00, $ff, $01, $00
.byte $00, $ff, $00, $00, $00, $01, $ff, $ff, $ff, $00, $00, $00, $01, $00, $ff, $01 ; J-BLOCK
.byte $00, $ff, $00, $00, $00, $01, $01, $01, $ff, $00, $00, $00, $01, $00, $01, $ff
.byte $ff, $ff, $00, $00, $00, $01, $ff, $00, $ff, $01, $00, $00, $00, $01, $01, $00 ; Z-BLOCK
.byte $00, $ff, $00, $00, $01, $00, $01, $01, $01, $ff, $00, $00, $00, $ff, $ff, $00
.byte $00, $ff, $00, $00, $ff, $00, $ff, $01, $ff, $00, $00, $00, $00, $01, $01, $01 ; S-BLOCK
.byte $01, $ff, $00, $00, $01, $00, $00, $01, $ff, $ff, $00, $00, $00, $ff, $01, $00
.byte $ff, $01, $00, $00, $00, $ff, $00, $01, $ff, $00, $00, $00, $01, $00, $01, $01 ; L-BLOCK
.byte $00, $ff, $00, $00, $00, $01, $01, $ff, $ff, $ff, $00, $00, $ff, $00, $01, $00
.byte $ff, $fe, $ff, $ff, $ff, $00, $ff, $01, $fe, $00, $ff, $00, $00, $00, $01, $00 ; I-BLOCK
.byte $00, $fe, $00, $ff, $00, $00, $00, $01, $fe, $ff, $ff, $ff, $00, $ff, $01, $ff
.byte $00, $ff, $00, $00, $01, $ff, $01, $00                                         ; O-BLOCK

_tiles:
.byte $7b, $7d, $7c, $7d, $7c, $7b, $7b
```

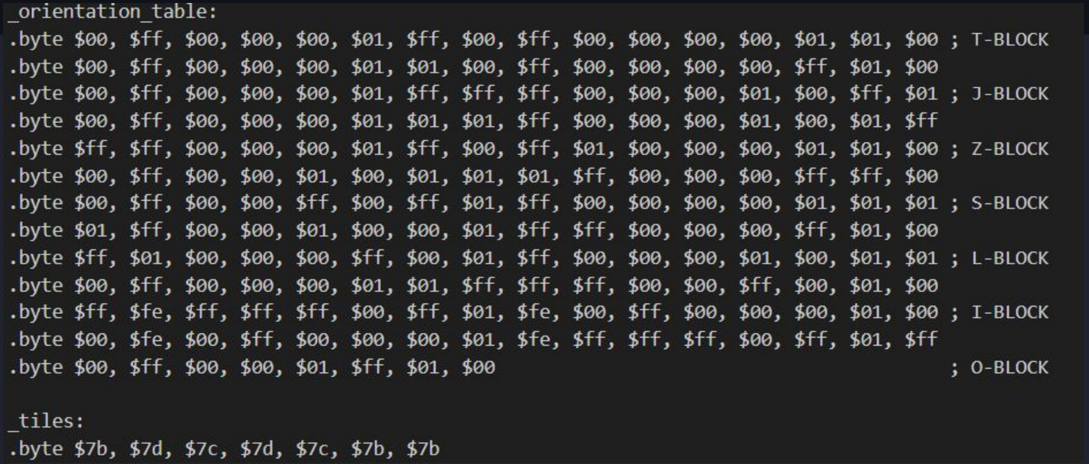


```
_update_table_index:
  lda #$69
  sta $0e
  lda #$fd
  sta $0f
  lda #$42 ; pieceorientation
  cmp #$14
  bcc @no_overflow
  lda #$fe
  sta $0f
  @no_overflow:
  lda a:_nextIDtoSprite,X     ; $8BDC   BD E5 8B
  rts
```

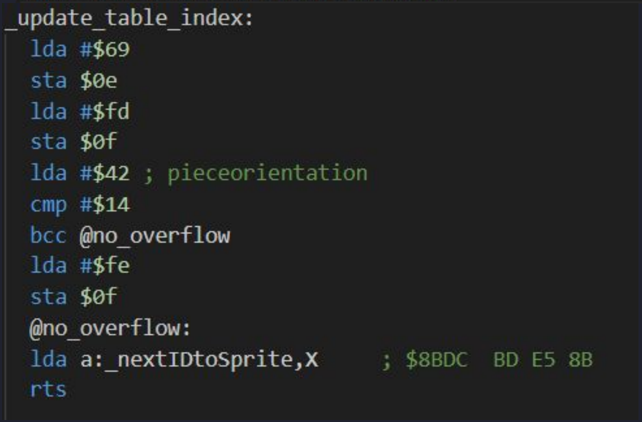
Spawning: Attempted Solutions

Cons

- *Involves lots of logic manipulation either way*
- *Takes too much time to experiment & test everything*
- *Involves replacing entire functions!*
- *Had to abandon both*



```
_orientation_table:
.byte $00, $ff, $00, $00, $00, $01, $ff, $00, $ff, $00, $00, $00, $00, $01, $01, $00 ; T-BLOCK
.byte $00, $ff, $00, $00, $00, $01, $01, $00, $ff, $00, $00, $00, $00, $ff, $01, $00
.byte $00, $ff, $00, $00, $00, $01, $ff, $ff, $ff, $00, $00, $00, $01, $00, $ff, $01 ; J-BLOCK
.byte $00, $ff, $00, $00, $00, $01, $01, $01, $ff, $00, $00, $00, $01, $00, $01, $ff
.byte $ff, $ff, $00, $00, $00, $01, $ff, $00, $ff, $01, $00, $00, $00, $01, $01, $00 ; Z-BLOCK
.byte $00, $ff, $00, $00, $01, $00, $01, $01, $01, $ff, $00, $00, $00, $ff, $ff, $00
.byte $00, $ff, $00, $00, $ff, $00, $ff, $01, $ff, $00, $00, $00, $00, $01, $01, $01 ; S-BLOCK
.byte $01, $ff, $00, $00, $01, $00, $00, $01, $ff, $ff, $00, $00, $00, $ff, $01, $00
.byte $ff, $01, $00, $00, $00, $ff, $00, $01, $ff, $00, $00, $00, $01, $00, $01, $01 ; L-BLOCK
.byte $00, $ff, $00, $00, $00, $01, $01, $ff, $ff, $ff, $00, $00, $ff, $00, $01, $00
.byte $ff, $fe, $ff, $ff, $ff, $00, $ff, $01, $fe, $00, $ff, $00, $00, $00, $01, $00 ; I-BLOCK
.byte $00, $fe, $00, $ff, $00, $00, $00, $01, $fe, $ff, $ff, $ff, $00, $ff, $01, $ff
.byte $00, $ff, $00, $00, $01, $ff, $01, $00                                         ; O-BLOCK

_tiles:
.byte $7b, $7d, $7c, $7d, $7c, $7b, $7b
```

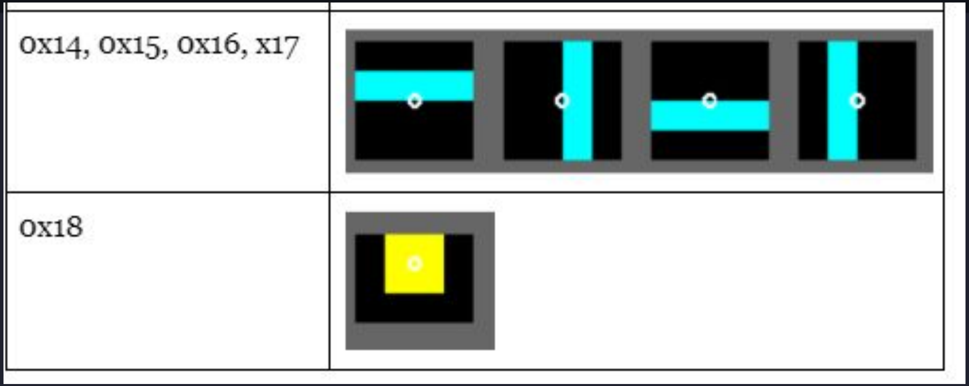


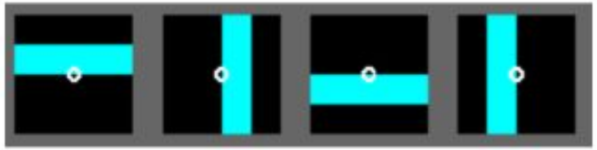
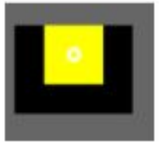
```
_update_table_index:
  lda #$69
  sta $0e
  lda #$fd
  sta $0f
  lda #$42 ; pieceorientation
  cmp #$14
  bcc @no_overflow
  lda #$fe
  sta $0f
@no_overflow:
  lda a:_nextIDtoSprite,X     ; $8BDC  BD E5 8B
  rts
```

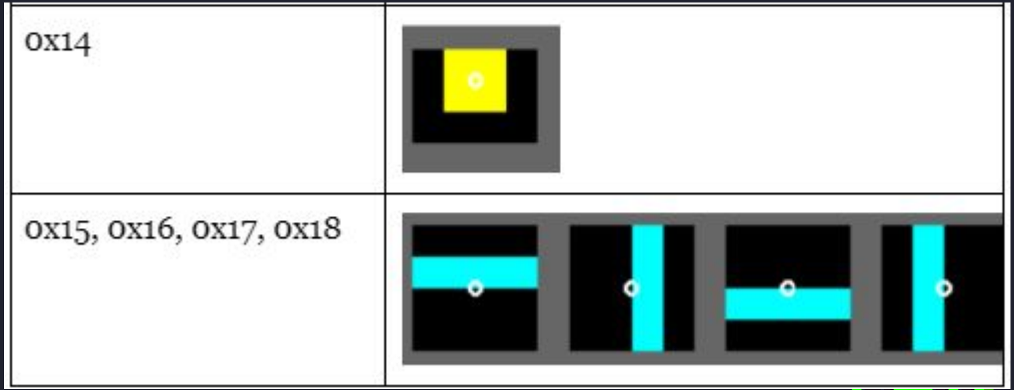
Spawning: Final (Partial) Solutions

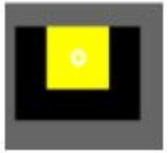
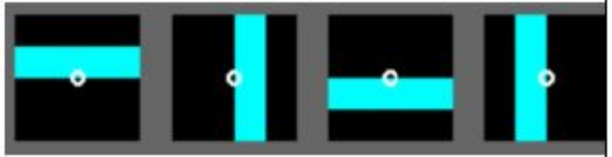
Restructure tables

- *O-piece = 0x14, I-piece = 0x15-0x18*
- *Spawn O-pieces instead of I-pieces*
- *Now only missing I-pieces*
- *Reordering tables was a simple task*



0x14, 0x15, 0x16, x17	
0x18	



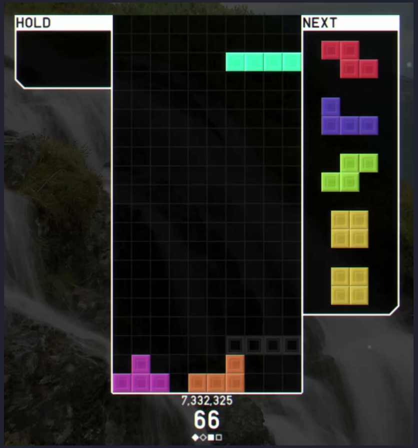
0x14	
0x15, 0x16, 0x17, 0x18	

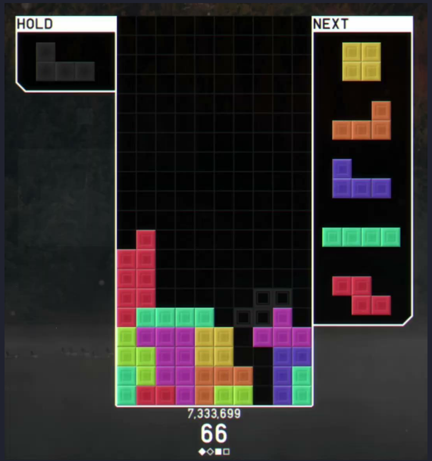
Wall Kicks

What Are Wall-Kicks?

- Wall-kicks “push” pieces around a little bit when you rotate them
- Makes the game a little bit friendlier to play
- The effects of these wall-kicks range from minor to weird

Video Examples

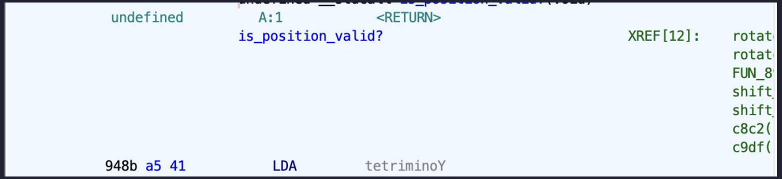




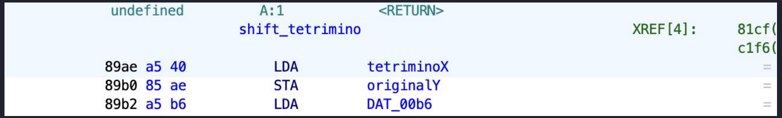
Existing Mechanics

There are a few helpful functions that we found early on

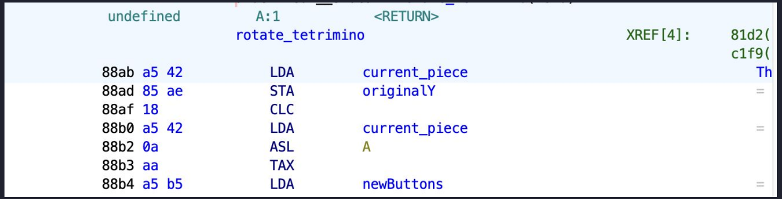
1) ***is_position_valid*** function for determining piece placement validity
2) ***shift_tetrimino*** can move tetriminos around the screen
3) ***rotate_tetrimino*** function for rotating the pieces



```
       undefined         A:1          <RETURN>
                         is_position_valid?                          XREF[12]:    rotat
                                                                                  rotat
                                                                                  FUN_8
                                                                                  shift
                                                                                  shift
                                                                                  c8c2(
                                                                                  c9df(
       948b a5 41          LDA          tetriminoY
```



```
       undefined         A:1          <RETURN>
                         shift_tetrimino                             XREF[4]:     81cf(
                                                                                  c1f6(
       89ae a5 40          LDA          tetriminoX                                    =
       89b0 85 ae          STA          originalY                                     =
       89b2 a5 b6          LDA          DAT_00b6                                      =
```



```
       undefined         A:1          <RETURN>
                         rotate_tetrimino                            XREF[4]:     81d2(
                                                                                  c1f9(
       88ab a5 42          LDA          current_piece                               Th
       88ad 85 ae          STA          originalY                                     =
       88af 18             CLC
       88b0 a5 42          LDA          current_piece                                 =
       88b2 0a             ASL          A
       88b3 aa             TAX
       88b4 a5 b5          LDA          newButtons                                    =
```

Wall-Kick Approach

1) Create a custom rotation function utilizing shift_tetrimino to check for piece placement
2) Check validity with is_position_valid

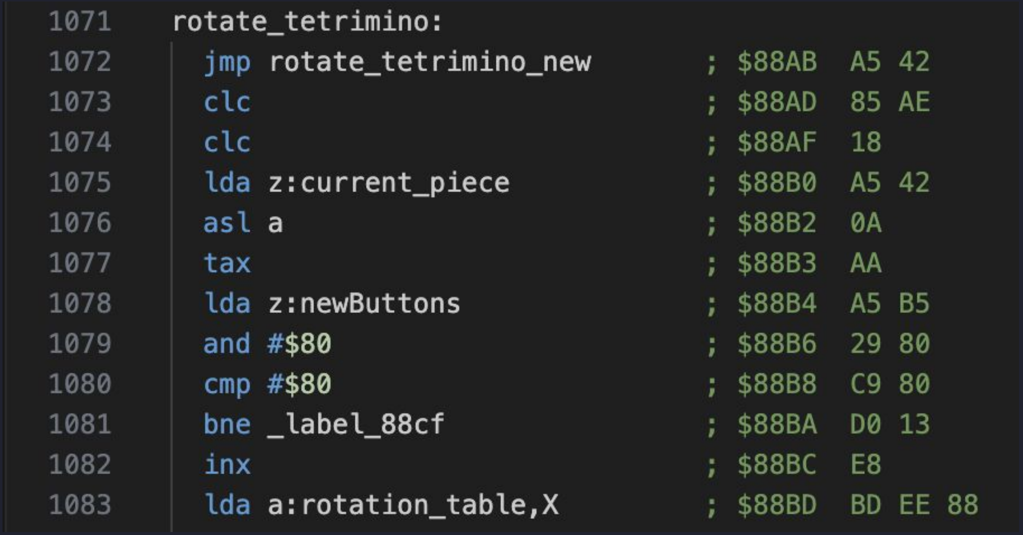
Wall-Kicks: Challenges & Solutions

Existing rotate doesn't do everything we need

- It rotates pieces, but if the piece doesn't perfectly fit as-is, it'll give up on the rotation entirely

Solution?

Add a jump to a new function at the end of the ROM



```
rotate_tetrimino:
  jmp rotate_tetrimino_new          ; $88AB   A5 42
  clc                               ; $88AD   85 AE
  clc                               ; $88AF   18
  lda z:current_piece               ; $88B0   A5 42
  asl a                             ; $88B2   0A
  tax                               ; $88B3   AA
  lda z:newButtons                  ; $88B4   A5 B5
  and #$80                          ; $88B6   29 80
  cmp #$80                          ; $88B8   C9 80
  bne _label_88cf                   ; $88BA   D0 13
  inx                               ; $88BC   E8
  lda a:rotation_table,X            ; $88BD   BD EE 88
```

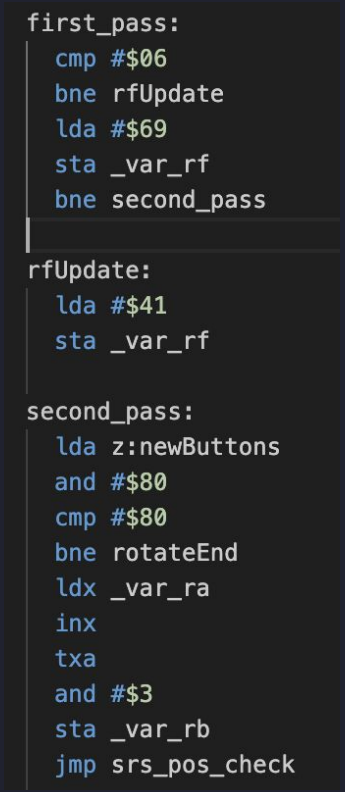
Wall-Kicks: Challenges & Solutions

The plan kind of got derailed immediately

- shift_tetrimino doesn't actually work how we hoped it did

Solution?

Write our own tetrimino shift checks (wall-kick biggest roadblock)



```
first_pass:
    cmp #$06
    bne rfUpdate
    lda #$69
    sta _var_rf
    bne second_pass

rfUpdate:
    lda #$41
    sta _var_rf

second_pass:
    lda z:newButtons
    and #$80
    cmp #$80
    bne rotateEnd
    ldx _var_ra
    inx
    txa
    and #$3
    sta _var_rb
    jmp srs_pos_check
```

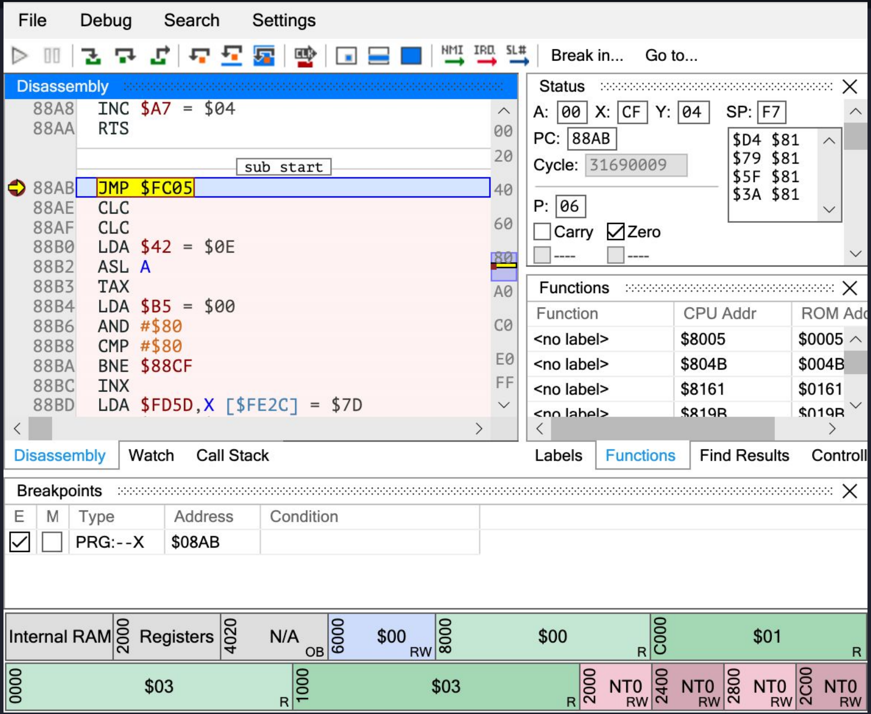
Wall-Kicks: Challenges & Solutions

6502 assembly is hard to work with

- Hard to write and there isn't as much documentation out there as we would've liked

Solution?

A lot of trial and error with the debugger in the emulator



Disassembly

```
88A8  INC $A7 = $04
88AA  RTS

          sub start
88AB  JMP $FC05
88AE  CLC
88AF  CLC
88B0  LDA $42 = $0E
88B2  ASL A
88B3  TAX
88B4  LDA $B5 = $00
88B6  AND #$80
88B8  CMP #$80
88BA  BNE $88CF
88BC  INX
88BD  LDA $FD5D,X [$FE2C] = $7D
```

Status

```
A: 00  X: CF  Y: 04     SP: F7
PC: 88AB                $D4 $81
Cycle: 31690009         $79 $81
                        $5F $81
                        $3A $81
P: 06
☐ Carry  ☑ Zero
☐ ----   ☐ ----
```

Functions

Function	CPU Addr	ROM Add
<no label>	$8005	$0005
<no label>	$804B	$004B
<no label>	$8161	$0161
<no label>	$819B	$019B

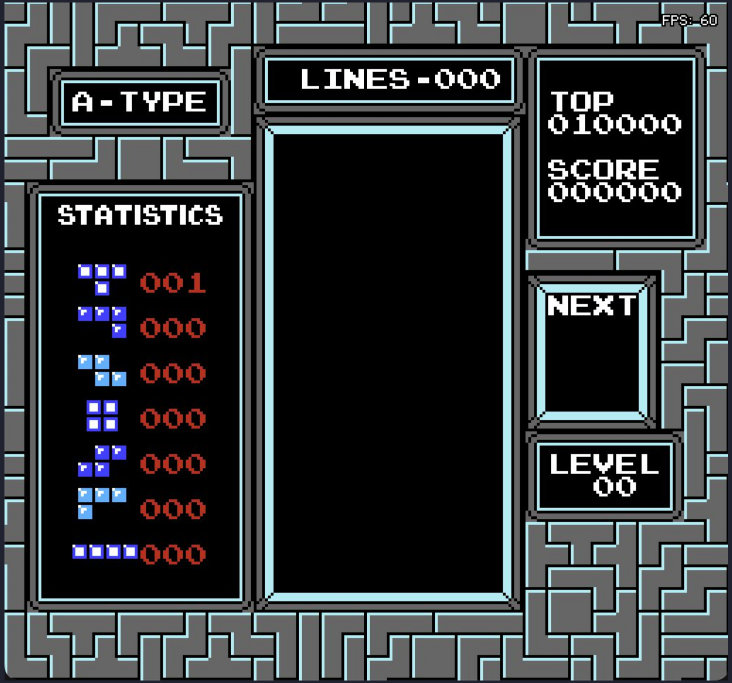
Breakpoints

E	M	Type	Address	Condition
☑	☐	PRG:--X	$08AB	

Wall-Kicks: Challenges

Given the allotted time, we couldn't flesh out wall-kicks correctly

- We kept running new issues with every turn, and debugging only got harder
- Had to leave them out the final product



Final Product

Final Product

Additions!

- +6 new rotation states!
- Expanded graphics table

What we'd add next time

- Proper spawning (smaller graphics table)
- Wall-kicking
- Fix statistics
- Fix minor graphical issues

Overall?

- very proud :)

Demo!

Questions?

Thanks for watching!

P.S.
If you give us really good grades we'll also put your name in the title screen! Not a bribe