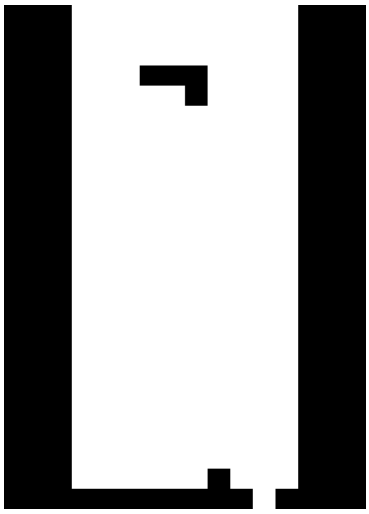
8 Tetranglix: This Tetris is a Boot Sector

by Juhani Haverinen, Owen Shepherd, and Shikhin Sethi

Since Dakarand in a 512-byte boot sector would have been too easy, and since both Tetris and 512-byte boot sectors are the perfect ingredients to a fun evening, the residents of #osdev-offtopic on FreeNode took to writing a Tetris clone in the minimum number of bytes possible. This tetris game is available by unzipping this PDF file, through Github,[6] by typing the hex from page 32, or by scanning the barcode on page 31.



There's no fun doing anything without a good challenge. This project presented plenty, a few of which are described in this article.

To store each tetramino, we used 32-bit words as bitmaps. Each tetramino, at most, needed a 4 by 4 array for representation, which could easily be flatenned into bitmaps.

```
; All tetraminos in bitmap format.
tetraminos:
    dw 0b0000111100000000   ; I     -Z--  -S--  -O--
    dw 0b0000111000100000   ; J
    dw 0b0000001011100000   ; L     0000  0000  0000
    dw 0b0000011001100000   ; O     0110  0011  0110
    dw 0b0000001101100000   ; S     0011  0110  0110
    dw 0b0000111001000000   ; T     0000  0000  0000
    dw 0b0000011000110000   ; Z
```

Instead of doing bound checks on the current position of the tetramino, to ensure the user can't move it out of the stack, we simply restricted the movement by putting two-block wide boundaries on the playing stack. The same also added to the esthetic appeal of the game.

To randomly determine the next tetramino to load, our implementation also features a Dakarand-style random number generator between the RTC and the timestamp counter.

```
; Get random number in AX.
rdtsc                       ; The timestamp counter.
xor ax, dx

; (INTERMEDIATE CODE)

; Yayy, more random.
add ax, [0x046C]            ; And the RTC (updated via BIOS).
```

The timestamp counter also depends on how much input the user provided. In this way, we ensure that the user adds to the entropy by playing the game.

Apart from such obvious optimizations, many nifty tricks ensure a minimal byte count, and these are what make our Tetranglix code worth reading. For example, the same utility function is used both to blit the tetramino onto the stack and to check for collision. Further optimization is achieved by depending upon the results of BIOS calls and aggressive use of inlining.

While making our early attempts, it looked impossible to fit everything in 512 bytes. In such moments of desperation, we attempted compression with a simplified variant of LZSS. The decompressor clocked at 41 bytes, but the compressor was only able to reduce the code by 4 bytes! We then tried LZW, which, although saved 21 bytes, required an even more complicated decompression routine. In the end, we managed to make our code dense enough that no compression was necessary.

[6] `https://github.com/Shikhin/tetranglix`

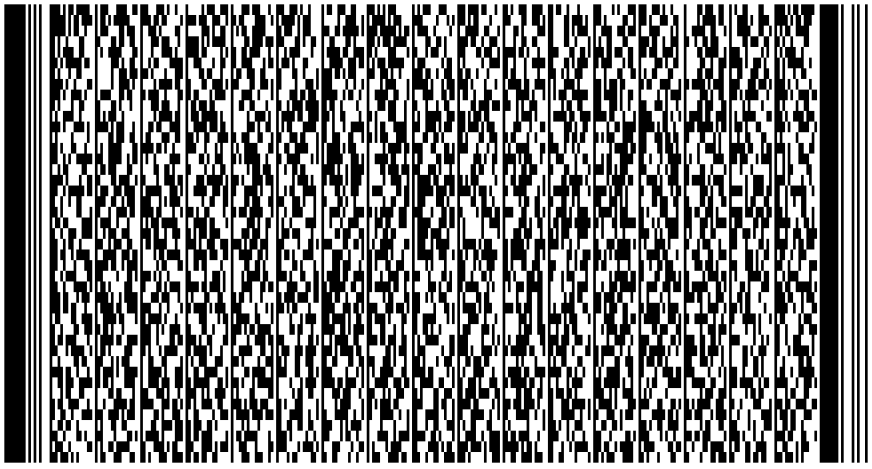
Since the project was written to meet a strict deadline, we couldn't spend more time on optimization and improvement. Several corners had to be cut.

The event loop is designed such that it waits for the entirety of two PIT (programmable interval timer) ticks—$109.8508mS$—before checking for user input. This creates a minor lag in the user interface, something that could be improved with a bit more effort.

Several utility functions were first written, then inlined. These could be rewritten to coexist more peacefully, saving some more space.

As a challenge, the authors invite clever readers to clean up the event loop, and with those bytes shaved off, to add support for scoring. A more serious challenge would be to write a decompression routine that justifies its existence by saving more bytes than it consumes.

```
; IT'S A SECRET TO EVERYBODY.
db "ShNoXgSo"
```



Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
0000_0000	ea	05	7c	00	00	31	db	8e	d3	bc	00	7c	8e	db	8e	c3
0000_0010	fc	bf	04	05	b9	b6	01	31	c0	f3	aa	b0	03	cd	10	b5
0000_0020	26	b0	03	fe	c4	cd	10	b8	00	b8	8e	c0	31	ff	b9	d0
0000_0030	07	b8	00	0f	f3	ab	be	2a	05	66	b8	db	db	db	db	66
0000_0040	89	44	fd	89	44	01	83	c6	10	81	fe	ba	06	76	f0	30
0000_0050	d2	be	24	05	bf	b8	7d	fb	8b	1e	6c	04	83	c3	02	39
0000_0060	1e	6c	04	75	fa	84	d2	75	37	fe	c2	60	0f	31	31	d0
0000_0070	31	d2	03	06	6c	04	b9	07	00	f7	f1	89	d3	d0	e3	8b
0000_0080	9f	e8	7d	bf	04	05	be	db	00	b9	10	00	30	c0	d1	e3
0000_0090	0f	42	c6	88	05	47	e2	f4	61	c7	04	06	00	e9	a5	00
0000_00a0	b4	01	cd	16	74	59	30	e4	cd	16	8b	1c	80	fc	4b	75
0000_00b0	06	fe	0c	ff	d7	72	46	80	fc	4d	75	06	fe	04	ff	d7
0000_00c0	72	3b	80	fc	48	75	38	31	c9	fe	c1	60	06	1e	07	be
0000_00d0	04	05	b9	04	00	bf	13	05	01	cf	b2	04	a4	83	c7	03
0000_00e0	fe	ca	75	f8	e2	ef	be	14	05	bf	04	05	b1	08	f3	a5
0000_00f0	07	61	e2	d7	ff	d7	73	07	b9	03	00	eb	ce	89	1c	fe
0000_0100	44	01	ff	d7	73	3f	fe	4c	01	30	d2	60	06	1e	07	ba
0000_0110	99	7d	e8	87	00	31	c9	be	2a	05	b2	10	30	db	ac	84
0000_0120	c0	0f	44	da	fe	ca	75	f6	84	db	75	0b	fd	60	89	f7
0000_0130	83	ee	10	f3	a4	61	fc	83	c1	10	81	f9	90	01	72	da
0000_0140	07	61	e9	f1	fe	60	bf	30	00	be	2a	05	b9	10	00	ac
0000_0150	aa	47	aa	47	e2	f9	83	c7	60	81	ff	a0	0f	72	ed	61
0000_0160	60	8a	44	01	b1	50	f6	e1	0f	b6	3c	d1	e7	83	c7	18
0000_0170	01	c7	d1	e7	b1	10	be	04	05	b4	0f	84	c9	74	16	fe
0000_0180	c9	ac	84	c0	26	0f	44	05	ab	ab	f6	c1	03	75	ec	81
0000_0190	c7	90	00	eb	e6	61	e9	bf	fe	08	05	c3	60	e8	35	00
0000_01a0	b1	10	84	c9	74	10	fe	c9	ac	ff	d2	47	f6	c1	03	75
0000_01b0	f1	83	c7	0c	eb	ec	61	c3	60	f8	ba	c2	7d	e8	dc	ff
0000_01c0	61	c3	3c	db	75	0e	81	ff	ba	06	73	04	3a	05	75	04
0000_01d0	83	c4	12	f9	c3	0f	b6	44	01	c1	e0	04	0f	b6	1c	8d
0000_01e0	78	06	01	c7	be	04	05	c3	00	0f	20	0e	e0	02	60	06
0000_01f0	60	03	40	0e	30	06	53	68	4e	6f	58	67	53	6f	55	aa

This is a complete Tetris game.