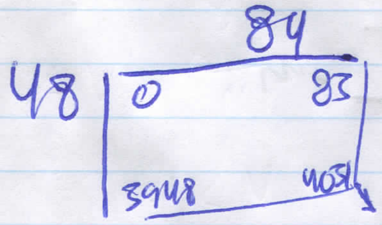
256K bit



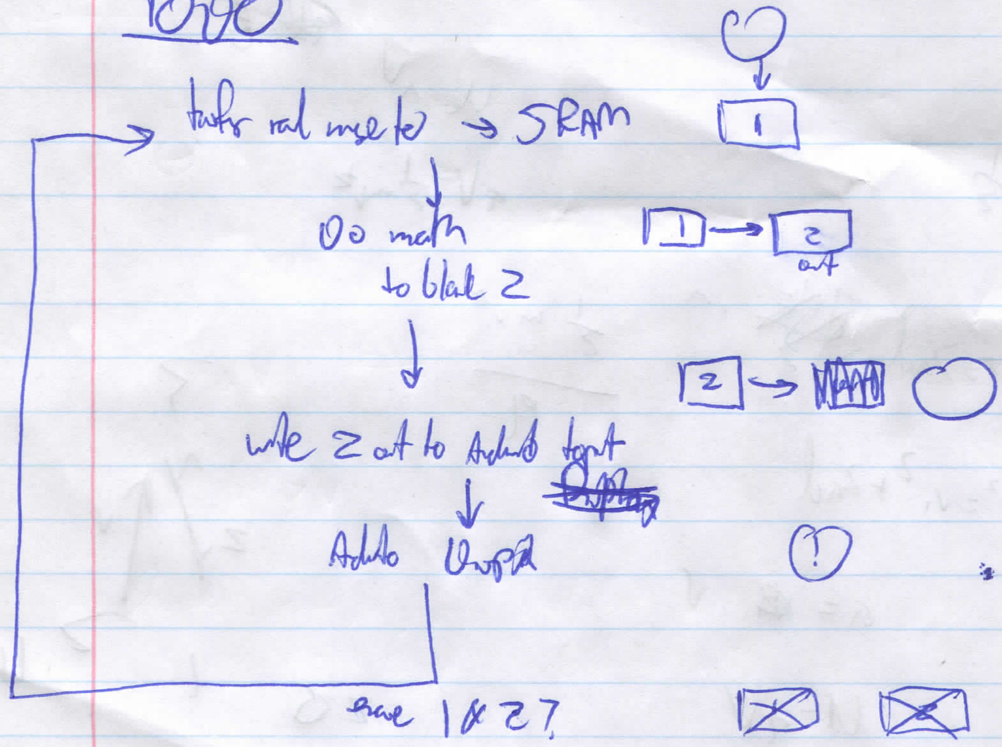
DONE

→ random generation of pixels

other

button interrupt etc

TODO


transfer real image to → SRAM

Do math to block 2

write 2 out to Arduino format

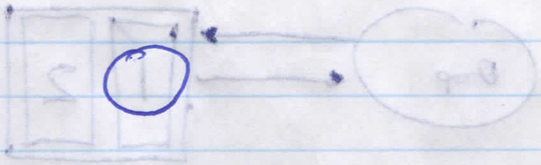
~~[illegible]~~

Arduino Output

save 1 & 2?

Map

Onplay random



Transfer to SRAM

Calculate + out to

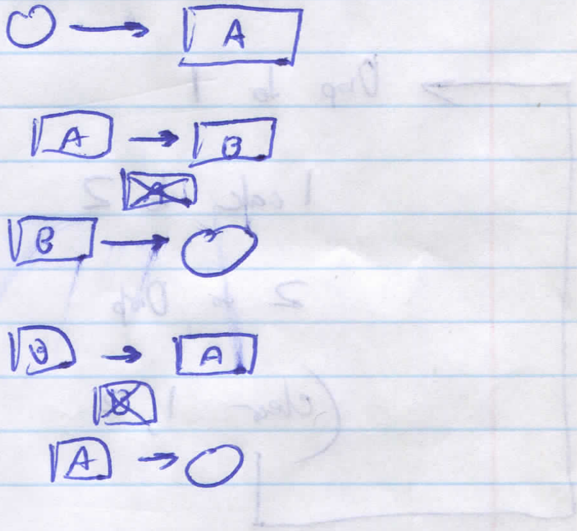
clear A

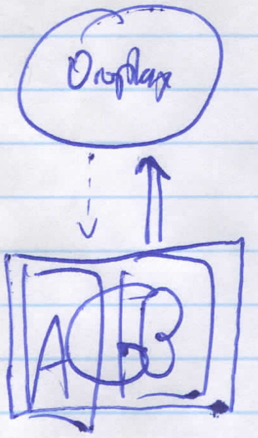
out to adrv

B to A

clear B

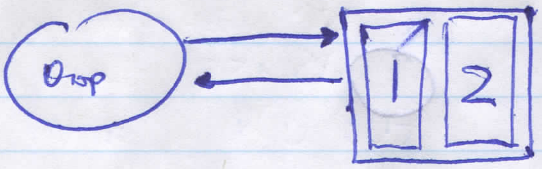
out to Ardut

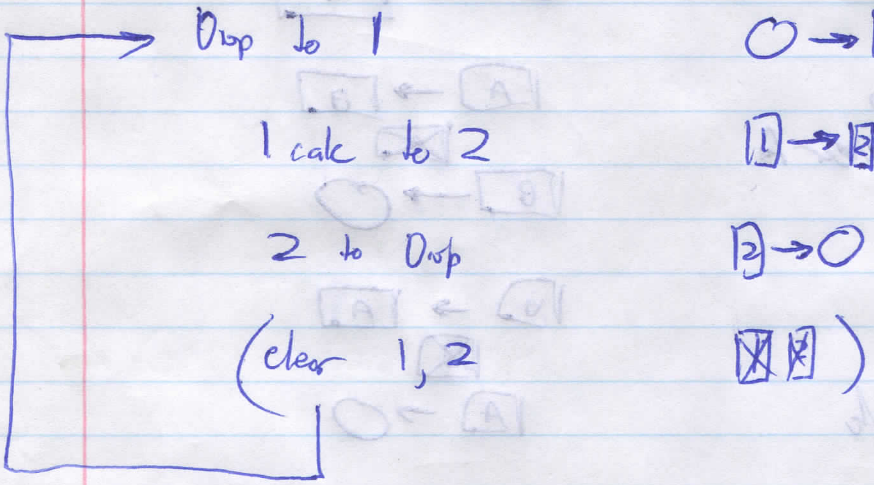




Onplay prints once to SRAM.
SRAM will continually out put the memory

NOT efficient...





Disp to 1 ○ → 1

1 calc to 2 1 → 2

2 to Disp 2 → ○

(clear 1, 2 ☒ ☒)

much simpler

info about 23K256

Arduino SRAM : ~~2048 bytes~~ 2 kilobyte =
2048 byte = 16 384 bits

256k =
1024 pages of 32 bytes / 256 bits 32 kilobyte
1024 · 256 = 262 144 bits
1024 pages of "32 words" of 8 bits

CS active low
HOLD → active low (keep HIGH)

SI: STATUS R: 00000101 W: 00000001

byte mode 00 00 0000
Page 10 00 0000
seq 01 00 0000

SI: Read 0000 0011 Write 0000 0010

Addresses: 0000 → 7FFF
0111 1111 1111 1111
↓
[illegible]
32767 words.

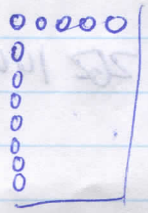
MSB LSB always
7 → 0

< Understand Arduino SPI

✓ - understand 23k 256

? ← understand PCD 8054

_ we draw pixel?



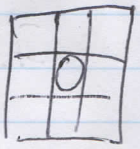
Get test code working

↓

figure out how to count lines information

RULES:

<2 neighbors,	dies
2-3	live
3>'	dies
3	live



1	X
2	✓
3	+
4	X
5	X
6	X
7	X
8	X
~~A~~	~~X~~

32767

0000 → 7FFFF

1 2

0 → 4031 4032 → 8063

0000 → 0FBF 0FC0 → ~~0~~ F7F

0 → 4031 4032 → 8063

0 → 8063 8064 → 16127

✓ t000

✓ screen → SPI

✓ – SPI → Screen

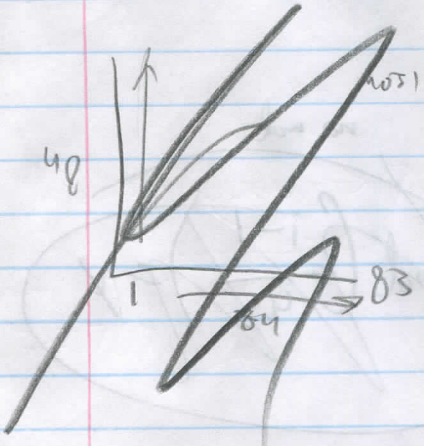
~~(rad_x, rad_y)~~

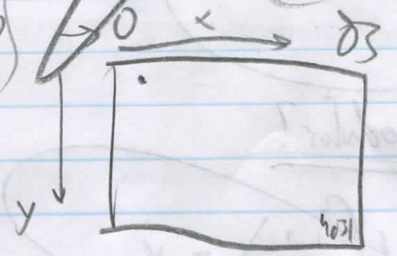
~~Addr = (rady − 1) · 84 + radx − 1~~

(8, 26) → 829

(9, 20)

0





0 1010 100

(x, y)

addr = ~~(y − 1) · 84 + x − 1~~

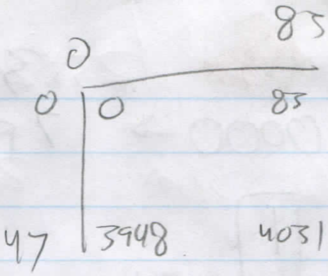
y · 84 + x

Adr: 0 → 4031

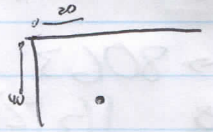
i: 1–4032

$i = 84y + x$

$i - 1 = 84y + x$



20, 40



$84(40)+20$ (+1)

~~addr = 3380~~

337az

addr = 3381

$3380 = 84y + x$

$x = \frac{i-1}{84y}$

3380 → 20

$y: \frac{i-1-x}{84}$

admo modulos?

$\text{mod}\left(\frac{i-1}{84}\right) = x$

$\text{mod}\left(\frac{3380}{84} =\right) : 20$

no nal

$\text{mod}\left(\frac{i-1}{48}\right) \neq y$

~~0 – 2017~~

~~2018 → 4033~~

0 → 2016

2017 → 4033

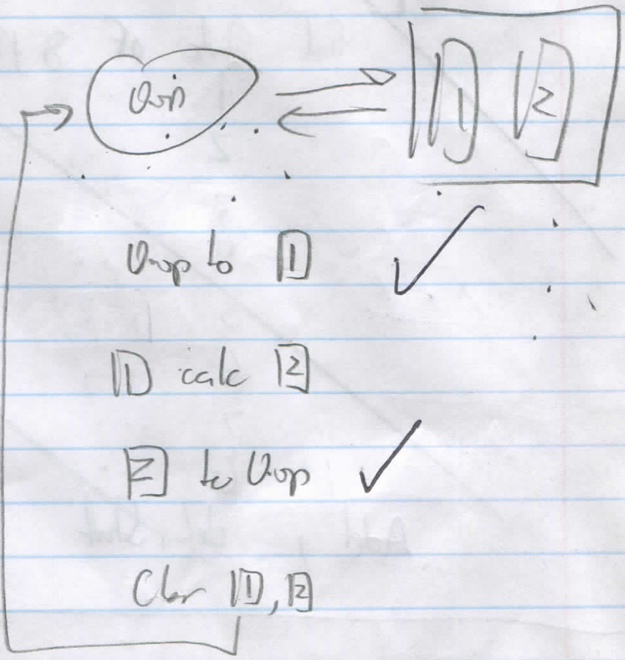
Screen -> SPI ✓

~~SPI~~ SPI -> Screen ✓

calculations ???

1) i) 0 -> 4033

2) i) 4034 -> 8066



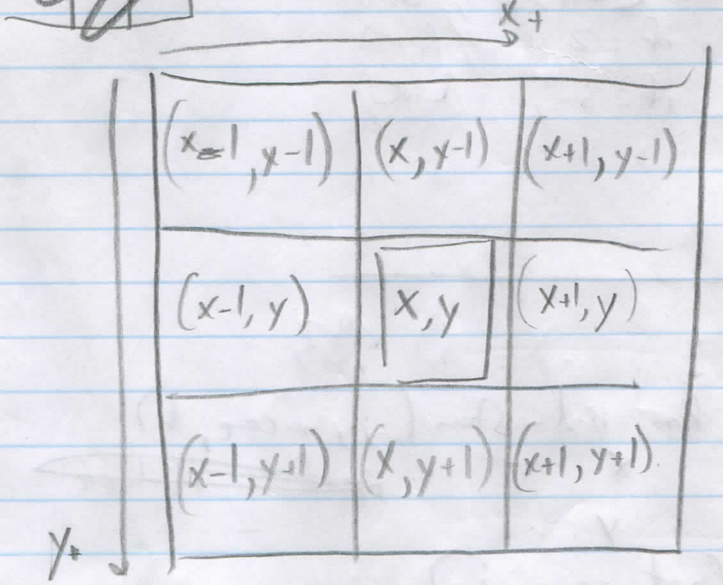
Disp to 1 ✓

1 calc 2

2 to Disp ✓

Clr 1, 2

0	X
1	X✓
2	✓
3	+
4	x
5	X
6	X
7	X
8	X



(x-1, y-1)	(x, y-1)	(x+1, y-1)
(x-1, y)	x, y	(x+1, y)
(x-1, y+1)	(x, y+1)	(x+1, y+1)

edge cases ??

~~loop[~~

~~find State of 8 thns~~

~~1 . . .~~

~~2 . . .~~

~~3 . . .~~

~~4 . . .~~

~~5 . . .~~

~~6 . . .~~

~~7 . . .~~

~~8 . . .~~

~~Add , detm State~~

```
loop[ State 1 + State 2+ ... State[8]
        if <2||>3 , dal
          if : 2  , A=A
          if=3 ,    Live
    ]
A ++
```

States:

```
    SpiRam, read-Stream ( i, mcore, 1)
          x . -
          y . . .
- $(Get new : )
      - new x
      - new y              j = 84y + x
    SpiRam, read-Stream (j, mcore, 1);
```

~~[illegible] +1, -1 etc~~

3121 ↓

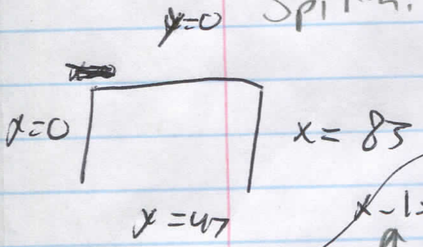
(13,37)

(14,37)

→ 3122

84(37)+14 -

SpiRam.read-Stream (84y+x, mem, 1)



x-1=83 ↑ y-1=47 ↑

edge cases : x = 0, y=0

x = 83, y = 47

↓ x+1=0 ↓ y+1=0

0,47

83,0

read (x,y)

↓

if x=0

~~ x Shift

if x = 83

~~ x Shift

if y=0

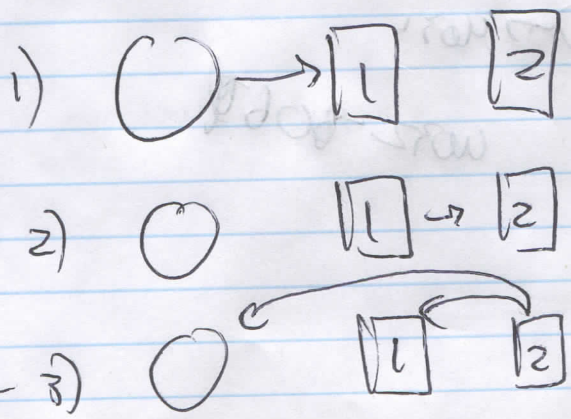
~~ y Shift

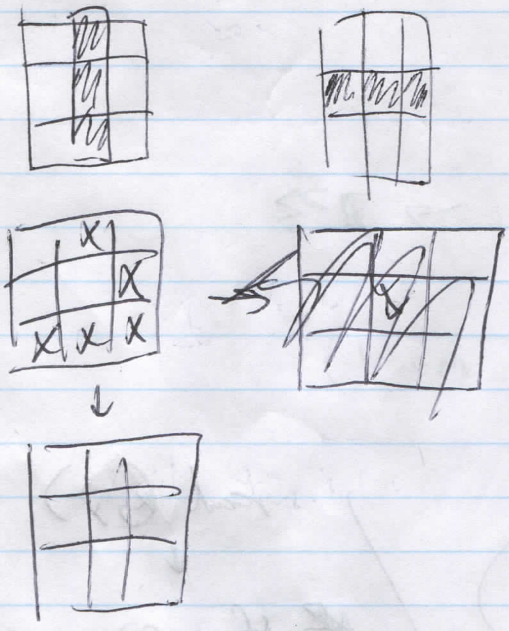
if y=47

~~ y Shift

Spi reading Shifts

A → A





potential errors: - neighbors aren't located properly

- counting neighbors wrong?
- wrong rules?
- bad transfer $\boxed{2} \rightarrow \boxed{1}$

X: 0 → 83

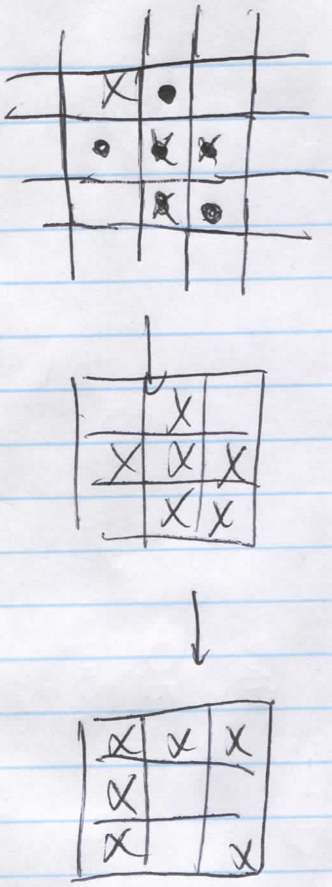
Y: 0 → 43

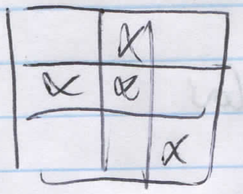
0 → 4031

4032 - 8064

0, 100, ++

→0] (100)
→99





SOMETHING IS FUCKING THE ALGORITHM

0, 4032, ++
→0] (4032)
→4031

4032, 8064, ++
4032]
8063

Number issues... X no...

YOU DONT OBSERVE EDGE CASES ??

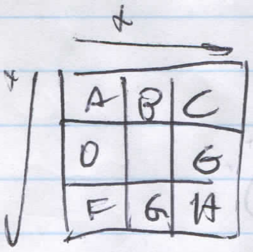
everything kep shifting left...
↓ Storage?
neighbors?

possible paths

- how you count neighbors
- rules / conditions
- finding neighbors

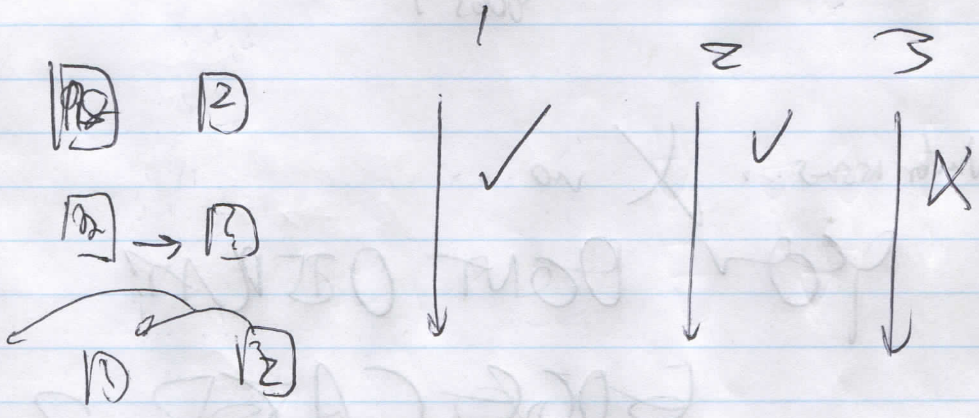
Geography.

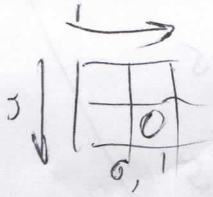
rules don't seem to apply?



A	B	C
0		G
F	G	1A

0 → 4032 → 8064



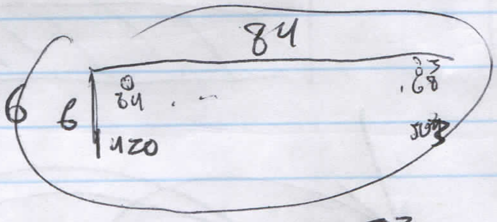


i -> 84 | 0 -> 83
j -> 6 | 0 -> 5

```
if (i>0)
    neighbor[WEST] = matrix[map((i-1, j))];
        map(i, j) -> (i + LCDWIDTH · j)
```

84 above LCDWIDTH

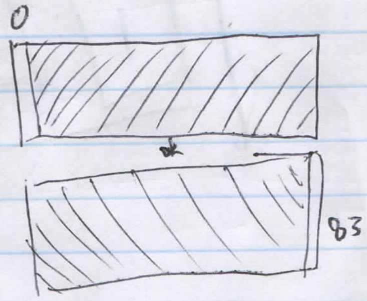
matrix is 84x6



504 total

0->503

West



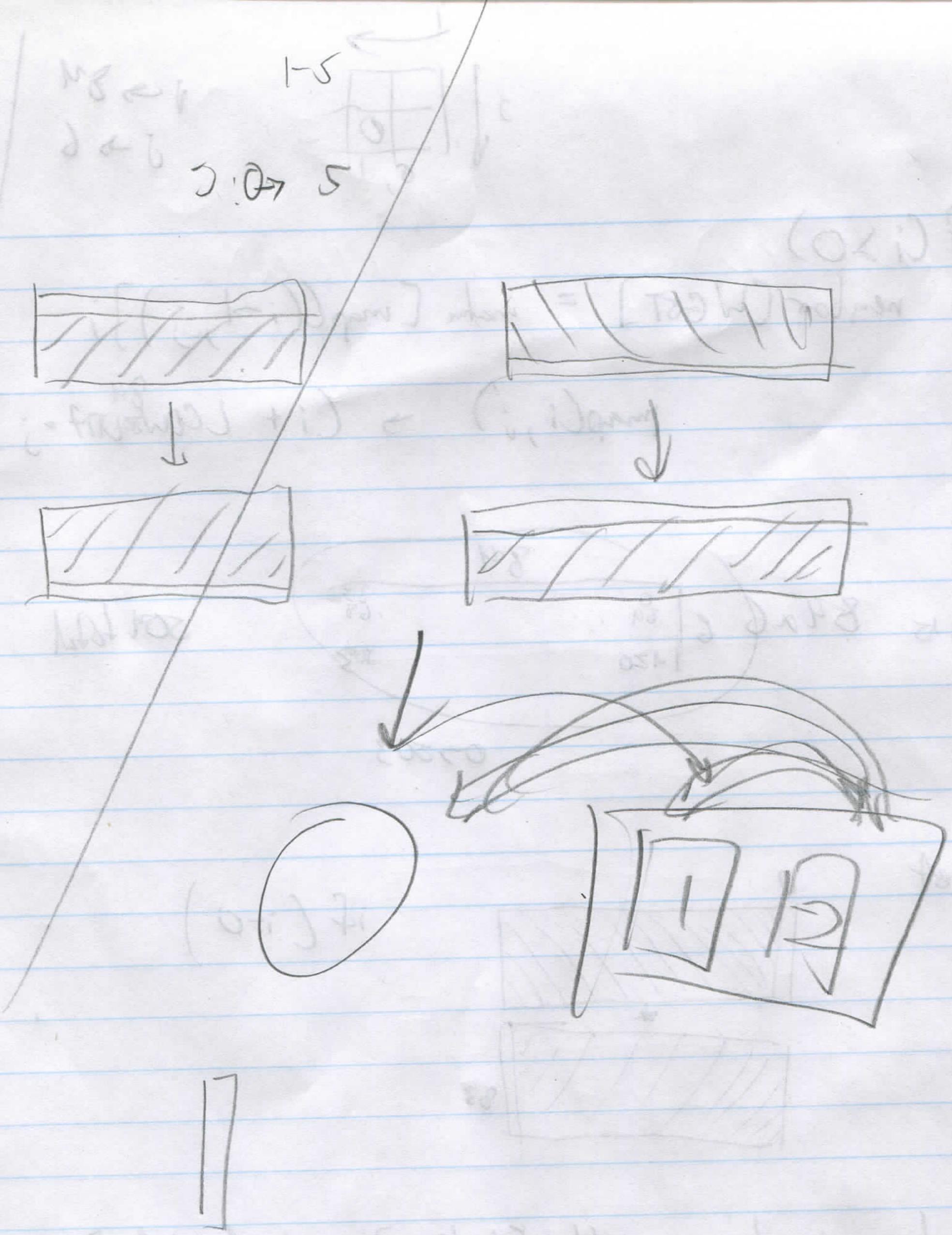
if (i=0)

What does ~~matrix~~ neighbor[WEST] look like?
add a print + delay

84-3

neighbor[WEST]
↓
in memory?
↓
Do addition to add matrix[map(83, j)]

0-26


1-5
Ↄ:0→ 5
1
2

arduino holds 8 pixels at a time

looks @ 8 other groups of 8 pixels.

```
// Nokia 5110 LCD-Display (SIZE_HxSIZE_V Bildpunkte)
// TSJWang
// February 15, 2017
// Conway's game of life using Arduino Nano and a Nokia 5110 Display
// a button on reset for initialization and a button on pin 2 for lighting.
/**modifying this code so it can do wrap world**/

#include <Adafruit_GFX.h>
#include <Adafruit_PCD8544.h>

#define LCD_WIDTH 84
#define LCD_HEIGHT 48
#define LCD_HEIGHT8 (LCD_HEIGHT >> 3)
```

01010100

00110000

48 shift right 3

00000110 = 6

some functions

```
#define map(i, j) (i + LCD_WIDTH*j)
#define map_x(i, j, k) (i)
#define map_y(i, j, k) (j*8+k)
#define map_i(x, y) (x)
#define map_j(x, y) (y/8)
#define map_k(x, y) (y%8)

#define for_i for (int i = 0; i < LCD_WIDTH; i++)
#define for_j for (int j = 0; j < LCD_HEIGHT8; j++)
#define for_k for (int k = 0; k < 8; k++)
#define for_n for (int n = 0; n < 8; n++)
```

84 0 -> 83

6 0 -> 5

```
#define for_x for (int x = 0; x < LCD_WIDTH; x++)
#define for_y for (int y = 0; y < LCD_HEIGHT; y++)
#define for_y_a for (int y_a = y - 1; y_a <= y + 1; y_a++)
#define for_x_a for (int x_a = x - 1; x_a <= x + 1; x_a++)
```

84

48

-1, 0, 1

```
#define bit_read(matrix, i, j, k) bitRead(matrix[map(i, j)], k)
#define bit_set(matrix, i, j, k) bitSet(matrix[map(i, j)], k)
#define bit_clear(matrix, i, j, k) bitClear(matrix[map(i, j)], k)

#define bit_set_xy(matrix, x, y) bit_set(matrix, map_i(x, y),  map_j(x, y), map_k(x, y))
#define bit_read_xy(matrix, x, y) bit_read(matrix, map_i(x, y),  map_j(x, y), map_k(x, y))
#define bit_clear_xy(matrix, x, y) bit_clear(matrix, map_i(x, y),  map_j(x, y), map_k(x, y));
```

ste row

```
#define pixel_set(i, j, k) display.drawPixel(i, j*8+k, BLACK)
#define pixel_unset(i, j, k) display.drawPixel(i, j*8+k, WHITE)
#define pixel_map(i, j, k) display.drawPixel(i, j*8+k, bit_read(matrix, i, j, k) == 1 ? BLACK :
WHITE)

#define between(n, a, b) ((n >= a) && (n <= b))

#define NORTH 0
#define NORTHEAST 1
#define EAST 2
#define SOUTHEAST 3
#define SOUTH 4
#define SOUTHWEST 5
#define WEST 6
```

7	0	1
6		2
5	4	3

```
#define NORTHWEST 7

#define RAND_RANGE 255

// Software SPI (slower updates, more flexible pin options):
// pin 13 - Serial clock out (SCLK)
// pin 11 - Serial data out (DIN)
// pin 4 - Data/Command select (D/C)
// pin 3 - LCD chip select (CS)
// pin 2 - LCD reset (RST)
Adafruit_PCD8544 display = Adafruit_PCD8544(8, 7, 5, 4, 3);
const int lite = 6;  //control for this interrupt pin will be 2
const int pot = A7;

// interrupt stuff
const byte interruptPin = 2;
volatile byte state = LOW;

// debouncing values
long debouncing_time = 200; //Debouncing Time in Milliseconds
volatile unsigned long last_micros;                                    84
                                                                     6 matrix
                        84         6
unsigned char matrix[LCD_WIDTH * LCD_HEIGHT8] = {0};                   84
unsigned char matrix_new[LCD_WIDTH * LCD_HEIGHT8] = {0};             0 matrix new
unsigned char toggle = 0;

void initialize_matrix() {
  for_x {
    for_y {
      char rand = random(RAND_RANGE);
      if (between(rand, 0, RAND_RANGE / 2)) {
        bit_set_xy(matrix, x, y);
      }
    }
  }
}

// implemented using dubaiss' "neighbours XXX" algorithm
void evolve_matrix() {

  // new state
  unsigned char neighbour[8] = {0};
  unsigned char byte_cell;
  unsigned char byte_cell_new;
  unsigned char byte_cell_count;
  // calculate new state

  for_i {  84
    for_j {  6

      byte_cell = matrix[map(i, j)];

      /* * get each bit's neighbour one byte at a time * */
```

i→84

j→6

k→8

n→8

else
compare column 0 with
to col 83

i=0

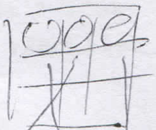
(i-1)%84=83

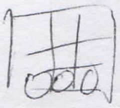
(i+1)%84=0

at least 1

```
/* east and west */
neighbour[WEST] = 0b00000000;
if (i > 0) {
  neighbour[WEST] = matrix[map((i - 1), j)];
}

neighbour[EAST] = 0b00000000;
if (i < (LCD_WIDTH - 1)) {
  neighbour[EAST] = matrix[map((i + 1), j)];
}

/* north */
neighbour[NORTH] = 0b00000000;
neighbour[NORTHEAST] = 0b00000000;
neighbour[NORTHWEST] = 0b00000000;
if (j > 0) {
  neighbour[NORTH] = matrix[map(i, j - 1)];
  if (i > 0) {
    neighbour[NORTHWEST] = matrix[map(i - 1, j - 1)];
  }
  if (i < (LCD_WIDTH - 1)) {
    neighbour[NORTHEAST] = matrix[map(i + 1, j - 1)];
  }
}
neighbour[NORTH] = (byte_cell << 1) | ((neighbour[NORTH] & 0b10000000) >> 7);
neighbour[NORTHEAST] = (neighbour[EAST] << 1) | ((neighbour[NORTHEAST] & 0b1000000) >> 7);
neighbour[NORTHWEST] = (neighbour[WEST] << 1) | ((neighbour[NORTHWEST] & 0b1000000) >> 7);

/* south */
neighbour[SOUTH] = 0b00000000;
neighbour[SOUTHEAST] = 0b00000000;
neighbour[SOUTHWEST] = 0b00000000;
if (j < (LCD_HEIGHT8 - 1)) {
  neighbour[SOUTH] = matrix[map(i, j + 1)];
  if (i > 0) {
    neighbour[SOUTHWEST] = matrix[map(i - 1, j + 1)];
  }
  if (i < (LCD_WIDTH - 1)) {
    neighbour[SOUTHEAST] = matrix[map(i + 1, j + 1)];
  }
}
neighbour[SOUTH] = ((neighbour[SOUTH] & 0b00000001) << 7) | (byte_cell >> 1);
neighbour[SOUTHEAST] = ((neighbour[SOUTHEAST] & 0b00000001) << 7) | (neighbour[EAST] >> 1);
neighbour[SOUTHWEST] = ((neighbour[SOUTHWEST] & 0b00000001) << 7) | (neighbour[WEST] >> 1);

/* calculate each bit of next gen */
byte_cell_new = 0b00000000;

for_k {
  byte_cell_count = 0;
  for_n {
    byte_cell_count += (neighbour[n] & 0b00000001);
  }
```

83

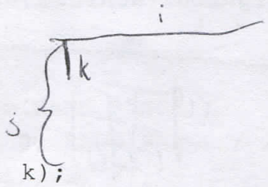




the limitation lays on the if loops

```
        byte_cell_new >>= 1;
        if ((byte_cell_count == 3) || ((byte_cell_count == 2) && (byte_cell & 0b00000001))) {
          byte_cell_new |= 0b10000000;
        }

        for_n {
          neighbour[n] >>= 1;
        }
        byte_cell >>= 1;
      }

      matrix_new[map(i, j)] = byte_cell_new;
    }
  }

  // apply new state
  for_i {
    for_j {
      matrix[map(i, j)] = matrix_new[map(i, j)];
    }
  }

}
```

```
void update_display() {

  uint8_t color;
  for_i {    89
    for_j {    6
      for_k {  8
        pixel_map(i, j, k);
      }
    }
  }
  display.display();
}
```



```
void setup()   {
  pinMode(lite, OUTPUT);
  pinMode(interruptPin, INPUT_PULLUP);
  pinMode(pot, INPUT);
  attachInterrupt(digitalPinToInterrupt(interruptPin), changelite, RISING);
  // random seed
  randomSeed(analogRead(0));
  // initialize display
  display.begin();
  // set contrast
  display.setContrast(56);
  // clears the screen and buffer
  display.clearDisplay();
  // initialize matrix
  initialize_matrix();
}


void loop() {
  update_display();
  evolve_matrix();
  delay(analogRead(pot));
}

/******************This function controls lighting********************/
void changelite(){
  if((long)(micros() - last_micros) >= debouncing_time * 1000) {
    state = !state;
    digitalWrite(lite, state);
    last_micros = micros();
  }
}
```