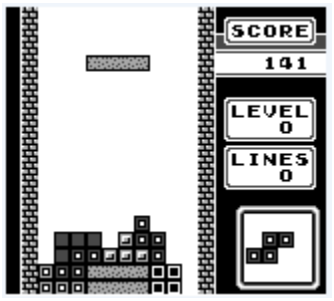
FINAL SPRINT – OPT/EXTENTION

This sprint we did half type-extension half optimization. We didn't have many possible optimizations other than assisting the extendibility of our emulator.

Summary of Features Implemented:

- -Screenshot disassembler
- -readMemory
- -Memory Banking

Screenshot was implemented to allow us to demonstrate the visual aspect of our IR. We binded our screenshot to the tab button, and stores data in a capture folder. We start by taking generating a screenshot of the current state of the screen. For example, consider this image of Tetris:



From here we can show the layers of the screen that we interpret. The background layer is displayed in the file BG.bmp and for the above image here is the generated from the snapshot above:

One thing to note is that the sprite layer is gone here.

We then generate two folders OAM and tiles. OAM are the tiles referring to sprites on the screen, these are mainly used for tiles that move around. There are 40 OAM tiles at any given time, and have coordinates stored in memory to know where to place them. Tiles are all the tiles found on memory map 0. The tiles show what is set on the VRAM, there are 256 of these at a given time, however several of them are likely garbage. These tiles include the font tiles.

readMemory allows for us to move around a few functions which cleans up the code. The need for this arose when we tried to implement the memory banker. Since we need to control the reads to make sure we are hitting the correct bank. It also moves a static function from being implemented every game loop to only being called when it is used. This makes the code more optimized. It's probably the best we can do for any optimization since optimizing actual byte code for the gameboy games would be near impossible since developers had cpu cycles in mind when designing the games as well as we don't know every location that could be jumped to beforehand.

The original project had no memory banking. Memory banking allows for the extension of additional game types. Memory banks 0 and 1 are functional. Since the gameboy CPU can only read and write to addresses 0000 - 7FFF for the cartridge, gameboy games use memory banking to increase the amount of data they can contain within the cartridge. Memory banks are 4000 bytes. Memory bank 0 is from 0000-3FFF, 1 from 4000-7FFF and so on. By implementing memory banking, we extend the amount of games we can support. Currently we support only MBC0, we have the ground work for MBC1 with Ram and battery, however as of writing this it doesn't seem we can fully support it.