Luke Jaffe
Operating Systems
Due 04/27/16

Final Project
xv6 Tetris

The goal of this project was to add a Tetris game program to xv6, which could be viewed on the 320x200 VGA, and controlled with keyboard interrupts. To do this, pixel-level control of the VGA was required. Further, the ability to write frequently to the VGA quickly and without flickering was set as a soft requirement. A ~0.1 second granularity refresh rate would be sufficient for a typical Tetris game. To control the game, the ability to write shared memory from keyboard interrupts was needed. The game should be responsive, in that keyboard interrupts would quickly modify the screen state to reflect the state of the game.

To set the display mode for the VGA, interrupt 0x10 must be used. To use and modify the VGA, mode 0x13 must be activated, which is supported by QEMU. Upon calling interrupt 0x10 with this mode, QEMU will resize the SDL window to match the VGA dimensions. A special function called int32(), written by Napalm, is used to execute BIOS interrupts by switching out to real-mode and then back into protected mode. This is defined in int32.nasm, which requires the nasm tool to build. This nasm tool is the only dependency for the project, aside from QEMU. To switch back into text mode, interrupt 0x10 must be called with mode 0x03.

There are two basic display methods available for this VGA. The first is a simple one-to-one mapping from byte position to pixel position, but has only 64k of memory. This is called chain-4 mode, and is the default for mode 0x13. The second available mode, unchained mode, can be used to access all 256k of video memory, but is a bit trickier to use. In this mode, the VGA memory is divided into four 64k planes, where the the first plane maps to pixels 0, 4, 8,... the second to 1,5,9,... and so on. This is shown in figure 1. This divides the memory into four pages, only one of which is visible at a time. Switching the page that the VGA displays is a very quick operation called "page flipping". This can be used to speed up rendering, but the overhead of mapping pixels to different planes can cause it to be slower for complex drawing operations. Both of these methods were implemented for this project, though there is no discernible difference in performance for the Tetris use case.

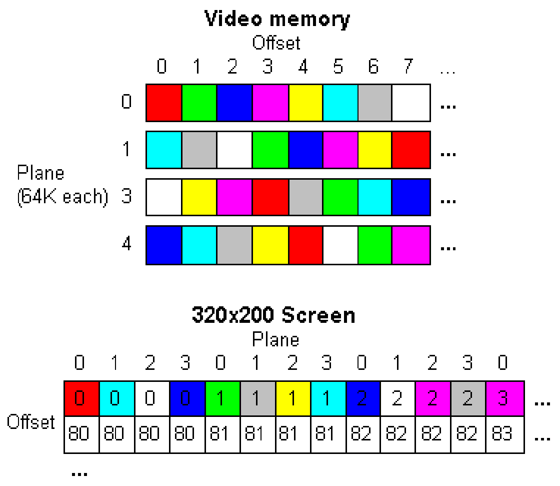

Figure 1: VGA Memory

Using keyboard interrupts as game input was a fairly simple implementation, requiring modifications to the console.c and proc.c files in xv6. In addition, a shared variable called keycode (int) was added to proc.h, to be shared between console.c and proc.c. Switching to real mode and running BIOS functions is not possible in an interrupt context, and will cause a crash. While we don't need to do these sensitive operations in the current Tetris implementation, we do some fairly heavy computation that is better handled outside of interrupt context. Therefore, we will use the console interrupt to wakeup the init process, and call our keyhandling code there.

When the console interrupt is triggered it receives a character. We check this character against the keys we use (left, right, up, down, space), and call the wakeup function on initproc if we get a hit. The init process just waits for child processes to die, and is otherwise asleep. So in the wait() function in proc.c, we check if the current process is initproc, and call our keyhandling code if it is. The keyhandling code will update the double buffer, and then immediately draw it to the VGA so we get instant feedback.

The actual Tetris game logic is relatively simple, once we have the ability to draw pixelwise on the VGA and execute code on keypresses. The game starts by setting a Tetris piece to the top of the screen. In the main loop, the piece will move down one block, then we check if it has landed on the bottom or on top of any other blocks. If it has landed, we lock the piece down and spawn a new piece at the top. If it is locked into the top of the screen, this indicates the losing condition, and the game ends, returning the score achieved to the user. Otherwise, the piece continues to move down till it hits something, with a certain number of cpu ticks between loop iterations so the user has time to react. When all the blocks in a row are filled, the row is cleared, and all blocks above that row move down one row. Clearing rows increments the user's score.

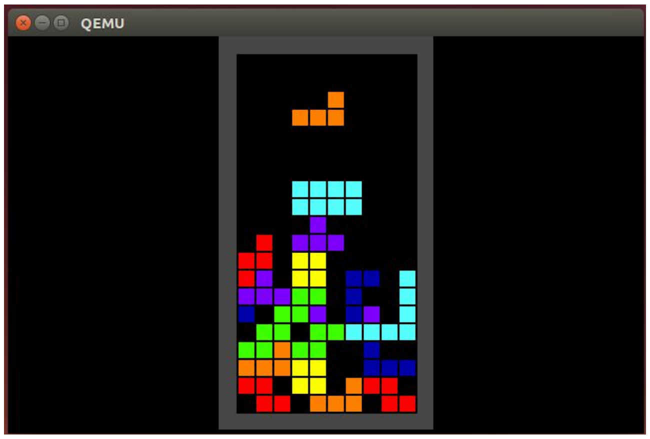

Figure 2: Game Screenshot

The user interacts with the game by moving the current piece left to write with the left/right arrow keys, rotating the piece with the up arrow key, and moving the piece down faster with the down arrow key, or dropping it directly to the bottom with the spacebar key. There are seven different tetrominoes, each of which has four rotations. The tetris wiki, listed as reference 3, offers detail on variations of the game rules, and was a great resource in implementing the game. A typical game screenshot is shown in figure 2.

Reference Links

[1] Writing to VGA in xv6: https://www.cs.uic.edu/bin/view/CS385fall14/Homework3

[2] Unchained mode and page flipping: http://www.brackeen.com/vga/unchain.html

[3] Tetris wiki: http://tetris.wikia.com/wiki/SRS