

KNOX

# GameBoy

GAME

Knox Game Design

October 2026

DESIGN

# Background

- Released in 1989
- Gunpei Yokoi, Nintendo Research & Development 1
- CPU: Sharp LR35902, SM83 instruction set
- Resolution 160 x 144 pixels
- Memory: 8 KB
- Power: 4 x AA (6 volts)

# RGBDS (Assembly)

- GameBoy assembler
- 8 bit registers
  - A, B, C, D, E, H, L
- 16 bit registers
  - BC, DE, HL
- OpCodes
  - LD – load
  - ADD, ADC, SUB, SBC – add, subtract
  - CP – compare
  - INC, DEC – increment, decrement
  - AND, OR, XOR – bitwise operators
- OpCodes
  - BIT, RES, SET – bit operations
  - RL, RR – bit rotation (multiply and divide by 2)
  - JP, CALL, RET – jump, call subroutine, return from subroutine
- Use square brackets to get value at specified location

# RGBDS – tools and build

- Tools

- Command line environment - MSYS2
- Build tools – RGBDS
  - assembler – rgbasm
  - linker – rgblink
  - checksum fix - rgbfix
- Editor –Visual Studio Code, vim

- Build

- Add RGBDS tools to path
- Run assembler, link, and fix

```
ldsmi@octorok MSYS /d/ldsmith/presentations/demos/gameboy/gameboy00
$ export PATH=${PATH}:/d/ldsmith/presentations/gameboy/rgbds-win64
```

```
ldsmi@octorok MSYS /d/ldsmith/presentations/demos/gameboy/gameboy00
$ rgbasm -o gameboy00.o gameboy00.asm
```

```
ldsmi@octorok MSYS /d/ldsmith/presentations/demos/gameboy/gameboy00
$ rgblink -o gameboy00.gb gameboy00.o
```

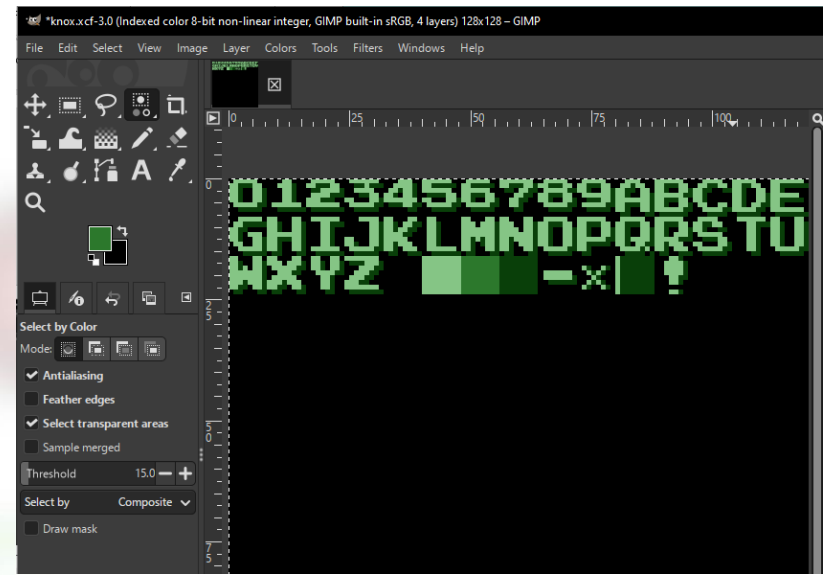
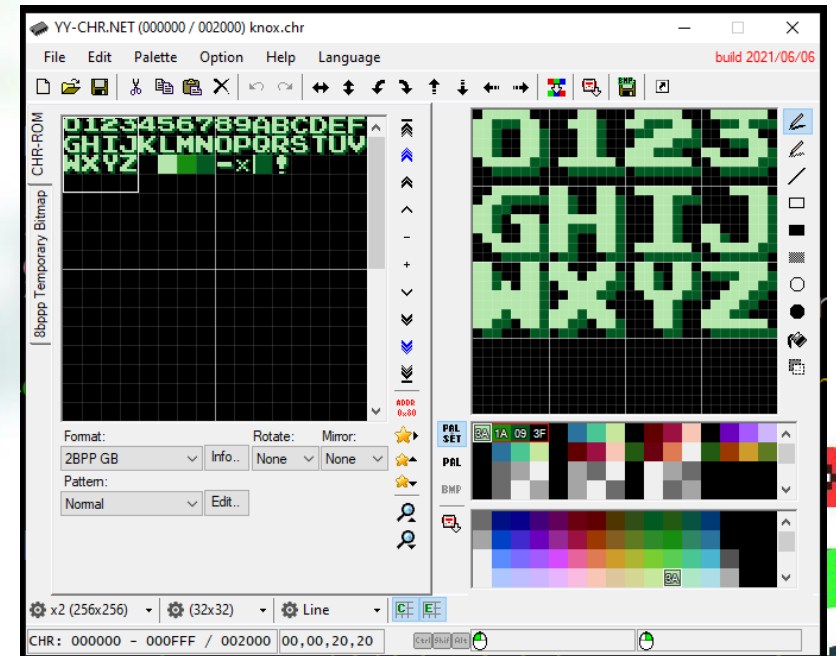
```
ldsmi@octorok MSYS /d/ldsmith/presentations/demos/gameboy/gameboy00
$ rgbfix -v -p 0xFF gameboy00.gb
```

# Tiles and Graphics

- Gimp for editing sprites
  - Set mode to index with maximum 4 colors
- YY-CHR for tile creation
  - Make sure to select **2BPP GB** for *Graphic format*
  - Copy and Paste image from Gimp
  - Export to <file>.chr
  - Use INCBIN "<file>.chr" to load tile data
    - Similar to include or require in other languages
  - Use str\_to\_nes.py to generate text data

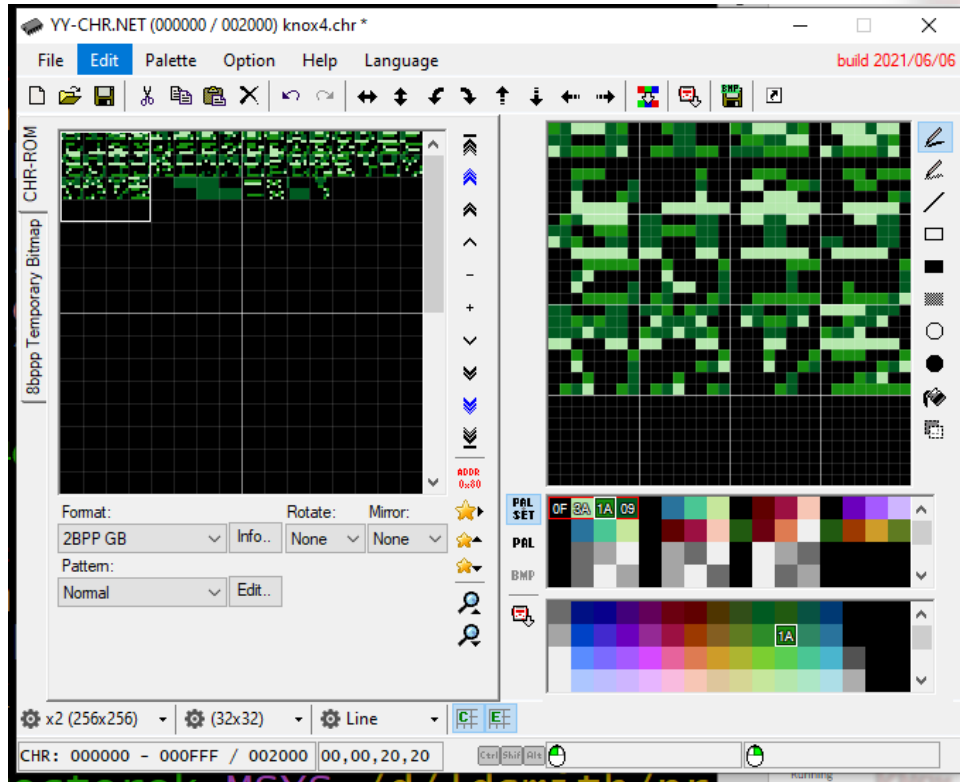
```
64 Tiles:
65 INCBIN "knox.chr"
66 .End:
```

```
$ python ../str_to_nes/str_to_nes.py
Enter string
knox game design
KNOX GAME DESIGN
$14, $17, $18, $21, $24, $10, $0a, $16, $0e, $24, $0d, $0e, $1c,
$12, $10, $17, $24, $24, $24, $24
```



DESIGN

# Possible Issues



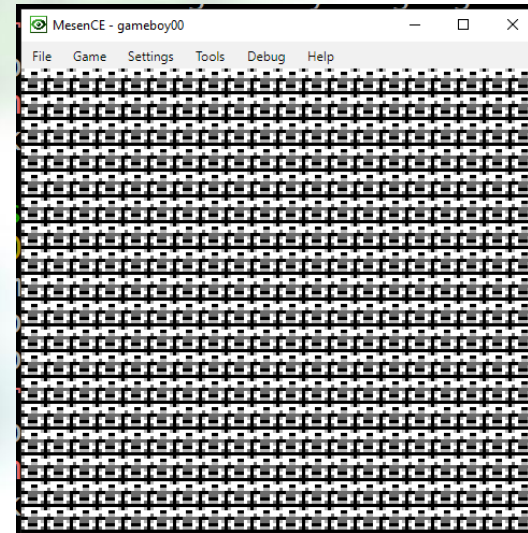
Make sure to change format to 2BPP GB **before** pasting, otherwise your tile data will look like this

```
CHR: 000000 - 000FFF / 007000
```

```
$ make
rgbasm -o gameboy00.o gameboy00.asm
rgblink -o gameboy00.gb gameboy00.o
error: Section "Tile data" is bigger than the max size for that
type: $7000 > $4000
Linking failed with 1 error
make: *** [Makefile:3: gameboy00.gb] Error 1
```

```
$ dd if=knox4.chr of=knox5.chr bs=1 count=8192
8192+0 records in
8192+0 records out
8192 bytes (8.2 kB, 8.0 KiB) copied, 0.0262464 s, 312 kB/s
```

Tile data cannot exceed \$4000. By default new YY-CHR files use \$7000. Use **dd** to truncate chr file to 8192 bytes.

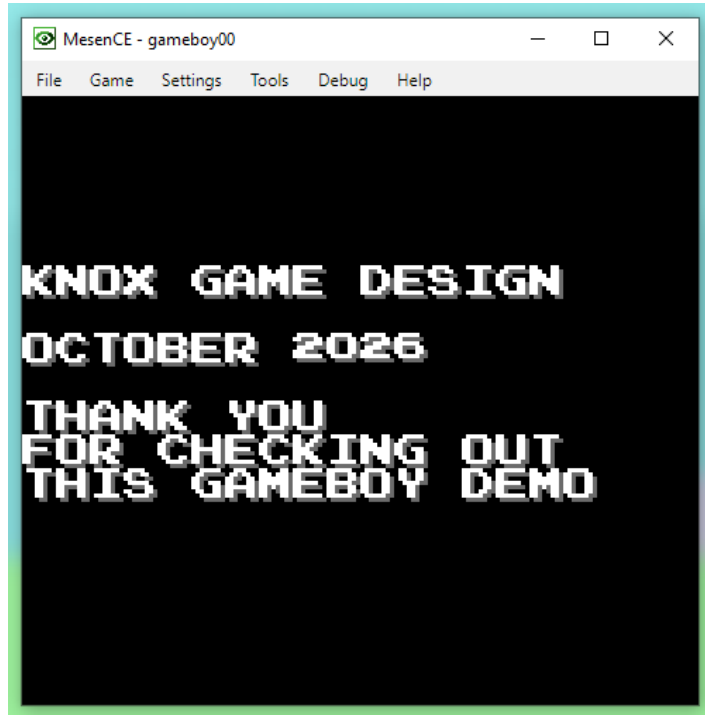


Using data in 2BPP NES format (default) will look like a scrambled mess in emulator.

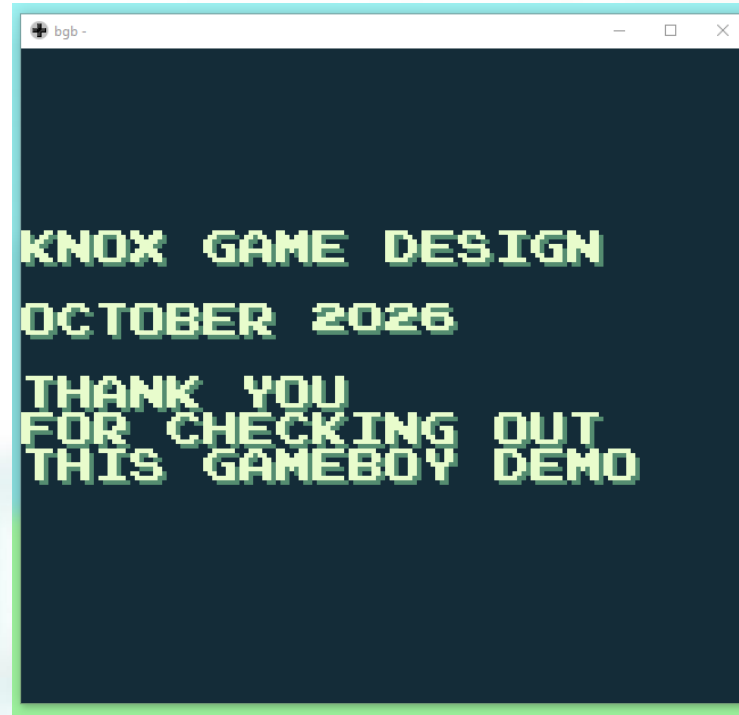
**KNOX**  
**GAME**  
**DESIGN**

# Running

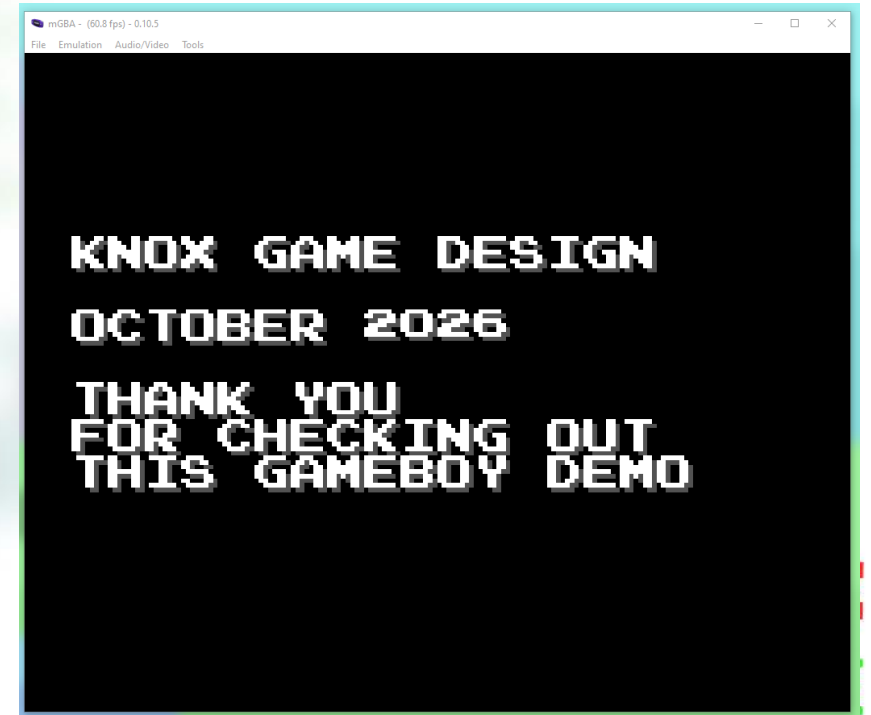
- Output is a .gb ROM file
- Run with emulator (such as Mensen, bgb, mGBA)
- Must reload (File > Open) file in emulator after each compile
  - Mensen: Ctrl+Shift+R / Reload ROM. Just Reset won't load new ROM.



Mensen



bgb



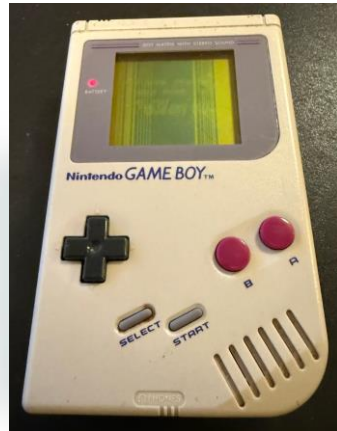
mGBA

DESIGN

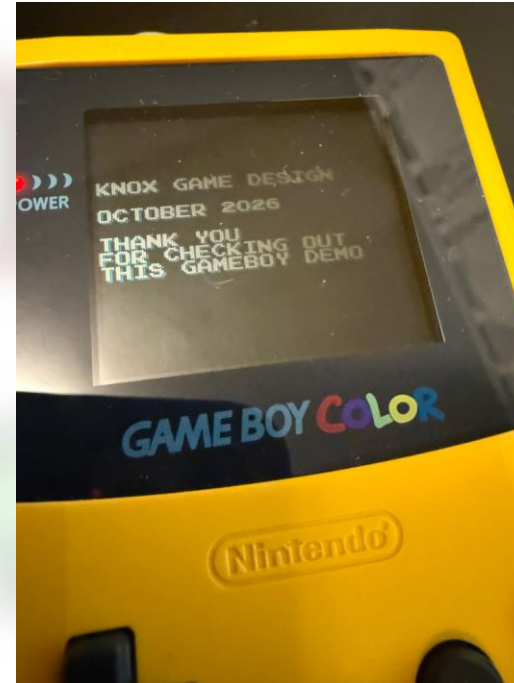
# Running on hardware



Everdrive GB X7



My original GameBoy



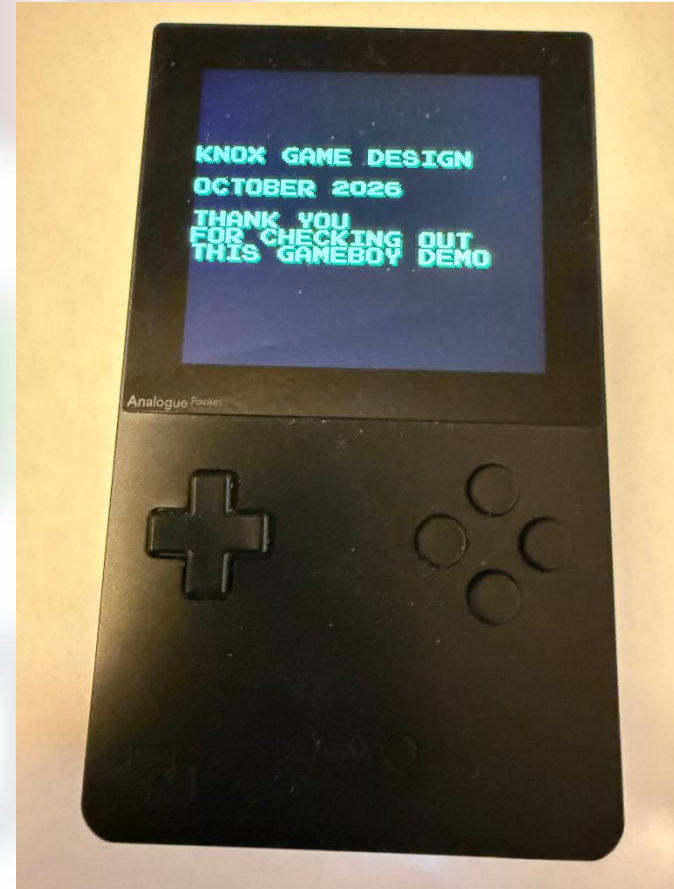
GameBoy Color

**KNOX**  
**GAME**  
**DESIGN**

# Running on Hardware



GameBoy Advance

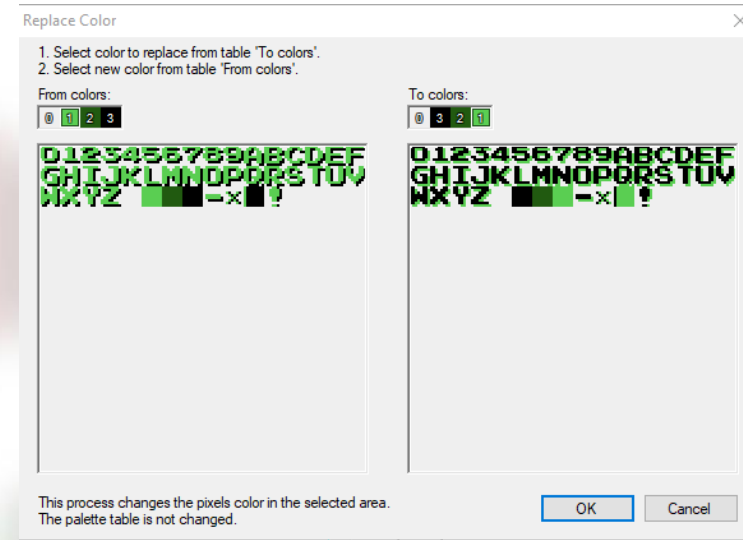


Analogue Pocket  
(open fpga or Everdrive)

**KNOX**  
**GAME**  
**DESIGN**

# Changing Palette

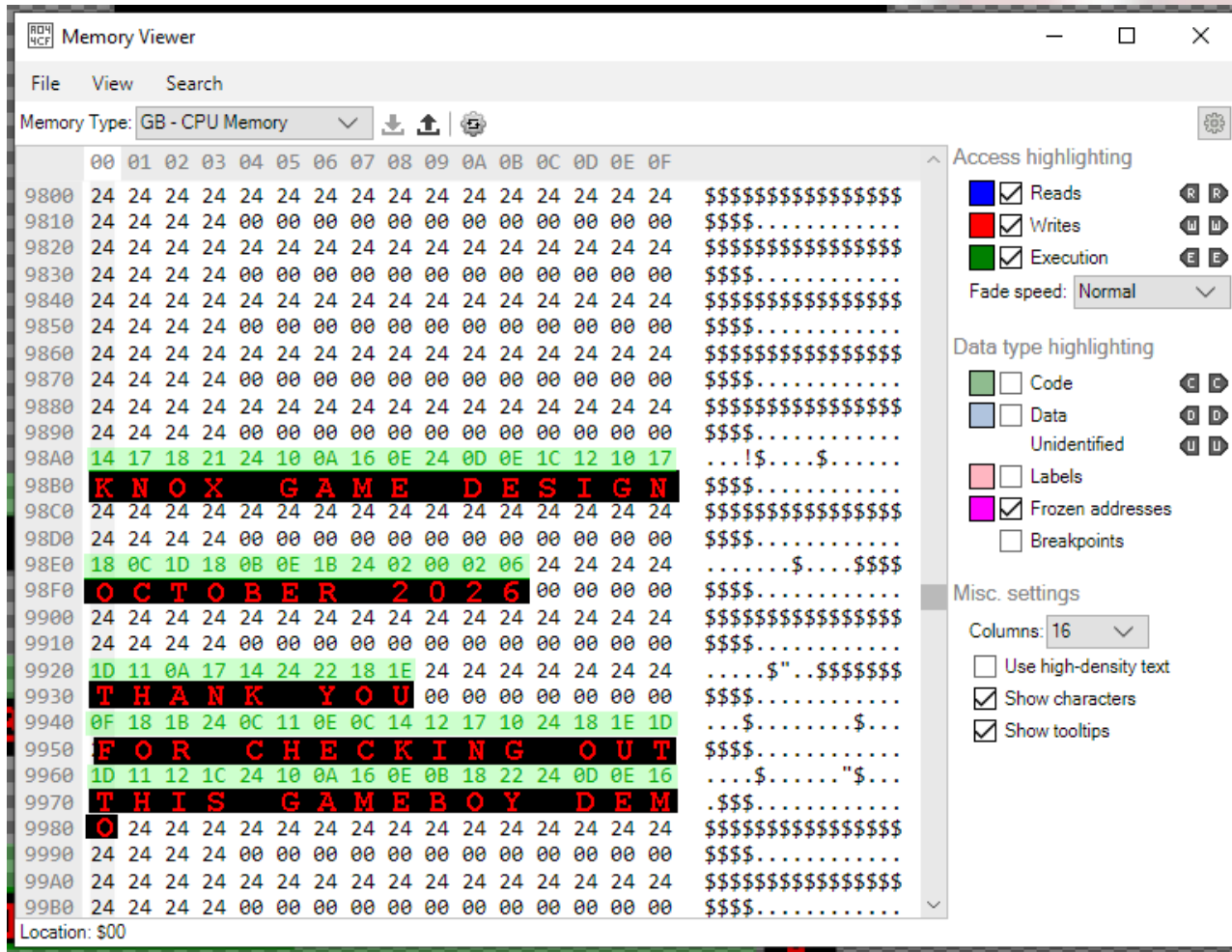
- Use Edit > Replace Color to change palette index order
  - Select *To Color* on right, then *From color* on left
- Suggest dark text on light background
  - Easier to read on non-backlit real hardware



**KNOX**  
**GAME**  
**DESIGN**

# Tile map in memory

- Mesen: Debug > Memory Viewer
- Tile map data starts at location \$9800 in memory



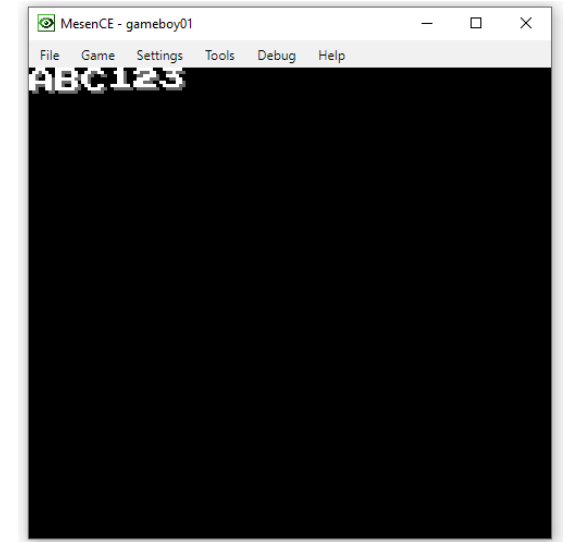
**KNOX**  
**GAME**  
**DESIGN**

# Updating tile map

- Update the tile map in real time
  - Square brackets returns the value stored at memory location
  - *hli* increments the value in the *hl* register after
- This code writes “ABC123” in the upper left corner

```
Done:
    ld hl, $9800
    ld a, $a
    ld [hli], a
    ld a, $b
    ld [hli], a
    ld a, $c
    ld [hli], a
    ld a, $1
    ld [hli], a
    ld a, $2
    ld [hli], a
    ld a, $3
    ld [hli], a

    jr Done
```

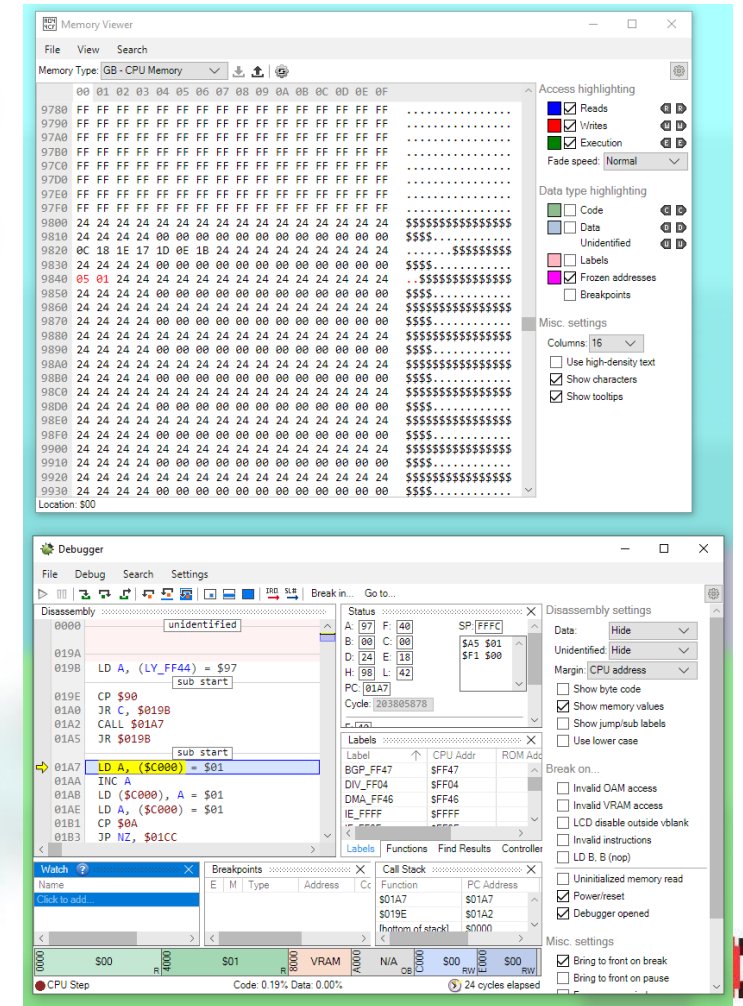


# Make a counter

- Use *call* to execute the *IncrementCounter* subroutine
- Define two bytes in WRAM0 to hold tens digit and ones digit
- Algorithm
  - Increment Ones Digit
  - Compare Ones Digit with 10 (\$a)
  - If equal, then reset Ones Digit back to zero and increment Tens Digit
    - Compare Tens Digit with 10 (\$a)
    - If equal, then reset Tens Digit back to zero
- Use debugger and memory viewer to verify counter working correctly
  - Updated memory bytes are in red text
  - Use step to step through instructions

```
SECTION "Counter", WRAM0
myCounterDigitOnes: db
myCounterDigitTens: db
```

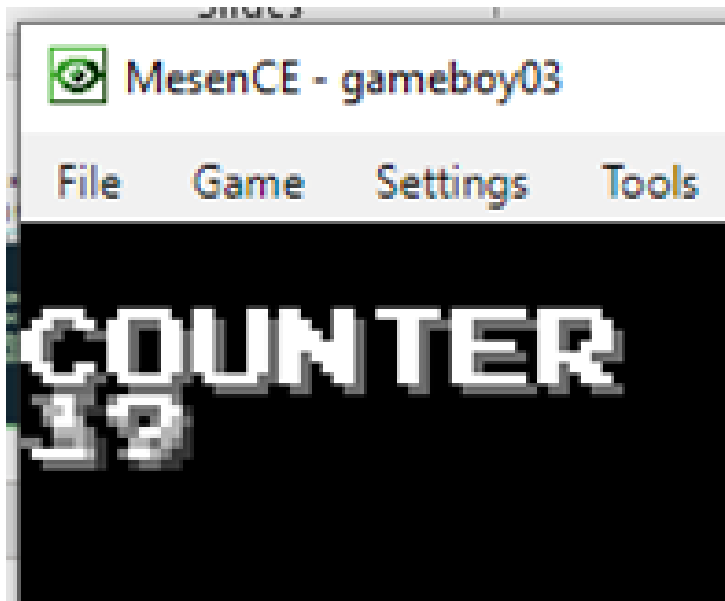
```
99 IncrementCounter:
100     ld a, [myCounterDigitOnes]
101     inc a
102     ld [myCounterDigitOnes], a
103
104     ld a, [myCounterDigitOnes]
105     cp a, $a
106     jp nz, CounterResetTensDone
107
108 CounterResetOnes:
109     ld a, 0
110     ld [myCounterDigitOnes], a
111
112     ld a, [myCounterDigitTens]
113     inc a
114     ld [myCounterDigitTens], a
115
116     cp a, $a
117     jp nz, CounterResetTensDone
118 CounterResetOnesDone:
119
120 CounterResetTens:
121     ld a, 0
122     ld [myCounterDigitTens], a
123 CounterResetTensDone:
124
125     ld hl, $9840
126     ld a, [myCounterDigitTens]
127     ld [hli], a
128     ld a, [myCounterDigitOnes]
129     ld [hli], a
130     ret
```



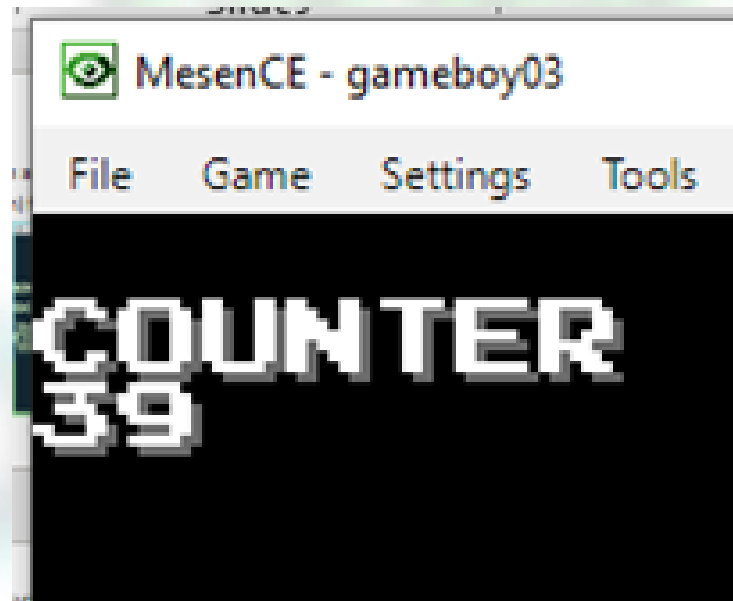
GAME  
DESIGN

# Emulator frame blending

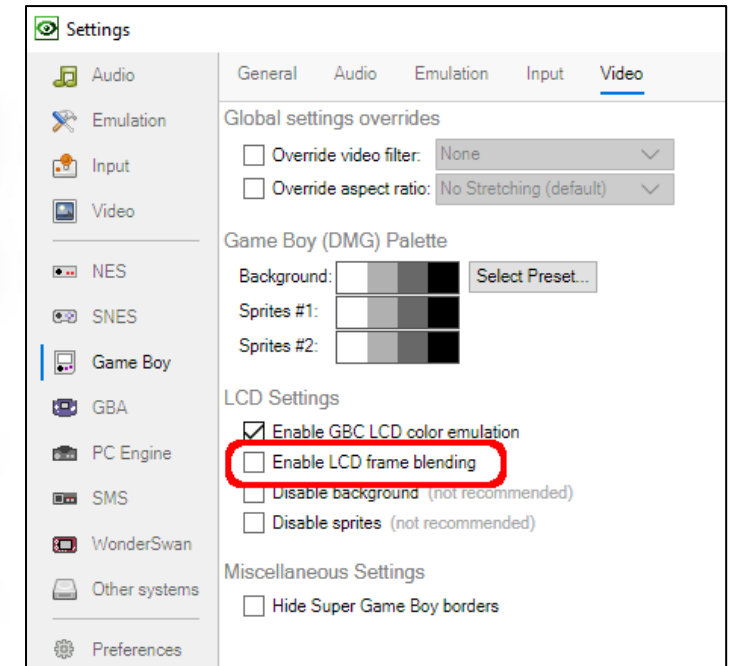
- Suggest disabling *Enable LCD frame blending* in emulator video settings
- If enabled, pausing and step debugging will result in a blurry image



Frame blending enabled



Frame blending disabled



**KNOX**  
**GAME**  
**DESIGN**

# VBlank

- Should only update tiles during VBlank
- Options
  - Check each cycle if VBlank is active
    - 144 vertical scanlines
    - rLY register contains the current scanline
    - Use [rLY] to access the register value
    - Check if greater than 144
  - Create Interrupt that triggers when VBlank starts
    - INT \$40 is VBlank interrupt
    - Load into rIE register
    - Set rIF register to zero
    - Enable interrupts with *ei*
    - Create section and label for VBlankInterrupt
    - Use *reti* to return from interrupt and keep interrupt enabled

Done:

```
ld a, [rLY]
cp 144
jr c, NotInVBlank

call IncrementCounter
```

NotInVBlank:

```
jr Done
```

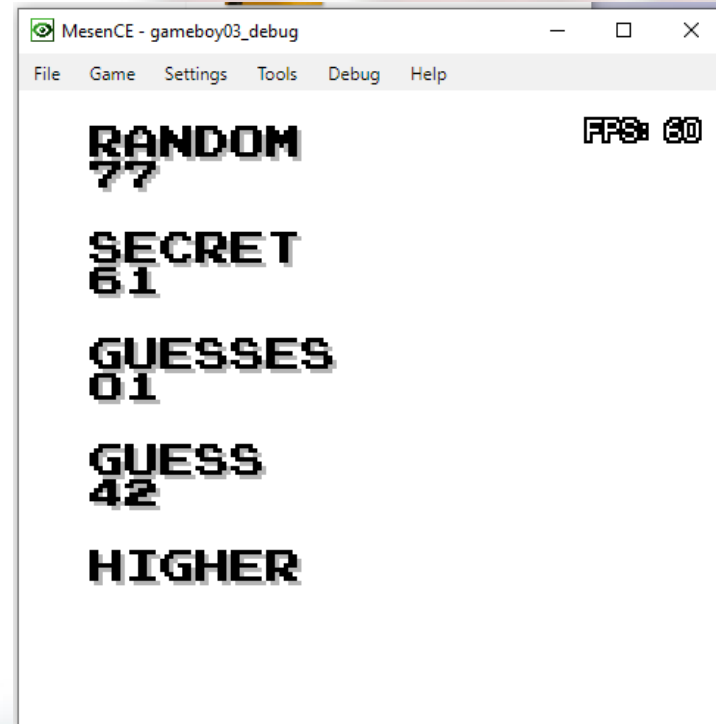
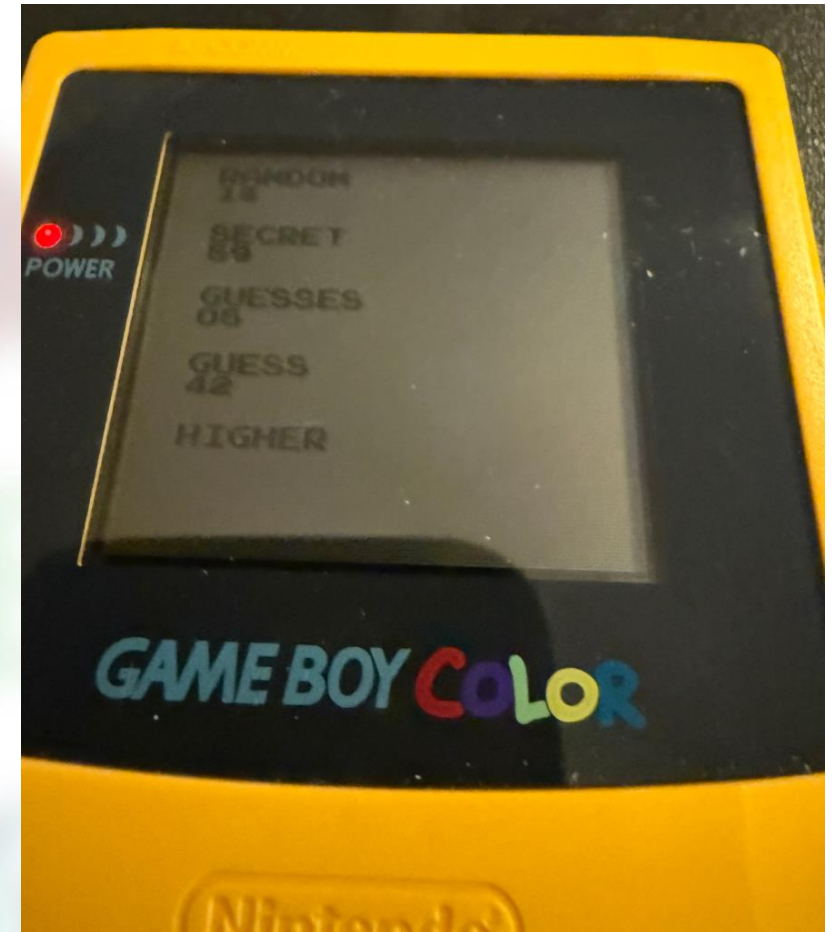
# Number Guessing Game

```
;store guess result
ld hl, wSecretNumDigitTens
ld a, [wGuessNumDigitTens]
cp a, [hl]
jp z, CheckGuessCorrect
jp c, CheckGuessLower
jp CheckGuessHigher

CheckGuessCorrect:
ld a, 0
ld [wResult], a
jp CheckGuessDone

CheckGuessHigher:
ld a, 1
ld [wResult], a
jp CheckGuessDone

CheckGuessLower:
ld a, 2
ld [wResult], a
jp CheckGuessDone
```



## Assembly for

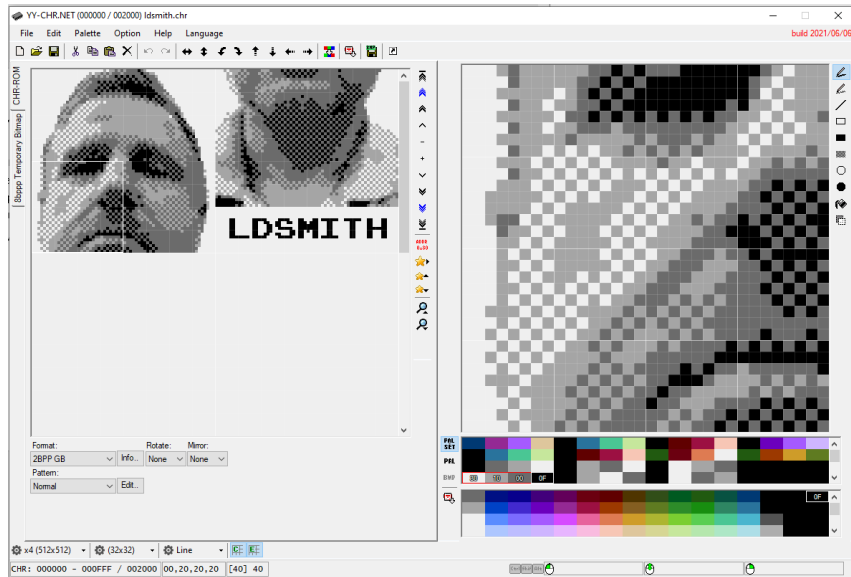
```
if num_guess == num_secret:
    result = 0 #equal
elif num_guess < num_secret:
    result = 2 #lower
else:
    result = 1 #higher
```

- if guess is lower, then display "higher" to the player. if the guess is higher, then display "lower" to the user
- first check tens digit, then check ones digit

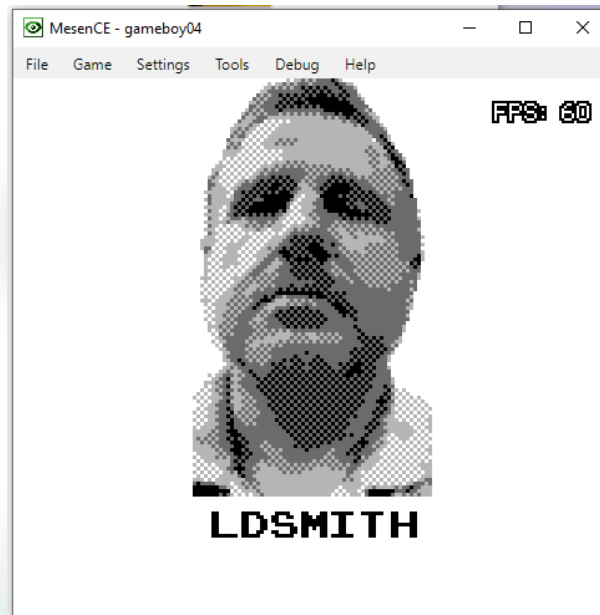
**KNOX**  
**GAME**  
**DESIGN**

# Photo Graphics

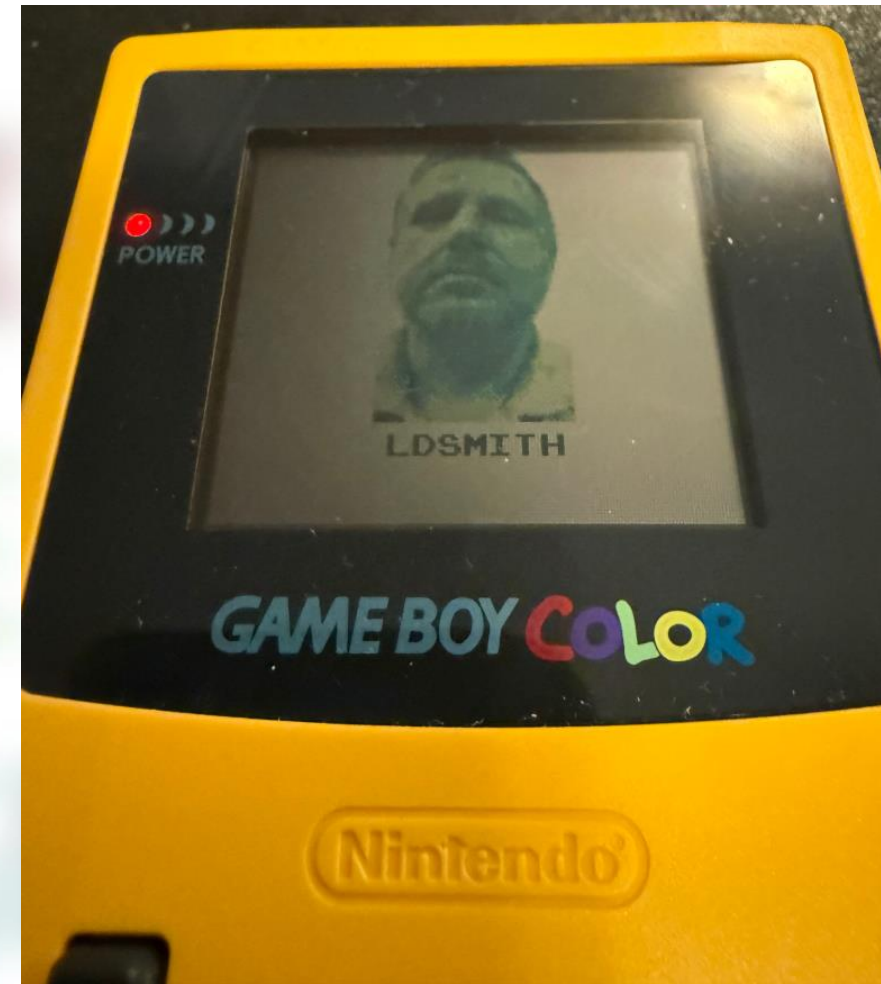
- Similar to NES photo graphics
  - Convert to grayscale
  - Convert to indexed palette, 7 colors
  - Make checker pattern for 3 in between colors
- Can only use top 8 rows from YY-CHR
  - split tall images



paste into YY-CHR and save



running on emulator  
(MesenCE)

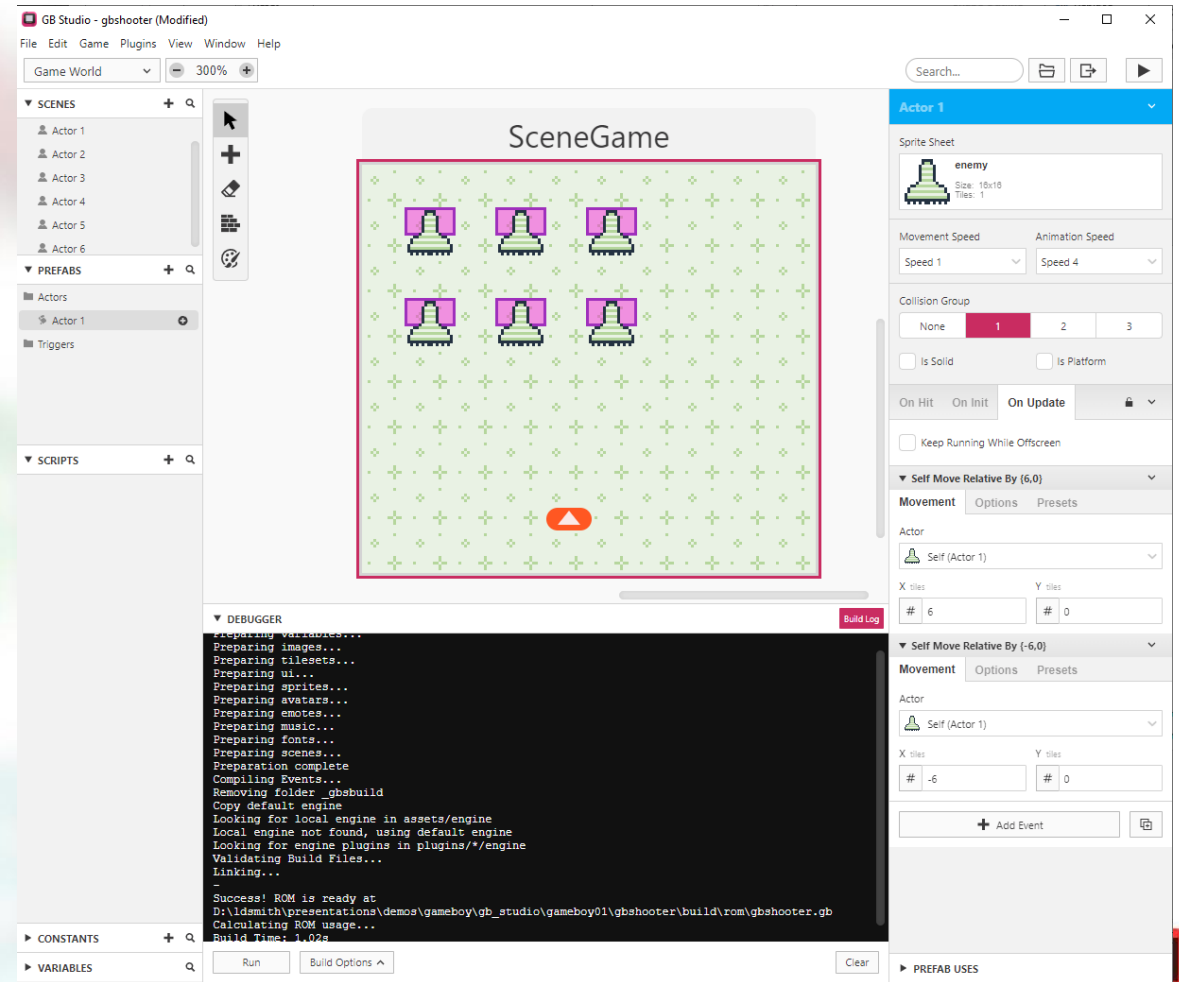


running on hardware  
(GameBoy Color)

**KNOX**  
**GAME**  
**DESIGN**

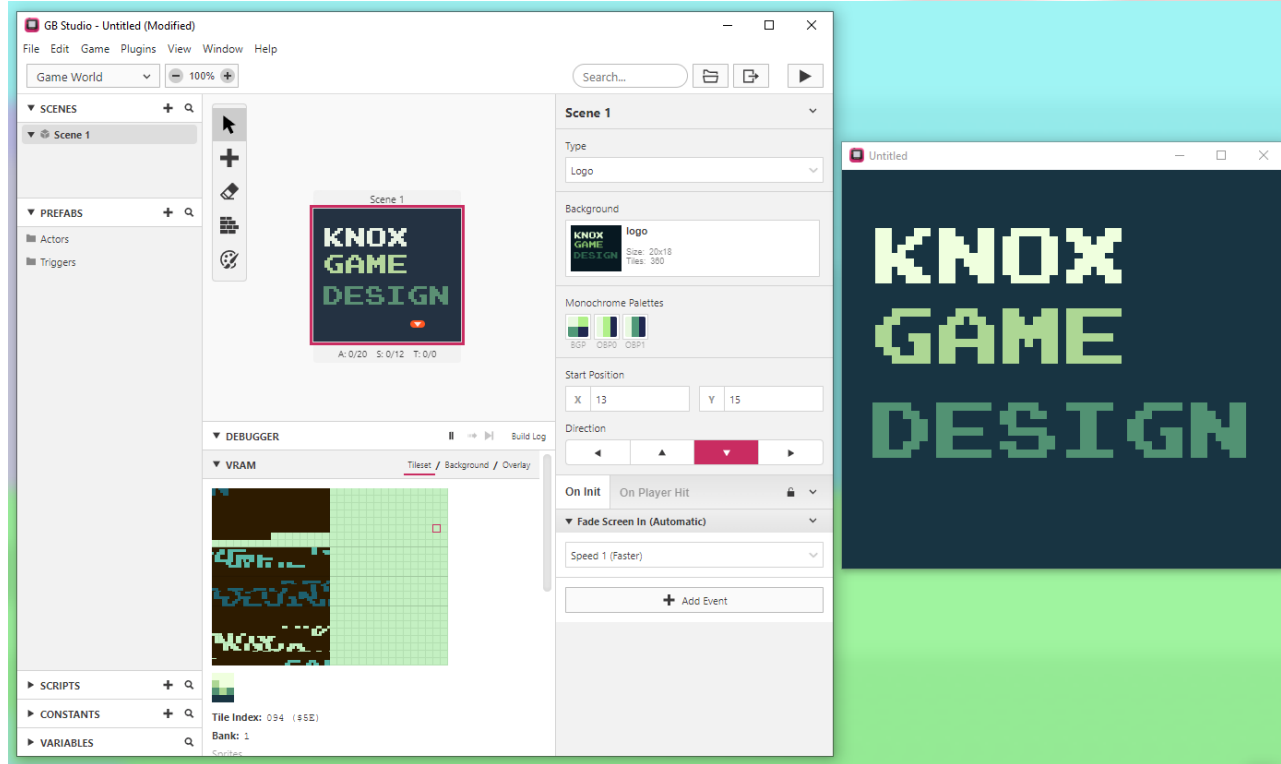
# GB Studio (IDE)

- Visual IDE editor
- Example starter template project or start from scratch
- Scenes, Prefabs, etc (similar to Unity, GameMaker, and Godot)
- Images
  - sprites / backgrounds folder in project directory
  - add through system file browser
  - PNG format. 4 index colors
  - Can use Gimp to create / edit



GAME  
DESIGN

# GB Studio – adding a logo screen

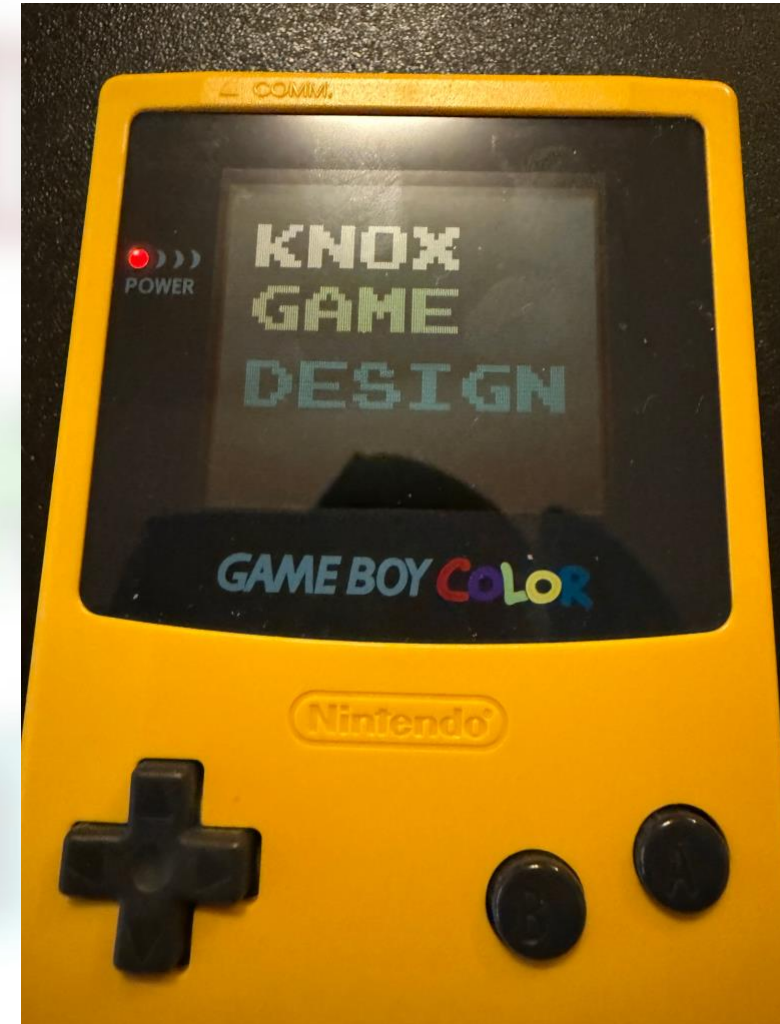


- Copy PNG file to <project>/assets/backgrounds
- Change scene type to Logo
- Test with *Start Debugger* or *Game > Run with Debugging*

**KNOX**  
**GAME**  
**DESIGN**

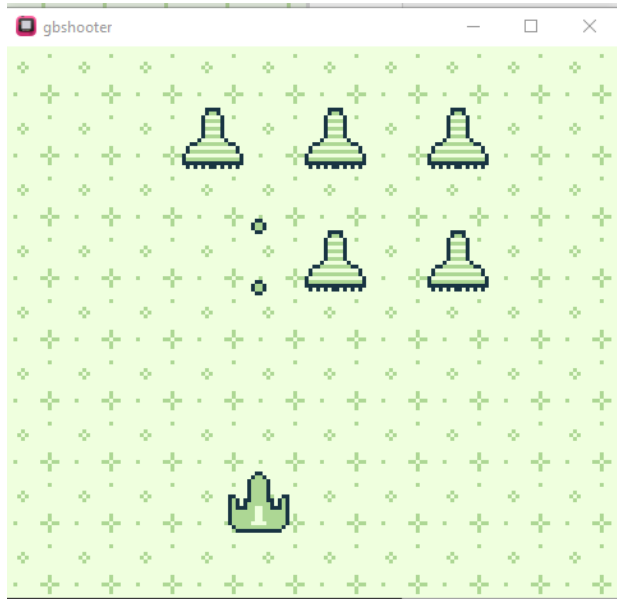
# GB Studio – build game

- Game > Export As > Export ROM
  - game is at <project>/build/rom
  - default build is *untitled.gb*
- Can play with
  - internal emulator
  - external emulator (Gambatte, mGBA, RetroArch, etc)
  - real hardware (Everdrive + GameBoy, GameBoy Color / Analogue Pocket)

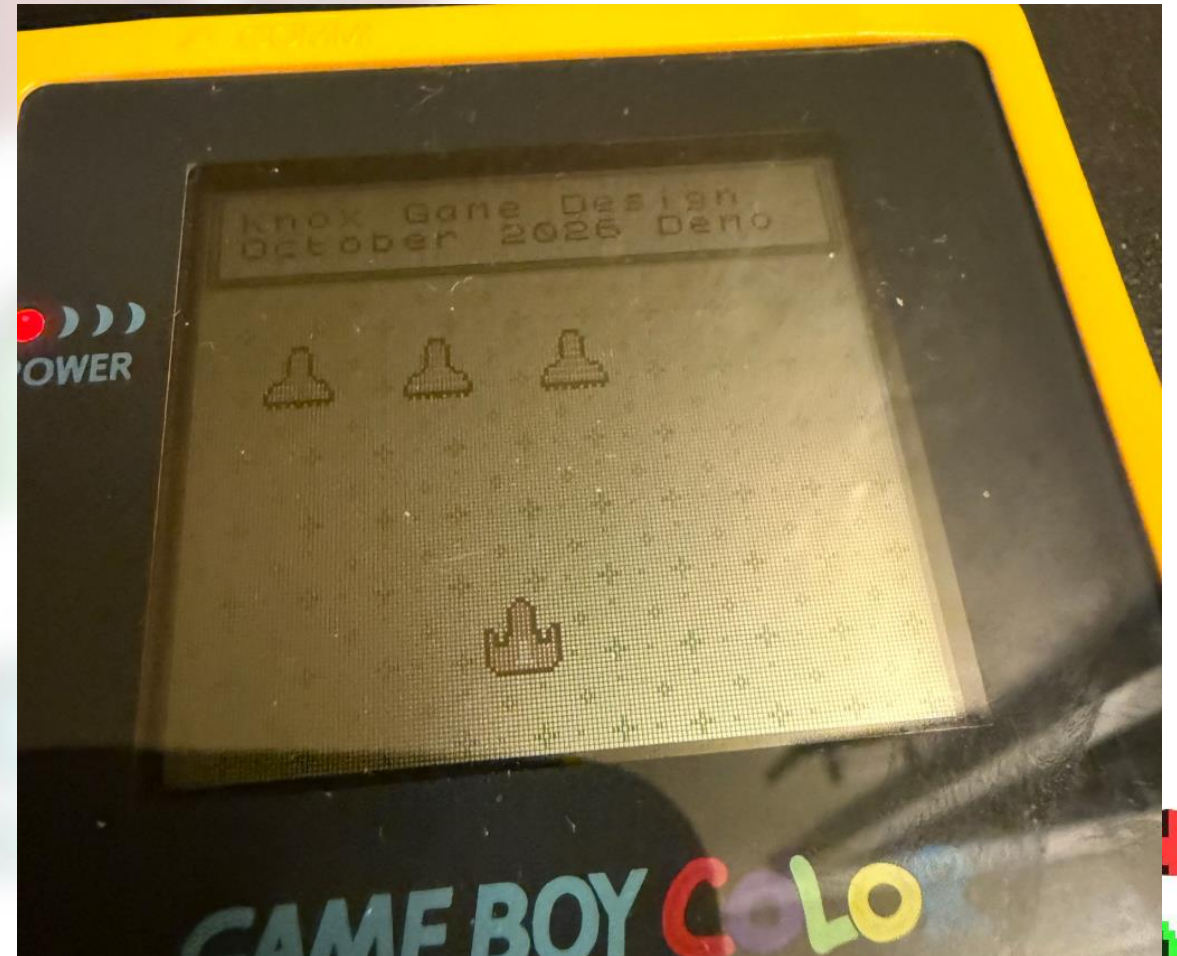


**KNOX**  
**GAME**  
**DESIGN**

# GB Studio – simple shooter



- Game objects: Player, Enemy (prefab), Bullet
- Enemy Movement: Right 6 tiles, Left 6 tiles
- Attach script to button (A)
  - Launch projectile event
  - Collide with group 1
  - Set source direction up
- Enemy: Collision group 1
  - On Hit: flicker then disable



KNOX  
GAME  
DESIGN