Recitation 03/15

Collision Maps and Complex Movement

Welcome! ^-^

Announcements

- **Spring break is next week!!!! :)**
- **Milestone 1** due **Friday (03/18)** at 11:59pm.
- **Lab08** released earlier today, due **Thursday (03/17)** at 11:59pm.
- **HW05** due **Friday (03/18)** at 11:59pm.

Today's agenda

1. Collision Maps
2. Complex Movement

Collision Maps

What is a collision map?

A collision map is a **bitmap**

- An array of palette indices on the map
  - Every other index of the palette is non-zero (or true, in C)
  - The first index of the palette is zero (or false, in C)
- Each index on the bitmap represents a pixel

The basic usage of a collision map is to create a black & white map to indicate where you can and can’t walk

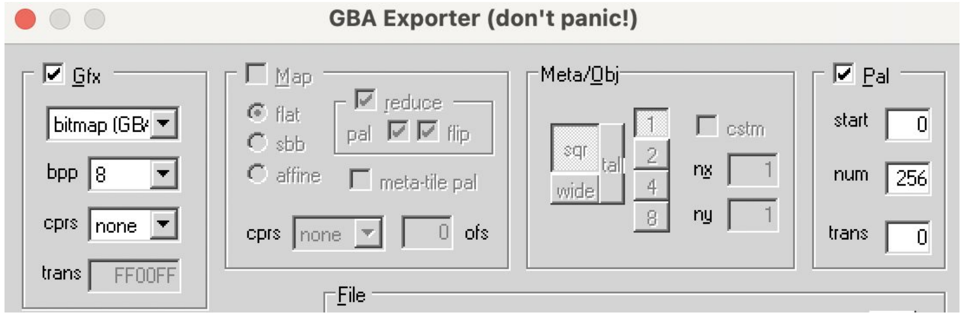
  - White area indicates walkable area

Exporting collision maps

Export settings:

- Bitmap (**not** tiles)
- 8bpp
- Pal checked

So just like a Mode 4 image!



GBA Exporter (don't panic!)

[✓] Gfx
bitmap (GB
bpp 8
cprs none
trans FF00FF

[ ] Map
[✓ flat] sbb affine
[✓] reduce
pal [✓] [✓] flip
[ ] meta-tile pal
cprs none 0 ofs

Meta/Obj
sqr wide tall
1 2 4 8
[ ] cstm
nx 1
ny 1

[✓] Pal
start 0
num 256
trans 0

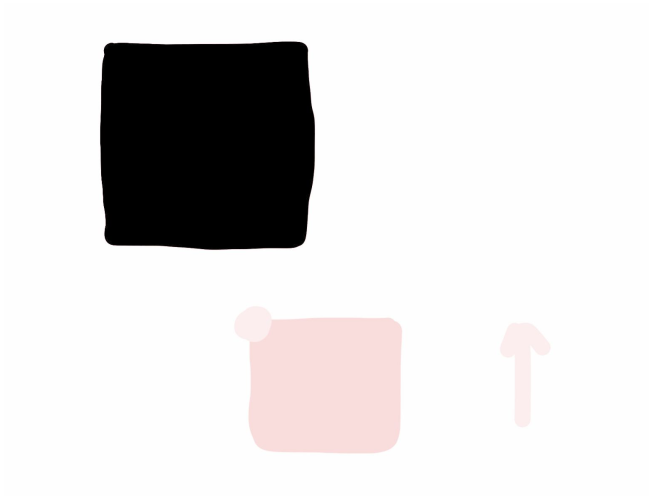
File

How to use them

When a sprite is moving:

- Check the bitmap @ that place on the screen (where the sprite is **trying** to move to)
- Where we check corresponds to the direction that we are moving in

Example: sprite moving up (part 1)



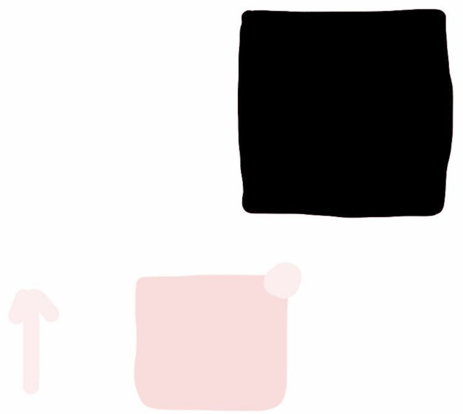
Since we are checking **specific** spots on our collision map, what position would we want to check to see if the **top left corner** of the sprite is colliding?

The top left pixel:

```
(sprite.worldCol, sprite.worldRow-1)
```

Note: this is assuming that `sprite.rdel == 1`

Example: sprite moving up (part 2)

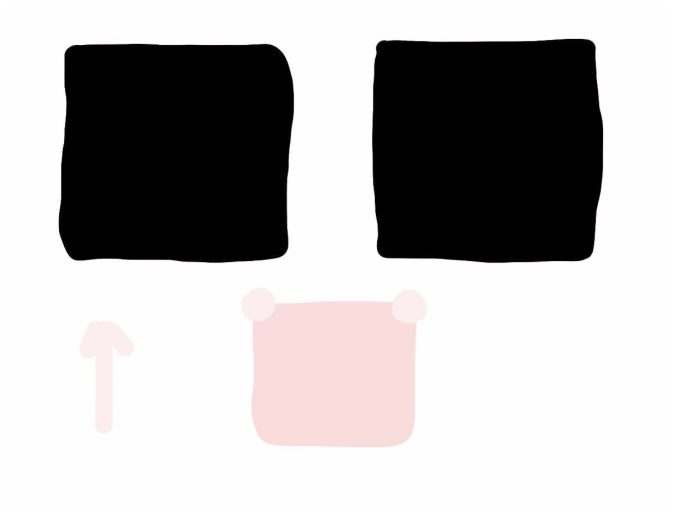


What if we had this scenario?

We would check the **top right** pixel:

```
(sprite.worldCol+sprite.width-1,
sprite.worldRow-1)
```

Example: sprite moving up (part 3)



What if we had this scenario?

Then we would check both the **top left** and **top right** pixels!

How to use them

When a sprite is moving:

- Check the bitmap @ that place on the screen (where the sprite is **trying** to move to)
- Where we check corresponds to the direction that we are moving in
- Look at **both** corners (of the sprite) in the direction we are trying to move
  - ↑: top 2 corners
  - ↓: bottom 2 corners
  - ←: left 2 corners
  - →: right 2 corners

Accessing a collision map

If the collision map is an array, how are we going to access an entry of it?

Treating the map as an unsigned char array, finding the location within the bitmap is simply:

```
    OFFSET(sprite's col, sprite's row, MAPWIDTH)
```

Where the row and col are the **location that the sprite is attempting to move to**, and MAPWIDTH is the **width of our collision map**.

Checking the color

We need to keep track of where different colors are in our palette!

Convention for this class is to let **black** be on **index 0**, and **white** on **index 1**.

Assuming that that's the case,

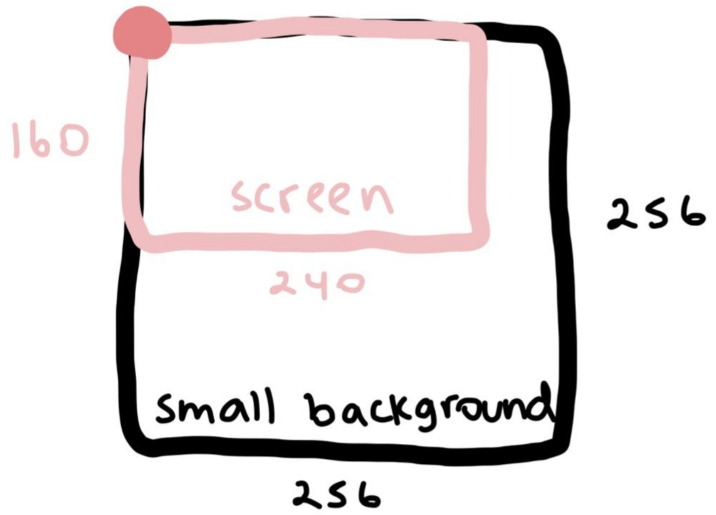
```
unsigned char* collisionMap = collisionBitmap; // you only have to do this once!
     if (BUTTON_HELD(BUTTON_UP) {
          if (collisionMap[OFFSET(col, row - rdel, MAPWIDTH)] &&
               collisionMap[OFFSET(col + width - 1, row - rdel, MAPWIDTH)]) {
                 row -= rdel;
          }
     }
```

`h0ff` and `v0ff`

What are they?

vOff is the vertical offset of *the screen in relation to the background*.
hOff is the horizontal offset of *the screen in relation to the background*.

What does that actually mean?

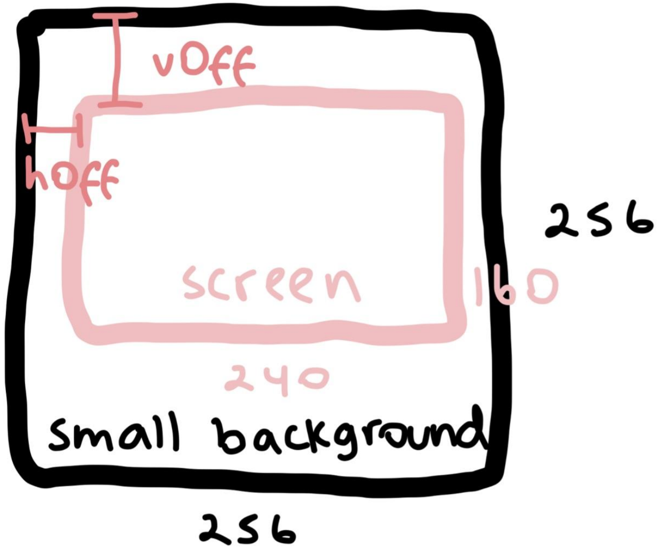


```
vOff = 0, hOff = 0
```

The screen is essentially a **window to the background**.

Note: When exporting backgrounds in Usenti, it pads it to the smallest valid dimension that your background fits into.

How do we use `hOff` and `vOff`?



```
vOff = 20, hOff = 10
```

Updating vOff and hOff is *necessary*, but **it doesn't actually update where the screen is**.

How do we update where the screen is?

```
#define REG_BG0HOFF (*(volatile unsigned short *)0x04000010)
#define REG_BG0VOFF (*(volatile unsigned short *)0x04000012)
```

```
REG_BG0HOFF = hOff;
REG_BG0VOFF = vOff;
```

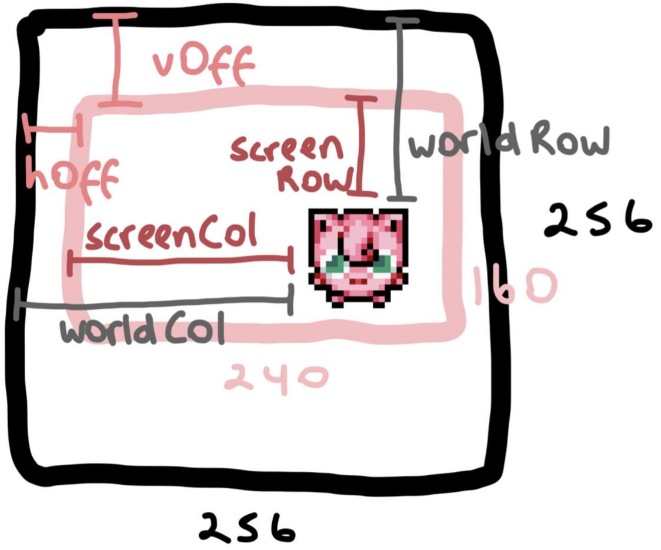
Note: These are **write-only registers**.

Complex Movement

What is complex movement?

Complex movement is when **our sprite's movement controls the position of the screen** on the background.

How does that work?



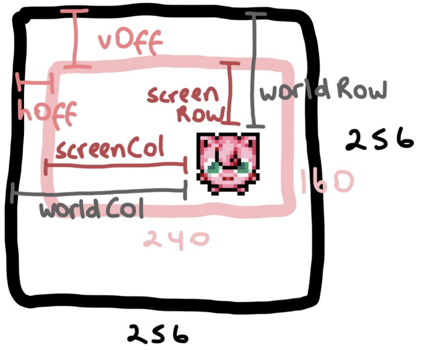
The OAM is looking to draw the sprite at `(screenCol, screenRow)`

```
screenCol = worldCol - hOff;
screenRow = worldRow - vOff;
```

There are **four cases** for complex movement!

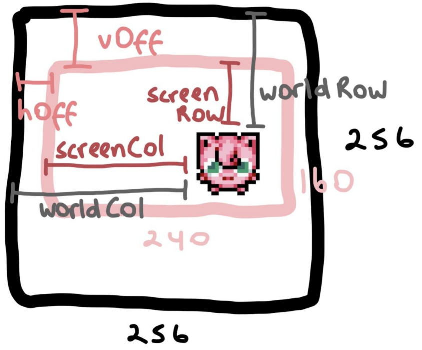
Sprite moving up

```
if BUTTON_PRESSED UP
    a)  can I decrement vOff safely?
            will vOff - 1 >= 0?
    b)  is my sprite halfway up the screen?
            (worldRow - vOff) <= (SCREENHEIGHT / 2)
    ➔   if **both** are true:
            vOff--;
```



Sprite moving down

```
if BUTTON_PRESSED DOWN
    a)  can I increment vOff safely?
            will vOff ++ < (MAPHEIGHT - SCREENHEIGHT)?
    b)  is my sprite halfway down the screen?
            (worldRow - vOff) > (SCREENHEIGHT / 2)
    ➔   if both are true:
            vOff++;
```



Where do complex movement checks go?

They are **nested within collision map checks**.

Can Jigglypuff move here based on the collision map?

    If yes, should I increment/decrement vOff/hOff?

```
if (button check)
    if (collision map check)
        move sprite //(sprite.col and sprite.row stuff)
        if (complex movement check)
            do complex movement //(vOff and hOff stuff)
```

Questions?

That's all for today!