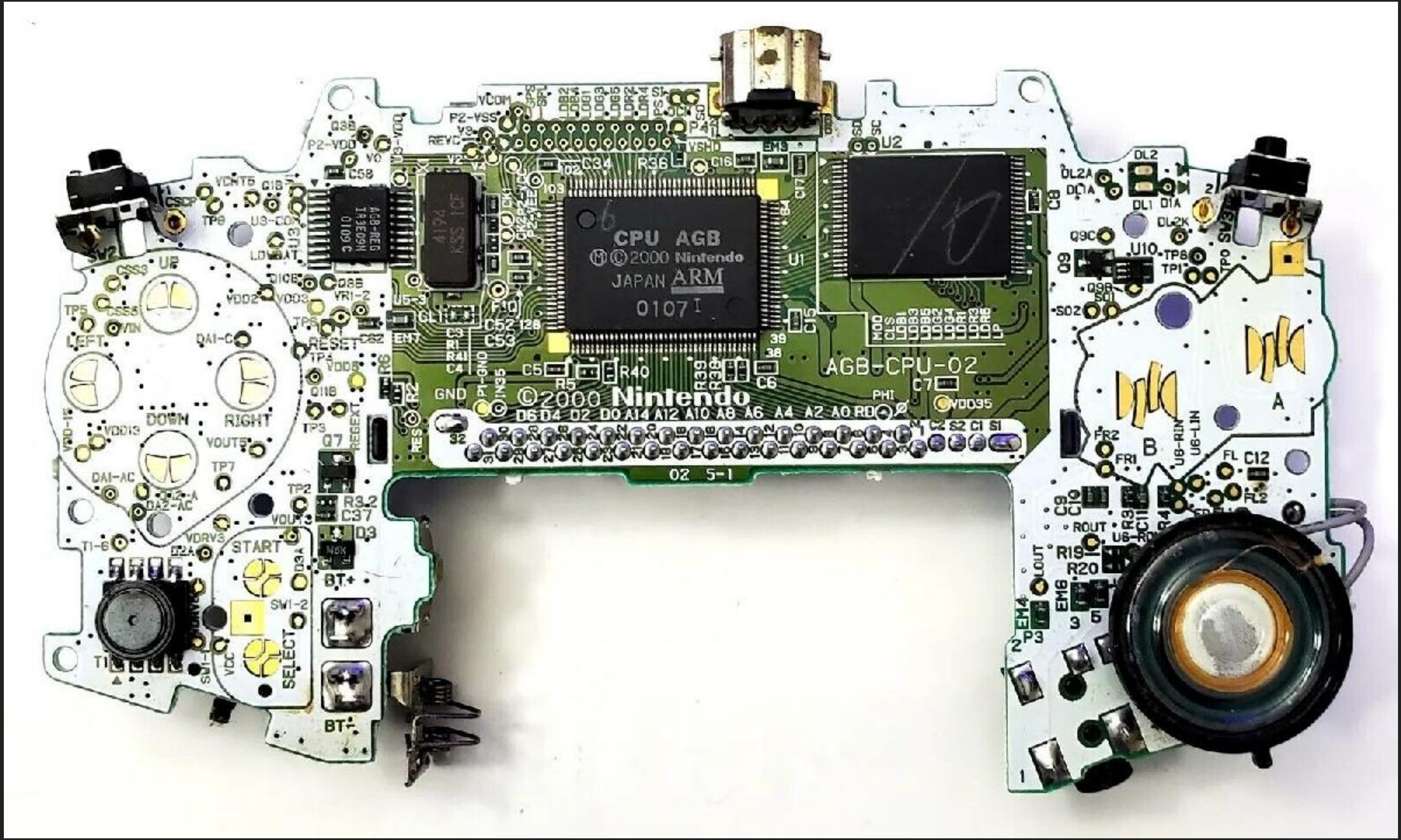
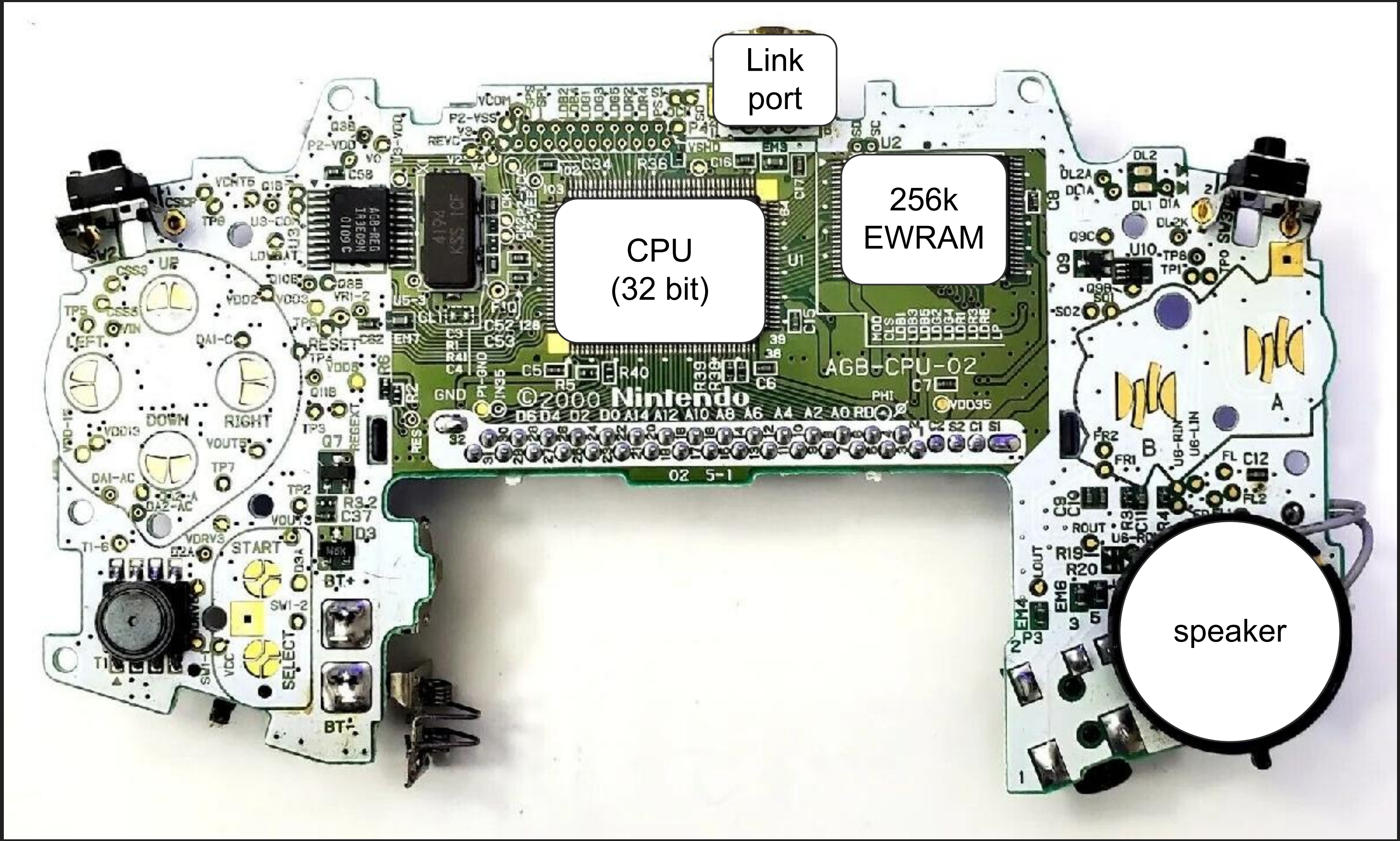
MAKING GAME BOY ADVANCE GAMES WITH GO

BRANDON ATKINSON

HARDWARE

Nintendo
POWER
GAME BOY ADVANCE
START
SELECT
B
A


CPU AGB
Ⓜ©2000 Nintendo
JAPAN ARM
0107 I
AGB-CPU-02
©2000 Nintendo
AGB-REG
UP
DOWN
LEFT
RIGHT
START
SELECT
A
B
RESET
BT+
BT-
GND
VCC
U1
U2
U3
U10
SW1
SW2
SW3
02 5-1



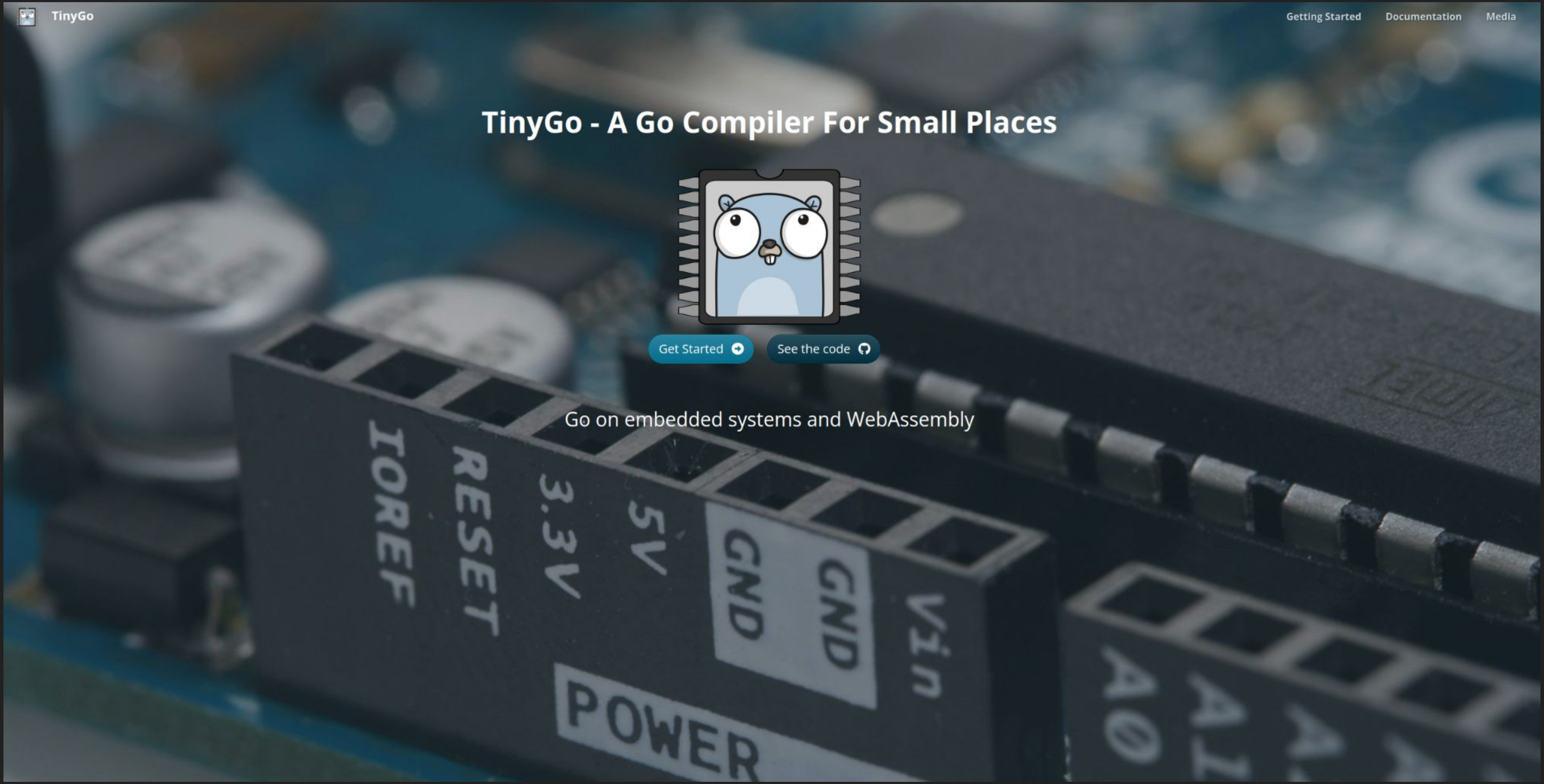
Link port

256k EWRAM

CPU (32 bit)

speaker

REFERENCES

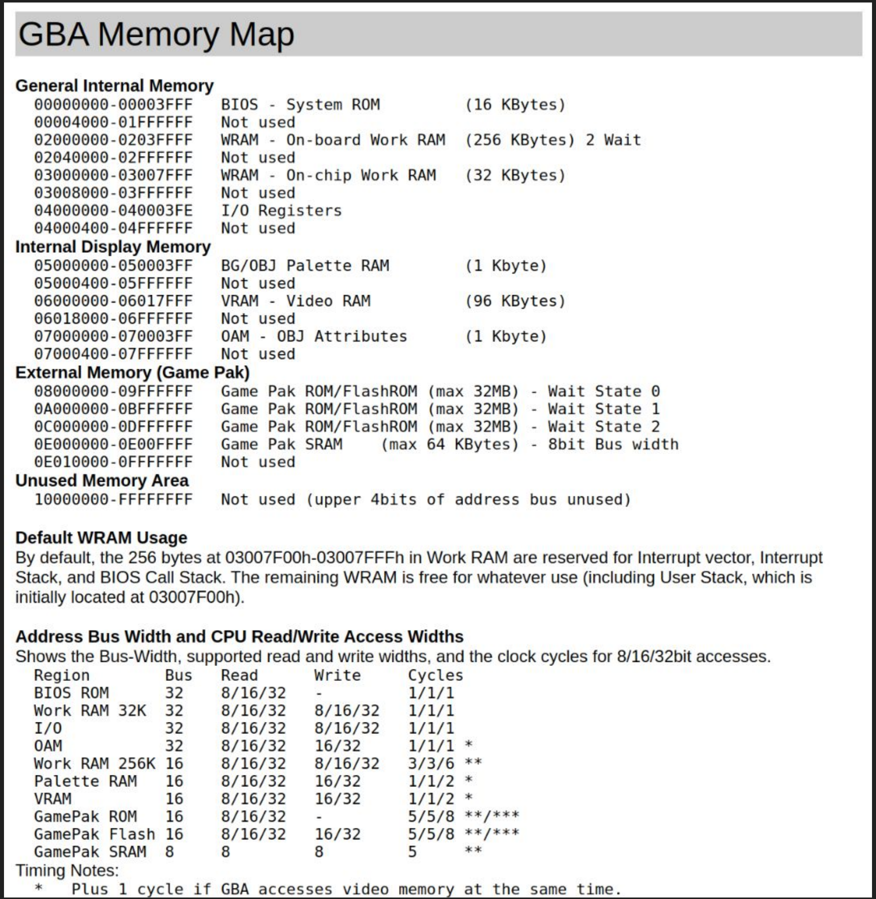


TinyGo - A Go Compiler For Small Places

Go on embedded systems and WebAssembly

GBATEK
https://problemkaputt.de/gbatek.htm

TONC
https://www.coranac.com/tonc/text/toc.htm



GBA Memory Map

General Internal Memory

```
  00000000-00003FFF   BIOS - System ROM         (16 KBytes)
  00004000-01FFFFFF   Not used
  02000000-0203FFFF   WRAM - On-board Work RAM  (256 KBytes) 2 Wait
  02040000-02FFFFFF   Not used
  03000000-03007FFF   WRAM - On-chip Work RAM   (32 KBytes)
  03008000-03FFFFFF   Not used
  04000000-040003FE   I/O Registers
  04000400-04FFFFFF   Not used
```

Internal Display Memory

```
  05000000-050003FF   BG/OBJ Palette RAM        (1 Kbyte)
  05000400-05FFFFFF   Not used
  06000000-06017FFF   VRAM - Video RAM          (96 KBytes)
  06018000-06FFFFFF   Not used
  07000000-070003FF   OAM - OBJ Attributes      (1 Kbyte)
  07000400-07FFFFFF   Not used
```

External Memory (Game Pak)

```
  08000000-09FFFFFF   Game Pak ROM/FlashROM (max 32MB) - Wait State 0
  0A000000-0BFFFFFF   Game Pak ROM/FlashROM (max 32MB) - Wait State 1
  0C000000-0DFFFFFF   Game Pak ROM/FlashROM (max 32MB) - Wait State 2
  0E000000-0E00FFFF   Game Pak SRAM    (max 64 KBytes) - 8bit Bus width
  0E010000-0FFFFFFF   Not used
```

Unused Memory Area

```
  10000000-FFFFFFFF   Not used (upper 4bits of address bus unused)
```

Default WRAM Usage

By default, the 256 bytes at 03007F00h-03007FFFh in Work RAM are reserved for Interrupt vector, Interrupt Stack, and BIOS Call Stack. The remaining WRAM is free for whatever use (including User Stack, which is initially located at 03007F00h).

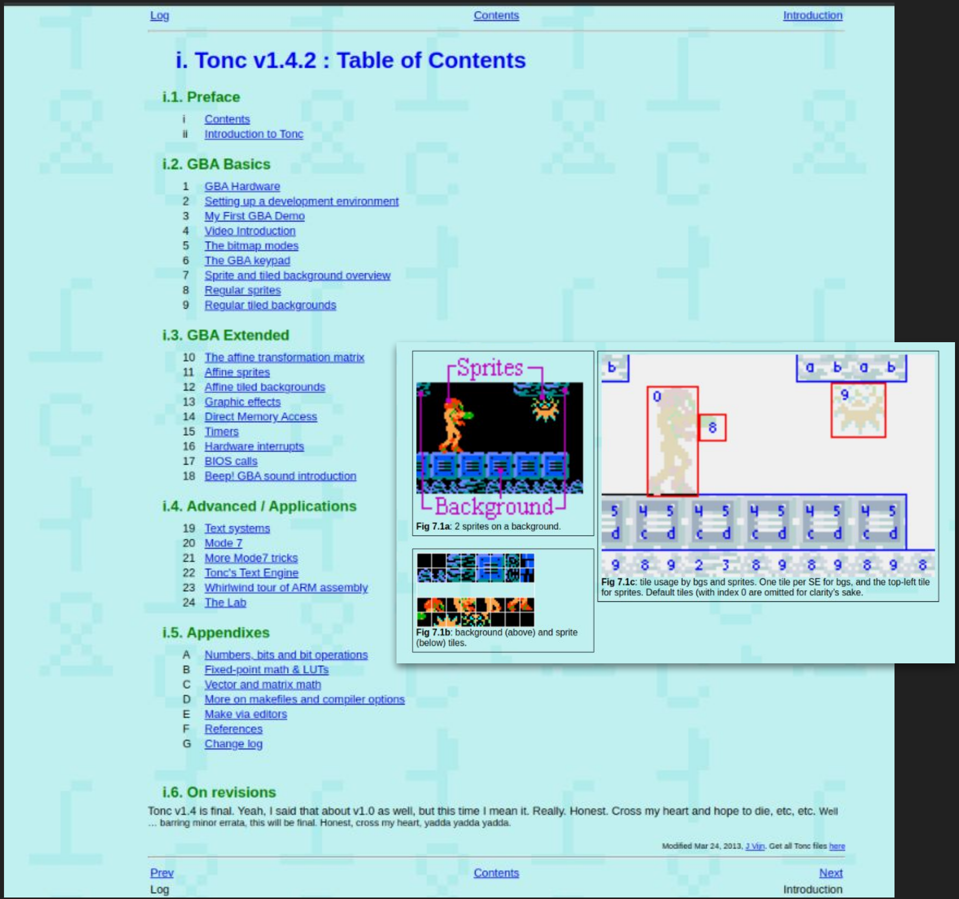
Address Bus Width and CPU Read/Write Access Widths

Shows the Bus-Width, supported read and write widths, and the clock cycles for 8/16/32bit accesses.

Region	Bus	Read	Write	Cycles
BIOS ROM	32	8/16/32	-	1/1/1
Work RAM 32K	32	8/16/32	8/16/32	1/1/1
I/O	32	8/16/32	8/16/32	1/1/1
OAM	32	8/16/32	16/32	1/1/1 *
Work RAM 256K	16	8/16/32	8/16/32	3/3/6 **
Palette RAM	16	8/16/32	16/32	1/1/2 *
VRAM	16	8/16/32	16/32	1/1/2 *
GamePak ROM	16	8/16/32	-	5/5/8 **/***
GamePak Flash	16	8/16/32	16/32	5/5/8 **/***
GamePak SRAM	8	8	8	5 **

Timing Notes:

```
  *   Plus 1 cycle if GBA accesses video memory at the same time.
```



Log | Contents | Introduction

i. Tonc v1.4.2 : Table of Contents

i.1. Preface

i.2. GBA Basics

i.3. GBA Extended

i.4. Advanced / Applications

i.5. Appendixes

Fig 7.1a: 2 sprites on a background.

Fig 7.1b: background (above) and sprite (below) tiles.

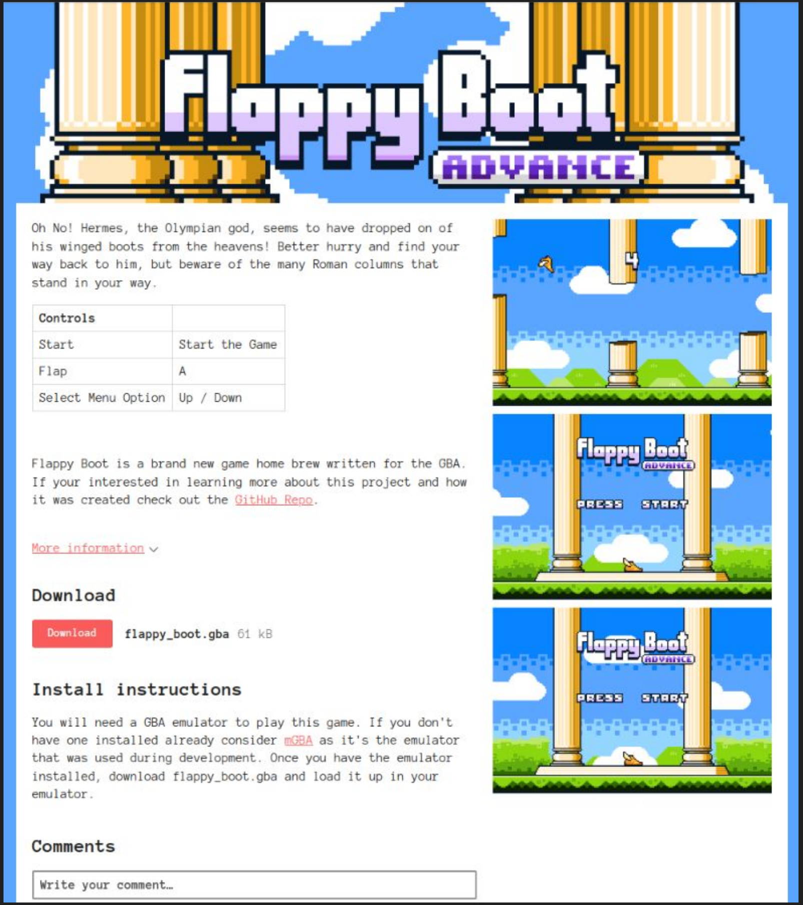
Fig 7.1c: tile usage by bgs and sprites. One tile per SE for bgs, and the top-left tile for sprites. Default tiles (with index 0 are omitted for clarity's sake.

i.6. On revisions

Tonc v1.4 is final. Yeah, I said that about v1.0 as well, but this time I mean it. Really. Honest. Cross my heart and hope to die, etc, etc. Well … barring minor errata, this will be final. Honest, cross my heart, yadda yadda yadda.

Modified Mar 24, 2013, J Vijn. Get all Tonc files here

Prev | Contents | Next
Log | | Introduction



Oh No! Hermes, the Olympian god, seems to have dropped on of his winged boots from the heavens! Better hurry and find your way back to him, but beware of the many Roman columns that stand in your way.

Controls	
Start	Start the Game
Flap	A
Select Menu Option	Up / Down

Flappy Boot is a brand new game home brew written for the GBA. If your interested in learning more about this project and how it was created check out the GitHub Repo.

More information ⌄

Download

Download flappy_boot.gba 61 kB

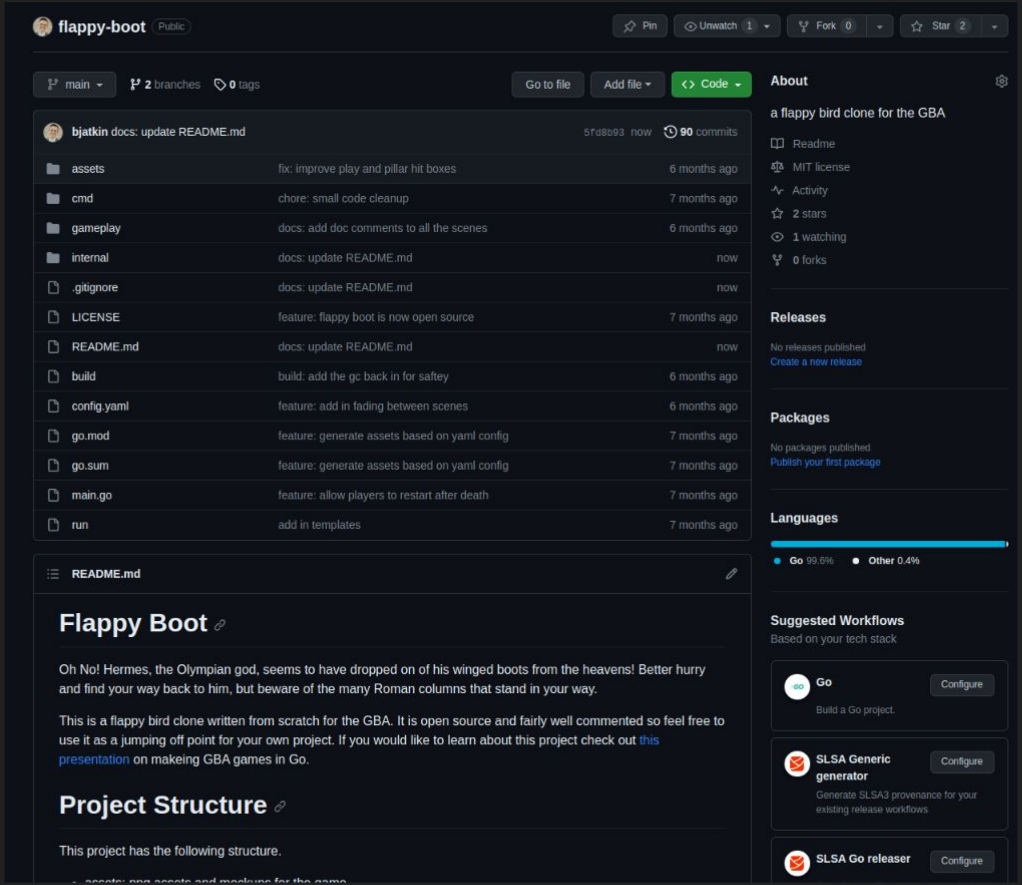
Install instructions

You will need a GBA emulator to play this game. If you don't have one installed already consider mGBA as it's the emulator that was used during development. Once you have the emulator installed, download flappy_boot.gba and load it up in your emulator.

Comments

Write your comment…

https://aanval.itch.io/flappy-boot-advance



flappy-boot Public

2 branches 0 tags

bjatkin docs: update README.md 5fd8b93 now 90 commits

assets	fix: improve play and pillar hit boxes	6 months ago
cmd	chore: small code cleanup	7 months ago
gameplay	docs: add doc comments to all the scenes	6 months ago
internal	docs: update README.md	now
.gitignore	docs: update README.md	now
LICENSE	feature: flappy boot is now open source	7 months ago
README.md	docs: update README.md	now
build	build: add the gc back in for saftey	6 months ago
config.yaml	feature: add in fading between scenes	6 months ago
go.mod	feature: generate assets based on yaml config	7 months ago
go.sum	feature: generate assets based on yaml config	7 months ago
main.go	feature: allow players to restart after death	7 months ago
run	add in templates	7 months ago

README.md

Flappy Boot

Oh No! Hermes, the Olympian god, seems to have dropped on of his winged boots from the heavens! Better hurry and find your way back to him, but beware of the many Roman columns that stand in your way.

This is a flappy bird clone written from scratch for the GBA. It is open source and fairly well commented so feel free to use it as a jumping off point for your own project. If you would like to learn about this project check out this presentation on makeing GBA games in Go.

Project Structure

This project has the following structure.

- assets: png assets and mockups for the game

About

a flappy bird clone for the GBA

Readme
MIT license
Activity
2 stars
1 watching
0 forks

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

Languages

● Go 99.6% ● Other 0.4%

Suggested Workflows

Based on your tech stack

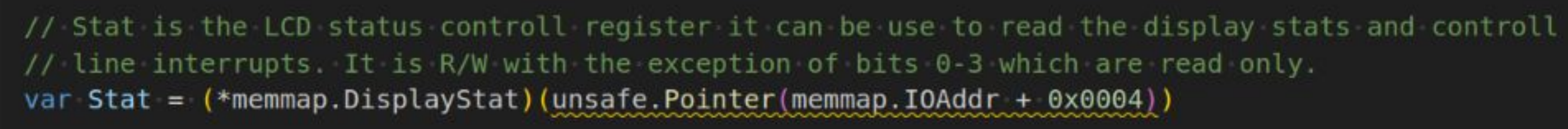
Go — Configure
Build a Go project.

SLSA Generic generator — Configure
Generate SLSA3 provenance for your existing release workflows

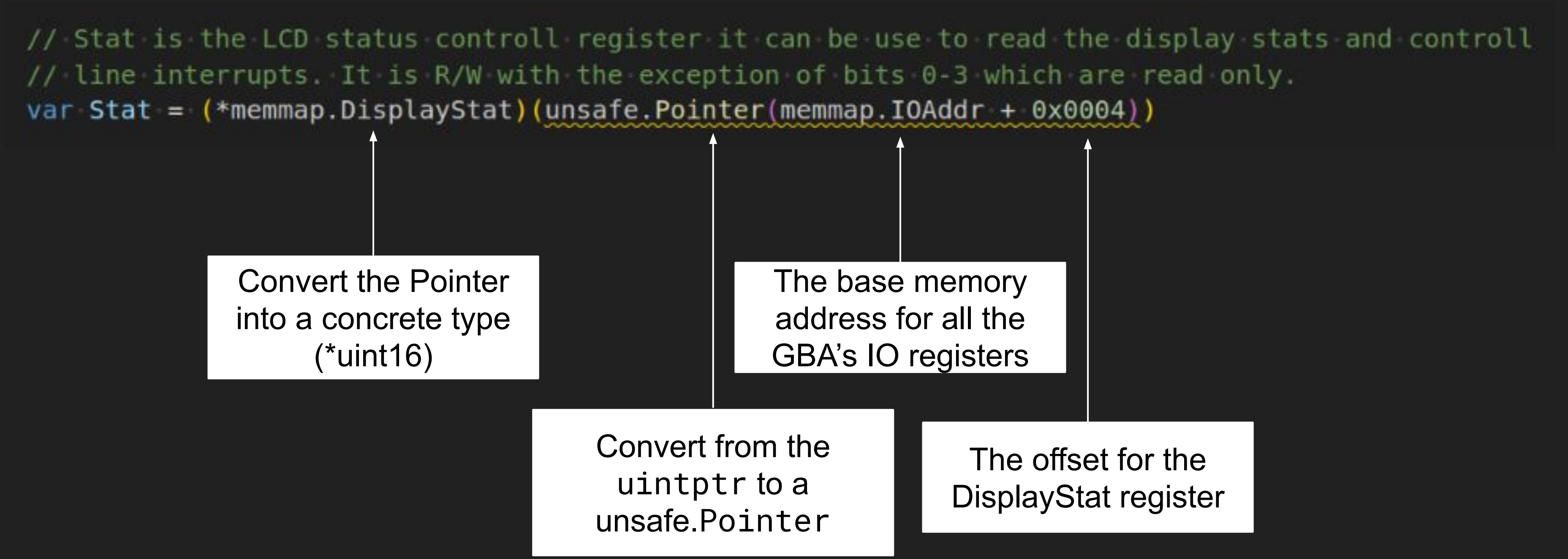
SLSA Go releaser — Configure

https://github.com/bjatkin/flappy-boot

CODE



```go
// Stat is the LCD status controll register it can be use to read the display stats and controll
// line interrupts. It is R/W with the exception of bits 0-3 which are read only.
var Stat = (*memmap.DisplayStat)(unsafe.Pointer(memmap.IOAddr + 0x0004))
```



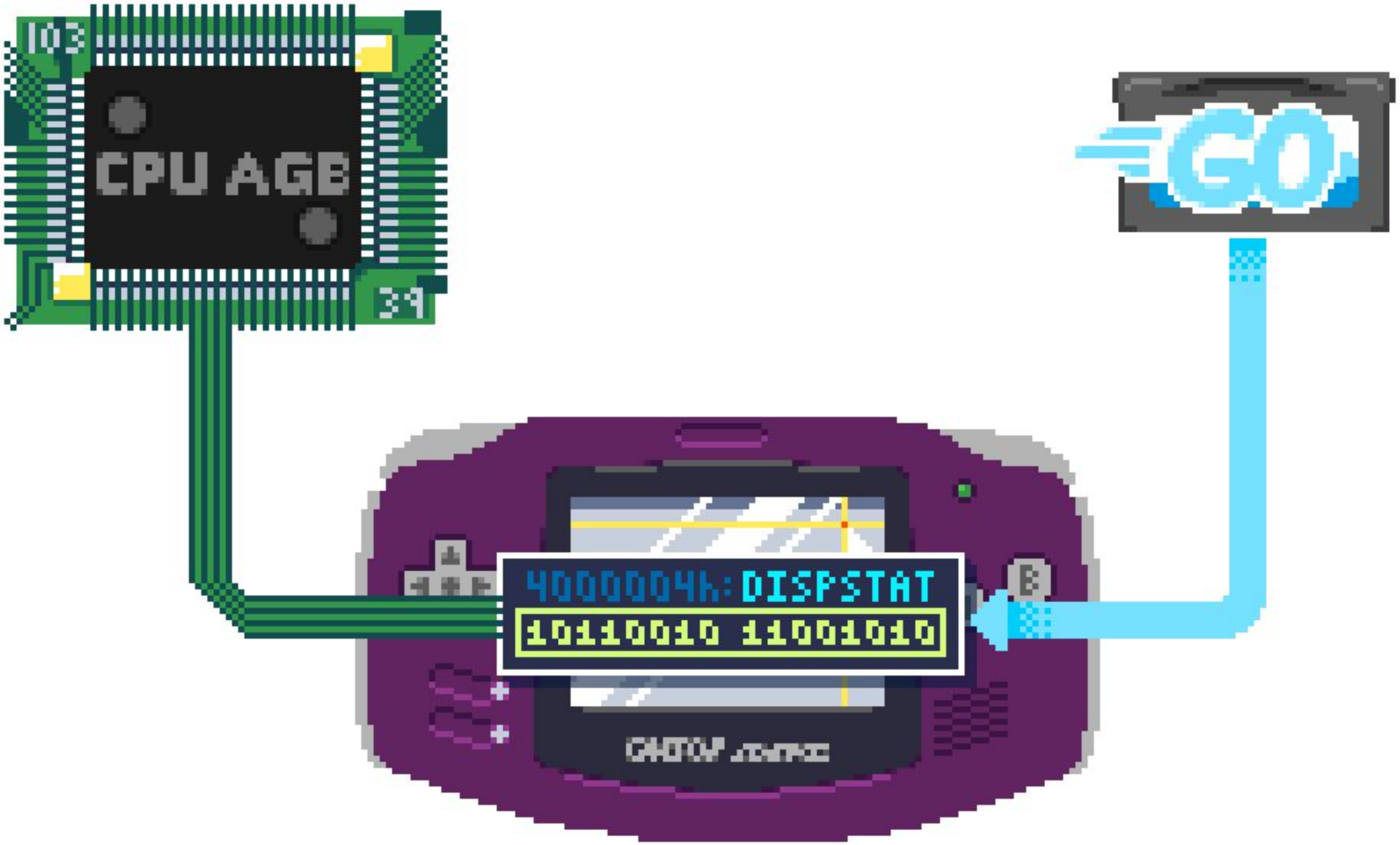
```go
// Stat is the LCD status controll register it can be use to read the display stats and controll
// line interrupts. It is R/W with the exception of bits 0-3 which are read only.
var Stat = (*memmap.DisplayStat)(unsafe.Pointer(memmap.IOAddr + 0x0004))
```

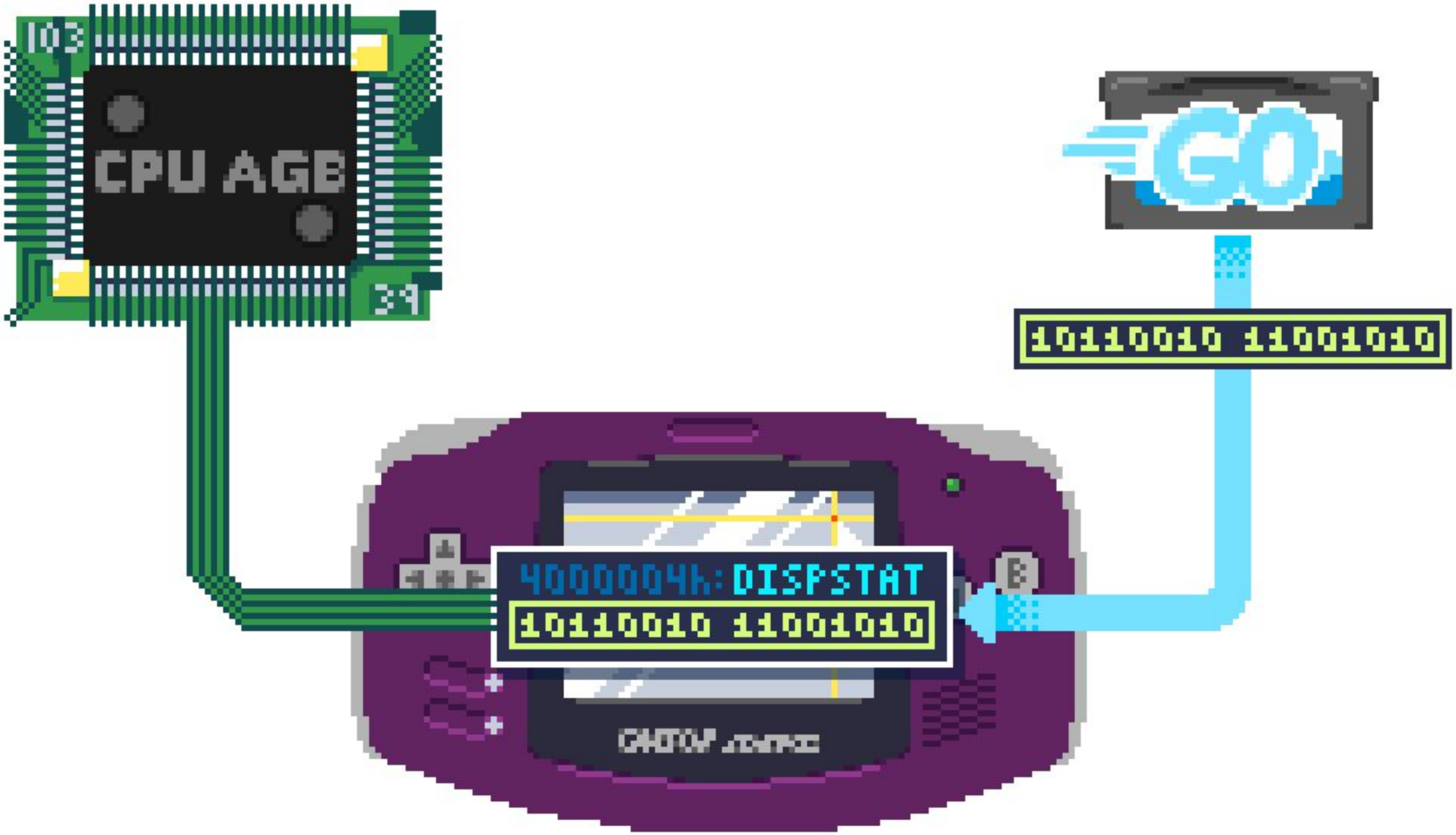
Convert the Pointer into a concrete type (*uint16)

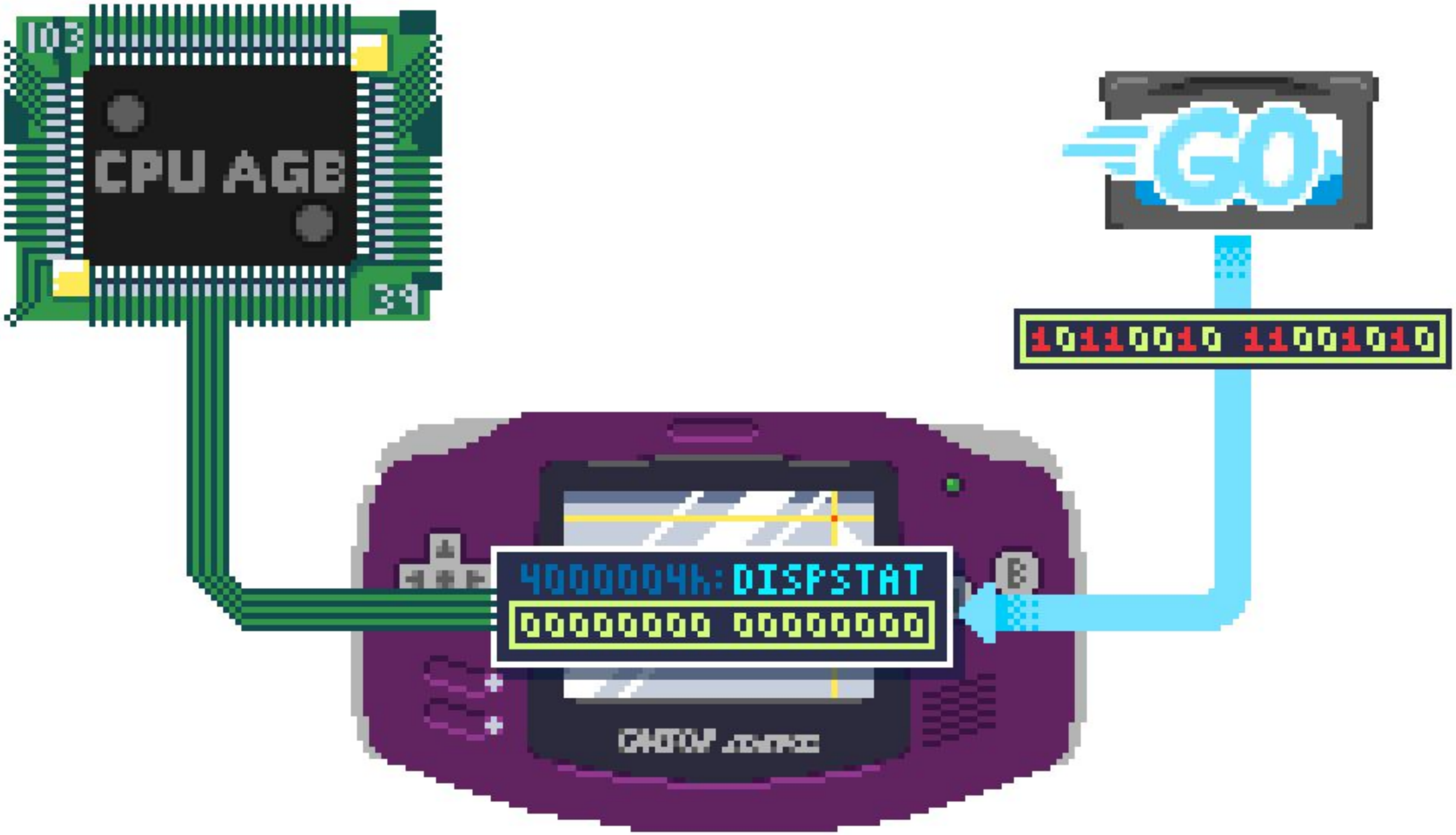
Convert from the `uintptr` to a unsafe.`Pointer`

The base memory address for all the GBA's IO registers

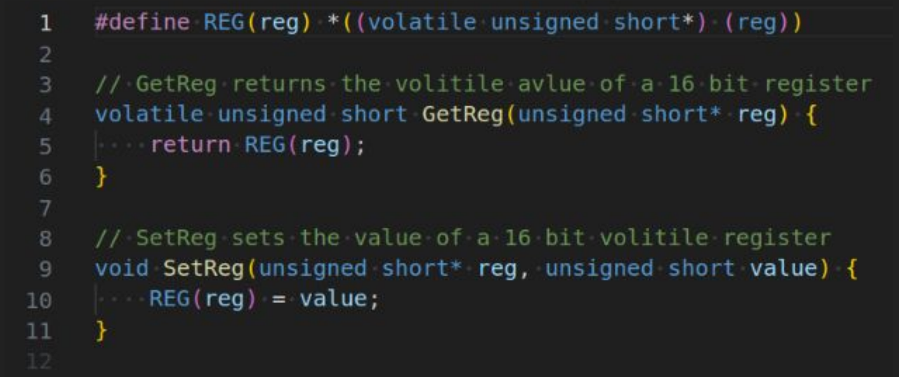
The offset for the DisplayStat register


103
CPU AGB
39
GO
4000004h:DISPSTAT
10110010 11001010
GAME BOY ADVANCE


103
CPU AGB
39
GO
10110010 11001010
4000004h:DISPSTAT
10110010 11001010
B

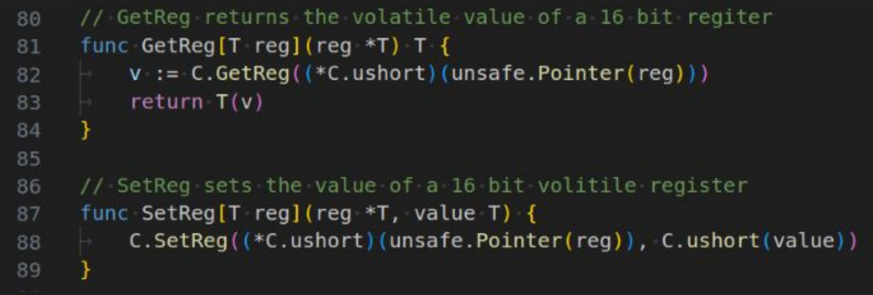

103
CPU AGB
39
GO
10110010 11001010
4000004h:DISPSTAT
00000000 00000000
B

memmap.h

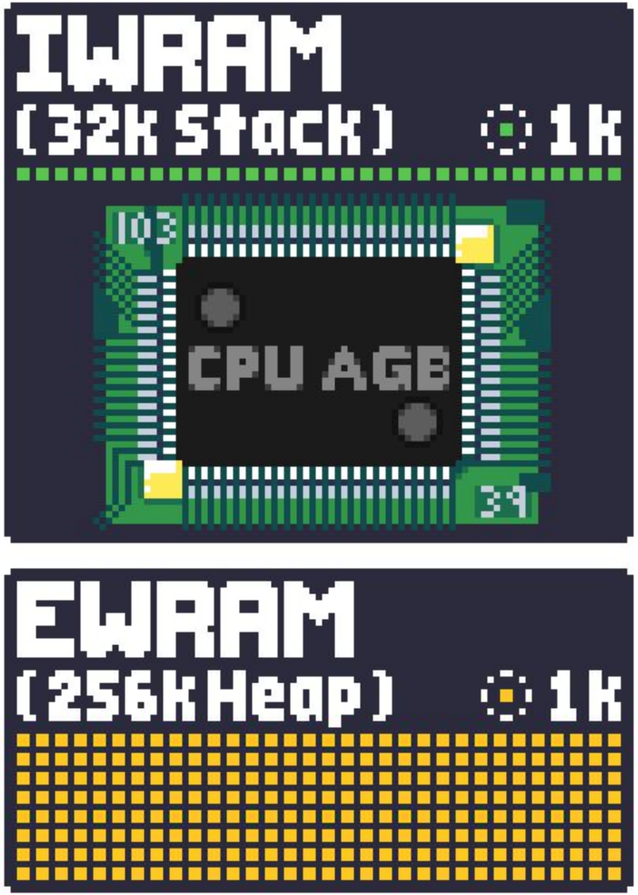


```c
#define REG(reg) *((volatile unsigned short*) (reg))

// GetReg returns the volitile avlue of a 16 bit register
volatile unsigned short GetReg(unsigned short* reg) {
    return REG(reg);
}

// SetReg sets the value of a 16 bit volitile register
void SetReg(unsigned short* reg, unsigned short value) {
    REG(reg) = value;
}
```

memmap.go

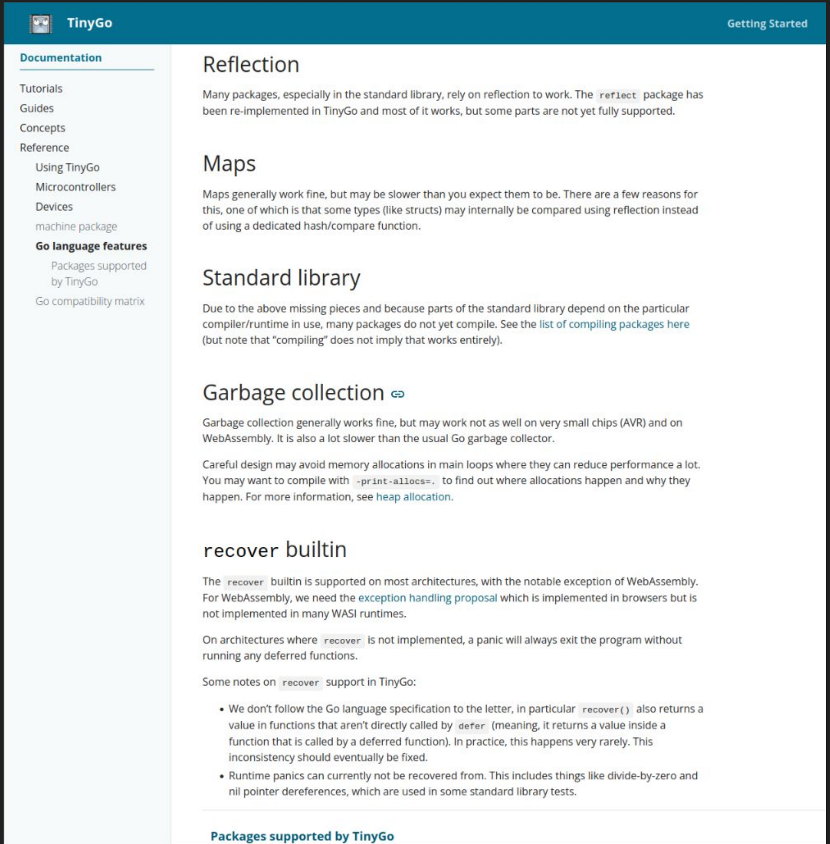


```go
// GetReg returns the volatile value of a 16 bit regiter
func GetReg[T reg](reg *T) T {
    v := C.GetReg((*C.ushort)(unsafe.Pointer(reg)))
    return T(v)
}

// SetReg sets the value of a 16 bit volitile register
func SetReg[T reg](reg *T, value T) {
    C.SetReg((*C.ushort)(unsafe.Pointer(reg)), C.ushort(value))
}
```


IWRAM
(32k Stack)
1k
103
CPU AGB
39
EWRAM
(256k Heap)
1k

Tiny Go Docs

https://tinygo.org/docs/reference/lang-support/



Reflection

Many packages, especially in the standard library, rely on reflection to work. The `reflect` package has been re-implemented in TinyGo and most of it works, but some parts are not yet fully supported.

Maps

Maps generally work fine, but may be slower than you expect them to be. There are a few reasons for this, one of which is that some types (like structs) may internally be compared using reflection instead of using a dedicated hash/compare function.

Standard library

Due to the above missing pieces and because parts of the standard library depend on the particular compiler/runtime in use, many packages do not yet compile. See the list of compiling packages here (but note that “compiling” does not imply that works entirely).

Garbage collection

Garbage collection generally works fine, but may work not as well on very small chips (AVR) and on WebAssembly. It is also a lot slower than the usual Go garbage collector.

Careful design may avoid memory allocations in main loops where they can reduce performance a lot. You may want to compile with `-print-allocs=.` to find out where allocations happen and why they happen. For more information, see heap allocation.

`recover` builtin

The `recover` builtin is supported on most architectures, with the notable exception of WebAssembly. For WebAssembly, we need the exception handling proposal which is implemented in browsers but is not implemented in many WASI runtimes.

On architectures where `recover` is not implemented, a panic will always exit the program without running any deferred functions.

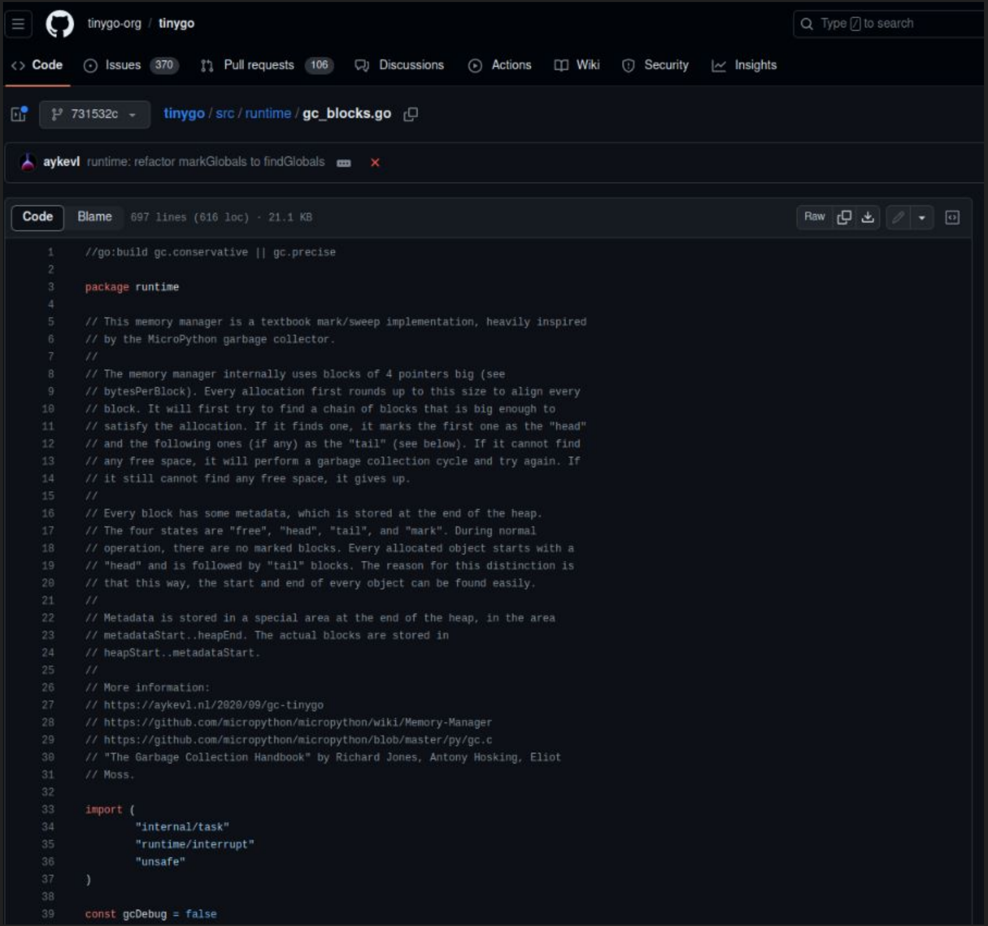
Some notes on `recover` support in TinyGo:

- We don’t follow the Go language specification to the letter, in particular `recover()` also returns a value in functions that aren’t directly called by `defer` (meaning, it returns a value inside a function that is called by a deferred function). In practice, this happens very rarely. This inconsistency should eventually be fixed.
- Runtime panics can currently not be recovered from. This includes things like divide-by-zero and nil pointer dereferences, which are used in some standard library tests.

Packages supported by TinyGo

Tiny Go GitHub

https://github.com/tinygo-org/tinygo/blob/release/src/runtime/gc_blocks.go



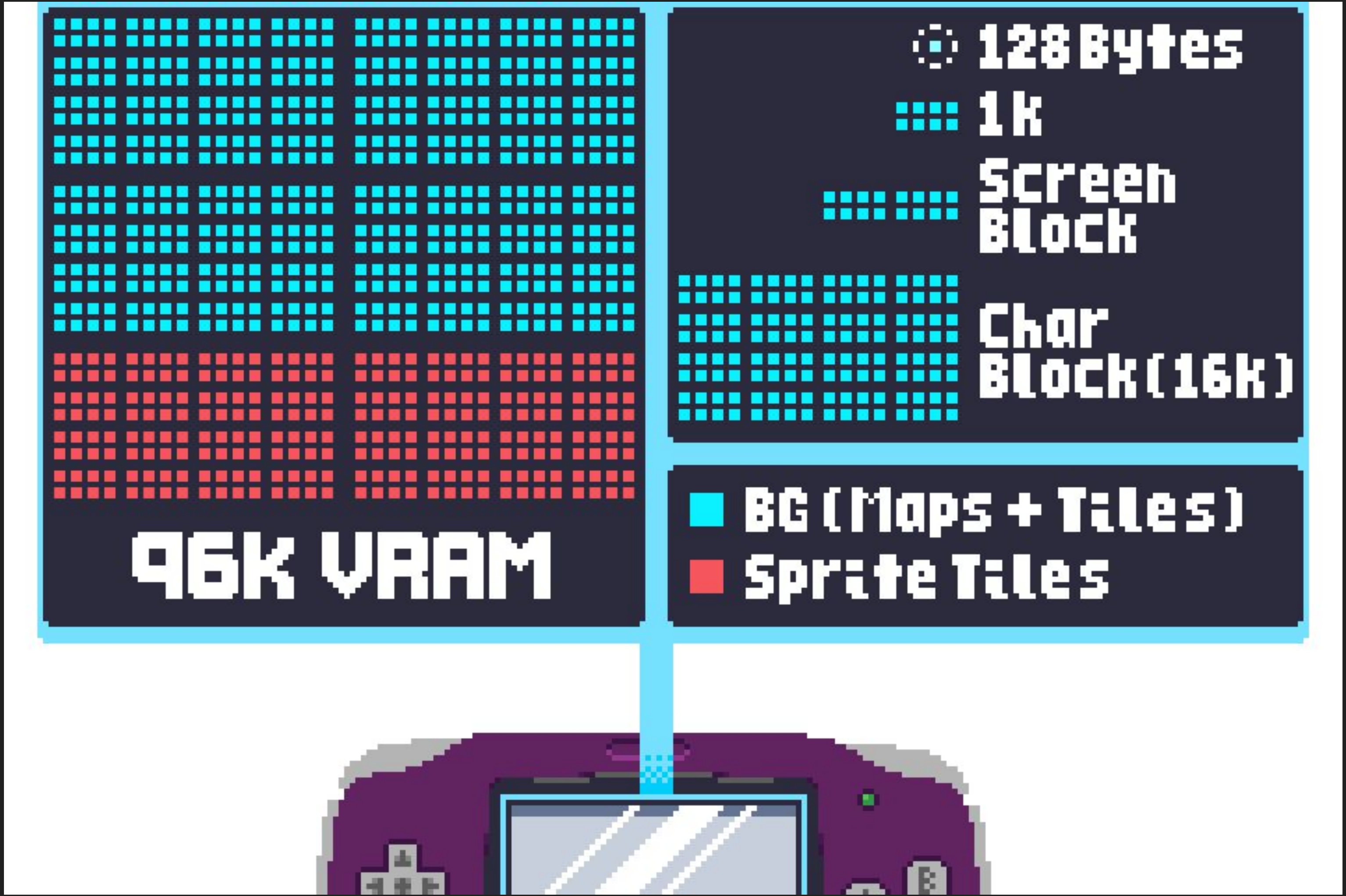
aykevl runtime: refactor markGlobals to findGlobals

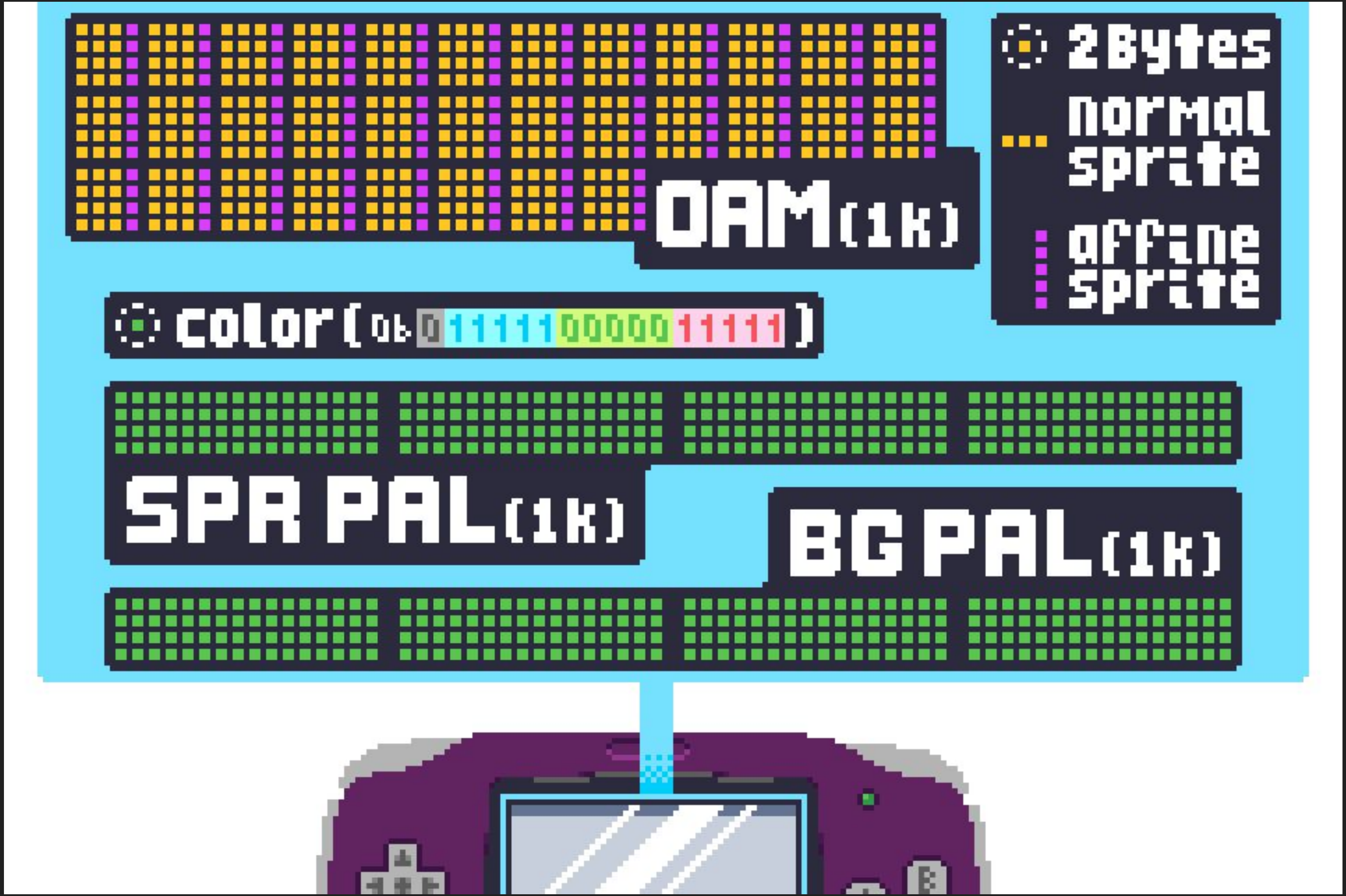
697 lines (616 loc) · 21.1 KB

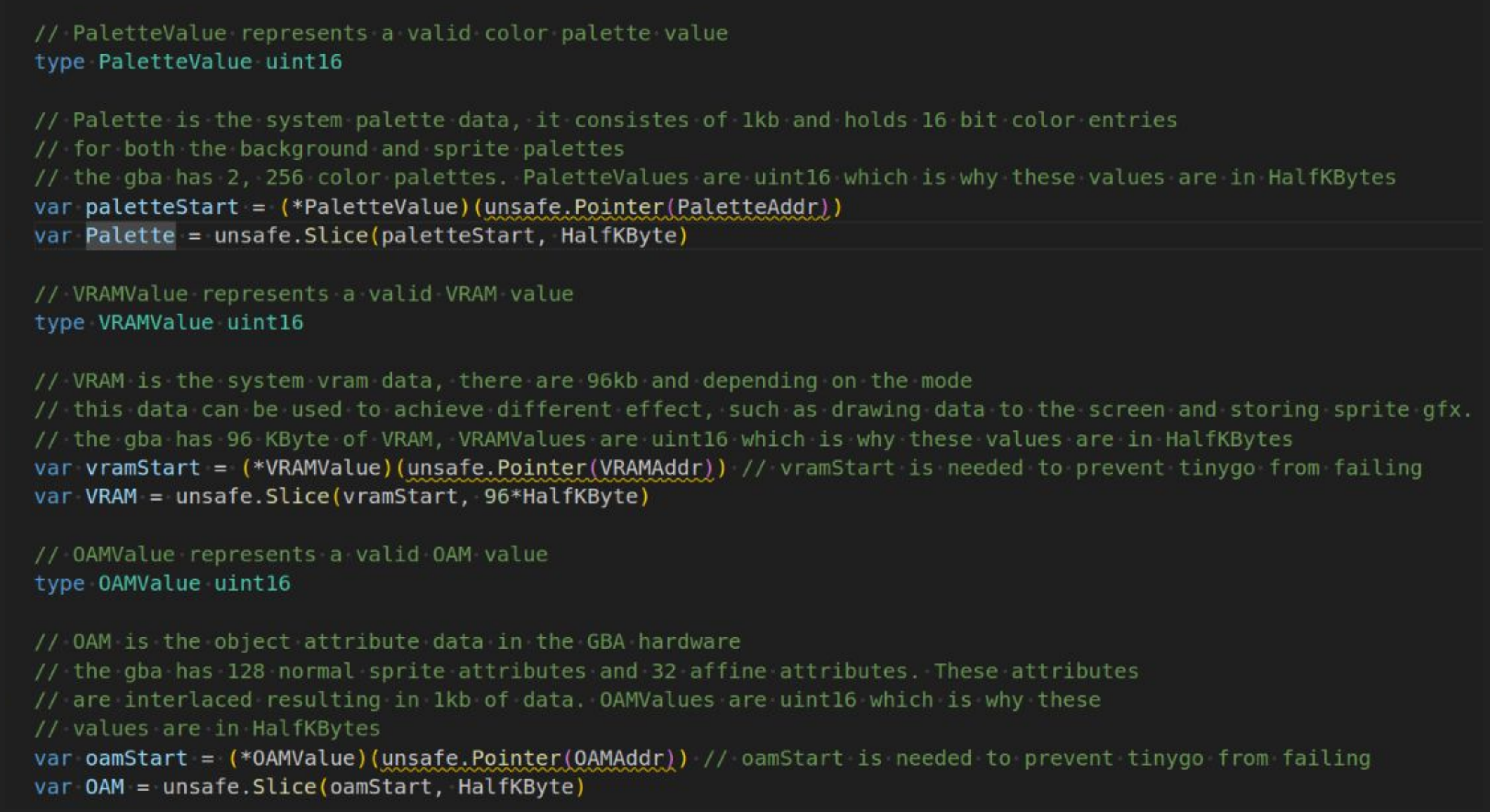
```go
//go:build gc.conservative || gc.precise

package runtime

// This memory manager is a textbook mark/sweep implementation, heavily inspired
// by the MicroPython garbage collector.
//
// The memory manager internally uses blocks of 4 pointers big (see
// bytesPerBlock). Every allocation first rounds up to this size to align every
// block. It will first try to find a chain of blocks that is big enough to
// satisfy the allocation. If it finds one, it marks the first one as the "head"
// and the following ones (if any) as the "tail" (see below). If it cannot find
// any free space, it will perform a garbage collection cycle and try again. If
// it still cannot find any free space, it gives up.
//
// Every block has some metadata, which is stored at the end of the heap.
// The four states are "free", "head", "tail", and "mark". During normal
// operation, there are no marked blocks. Every allocated object starts with a
// "head" and is followed by "tail" blocks. The reason for this distinction is
// that this way, the start and end of every object can be found easily.
//
// Metadata is stored in a special area at the end of the heap, in the area
// metadataStart..heapEnd. The actual blocks are stored in
// heapStart..metadataStart.
//
// More information:
// https://aykevl.nl/2020/09/gc-tinygo
// https://github.com/micropython/micropython/wiki/Memory-Manager
// https://github.com/micropython/micropython/blob/master/py/gc.c
// "The Garbage Collection Handbook" by Richard Jones, Antony Hosking, Eliot
// Moss.

import (
	"internal/task"
	"runtime/interrupt"
	"unsafe"
)

const gcDebug = false
```

GFX

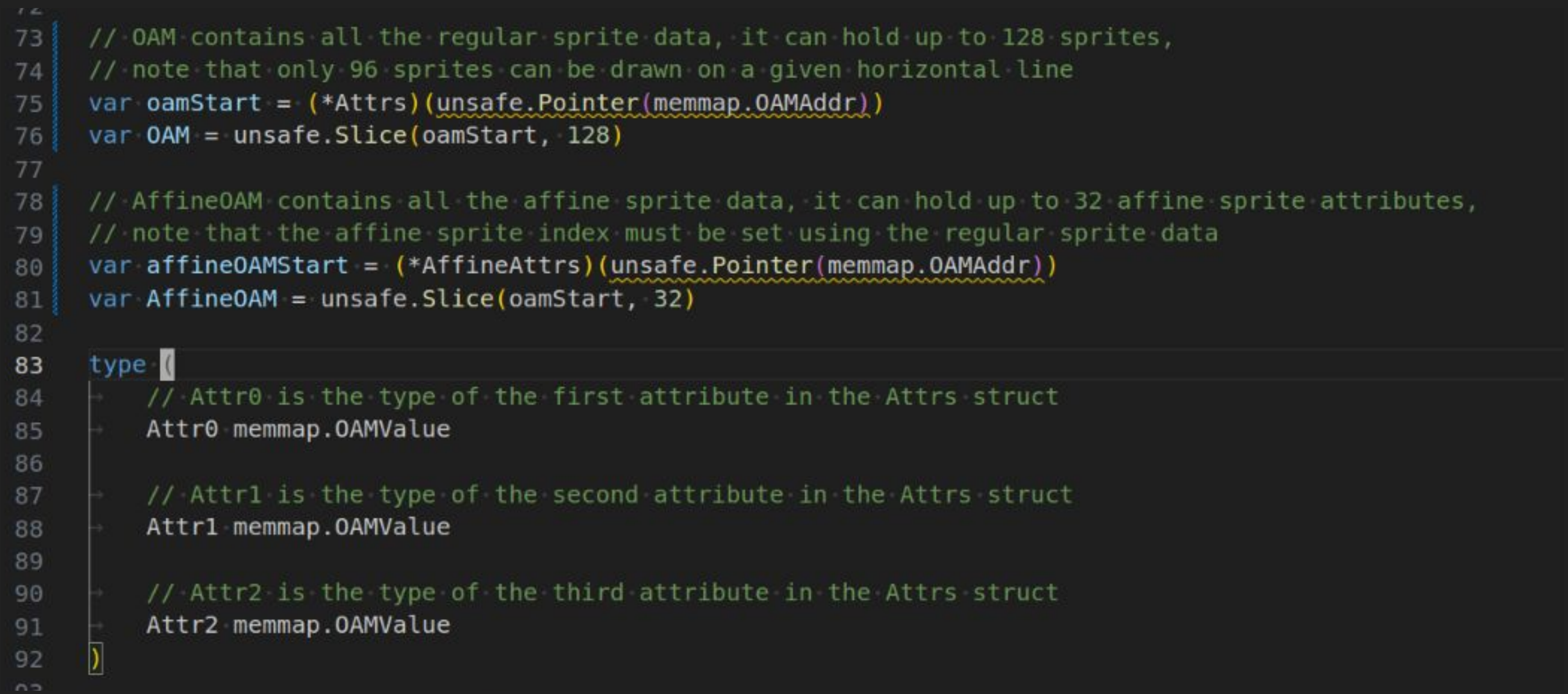
Mode	BG Layer 0	BG Layer 1	BG Layer 2	BG Layer 3	Colors	Sprite VRAM
0	normal	normal	normal	normal	16x16 or 256x1	32k
1	normal	normal	affine	-	16x16 or 256x1	32k
2	-	-	affine	affine	256x1	32k
3	-	-	bitmap[240x160]	-	32,768	16k
4	-	-	bitmap[240x160]x2	-	256/1	16k
5	-	-	bitmap[160x128]x2	-	32,768	16k


96K VRAM
128 Bytes
1k
Screen Block
Char Block (16k)
🟦 BG (Maps + Tiles)
🟥 Sprite Tiles

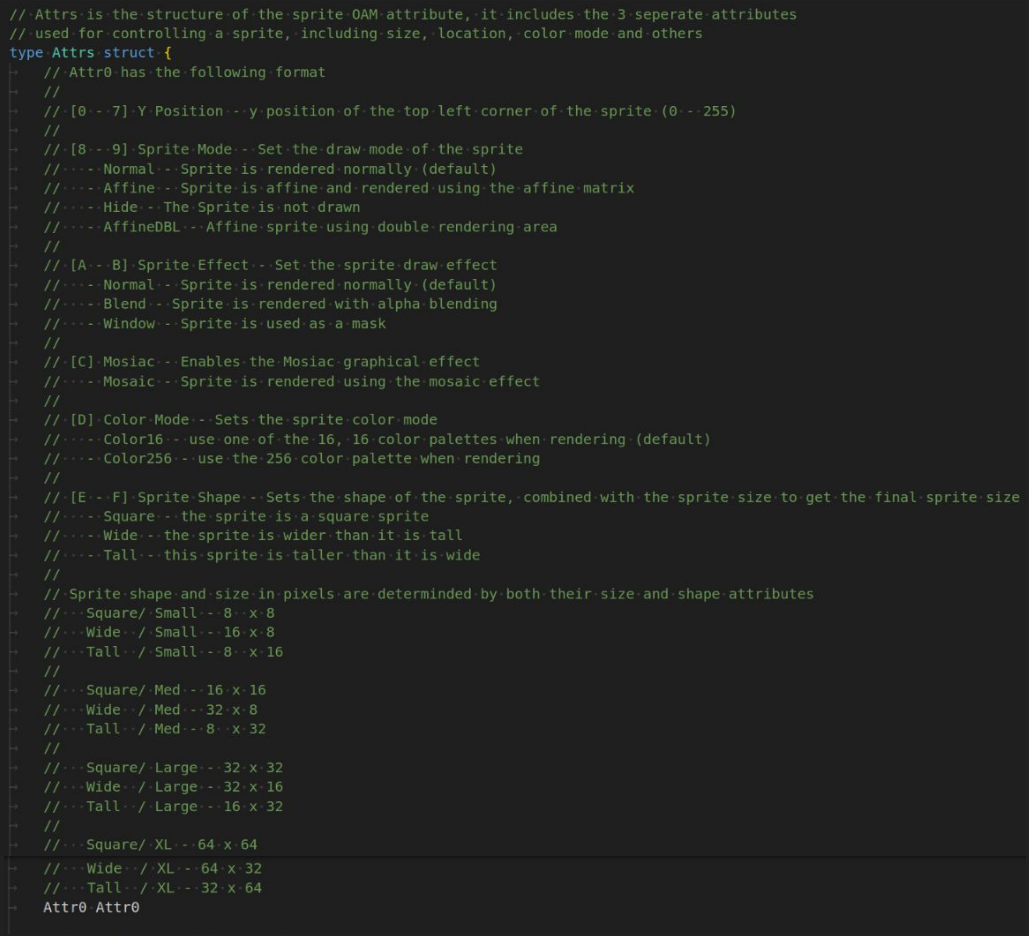

OAM(1k)
2Bytes
normal sprite
affine sprite
color(0b0111110000011111)
SPR PAL(1k)
BG PAL(1k)



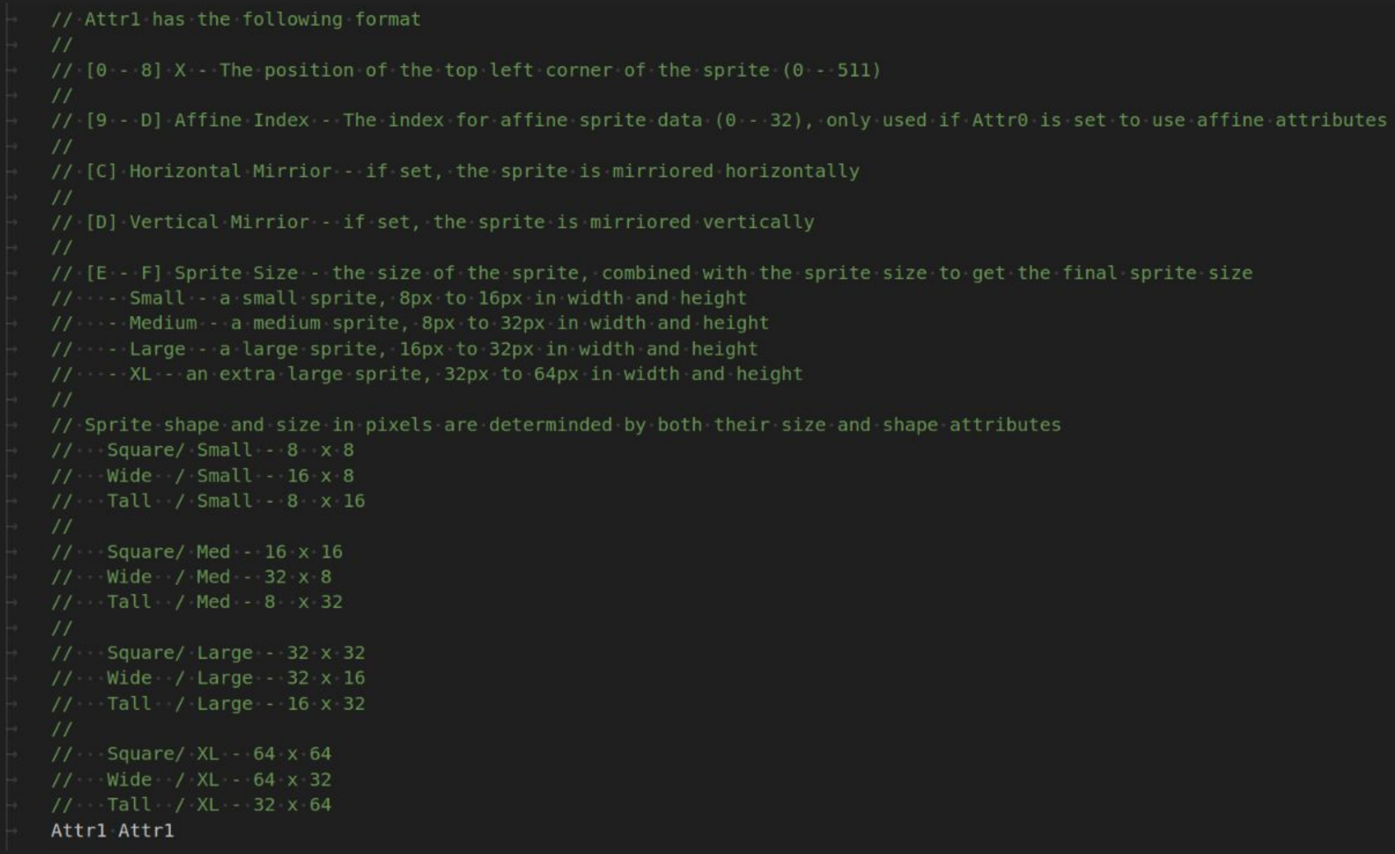
```go
// PaletteValue represents a valid color palette value
type PaletteValue uint16

// Palette is the system palette data, it consistes of 1kb and holds 16 bit color entries
// for both the background and sprite palettes
// the gba has 2, 256 color palettes. PaletteValues are uint16 which is why these values are in HalfKBytes
var paletteStart = (*PaletteValue)(unsafe.Pointer(PaletteAddr))
var Palette = unsafe.Slice(paletteStart, HalfKByte)

// VRAMValue represents a valid VRAM value
type VRAMValue uint16

// VRAM is the system vram data, there are 96kb and depending on the mode
// this data can be used to achieve different effect, such as drawing data to the screen and storing sprite gfx.
// the gba has 96 KByte of VRAM, VRAMValues are uint16 which is why these values are in HalfKBytes
var vramStart = (*VRAMValue)(unsafe.Pointer(VRAMAddr)) // vramStart is needed to prevent tinygo from failing
var VRAM = unsafe.Slice(vramStart, 96*HalfKByte)

// OAMValue represents a valid OAM value
type OAMValue uint16

// OAM is the object attribute data in the GBA hardware
// the gba has 128 normal sprite attributes and 32 affine attributes. These attributes
// are interlaced resulting in 1kb of data. OAMValues are uint16 which is why these
// values are in HalfKBytes
var oamStart = (*OAMValue)(unsafe.Pointer(OAMAddr)) // oamStart is needed to prevent tinygo from failing
var OAM = unsafe.Slice(oamStart, HalfKByte)
```



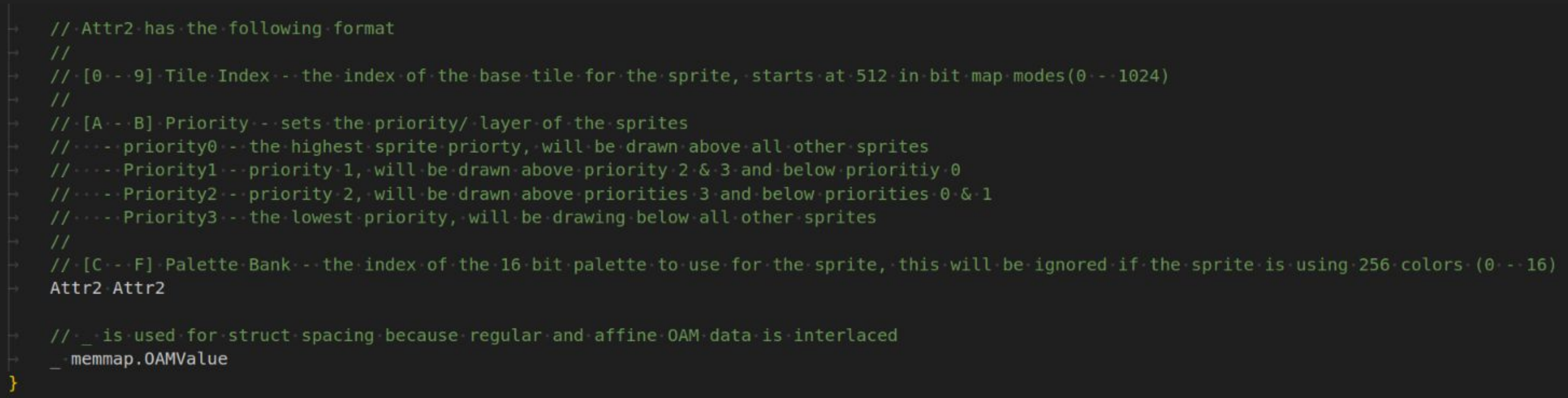
```go
// OAM contains all the regular sprite data, it can hold up to 128 sprites,
// note that only 96 sprites can be drawn on a given horizontal line
var oamStart = (*Attrs)(unsafe.Pointer(memmap.OAMAddr))
var OAM = unsafe.Slice(oamStart, 128)

// AffineOAM contains all the affine sprite data, it can hold up to 32 affine sprite attributes,
// note that the affine sprite index must be set using the regular sprite data
var affineOAMStart = (*AffineAttrs)(unsafe.Pointer(memmap.OAMAddr))
var AffineOAM = unsafe.Slice(oamStart, 32)

type (
	// Attr0 is the type of the first attribute in the Attrs struct
	Attr0 memmap.OAMValue

	// Attr1 is the type of the second attribute in the Attrs struct
	Attr1 memmap.OAMValue

	// Attr2 is the type of the third attribute in the Attrs struct
	Attr2 memmap.OAMValue
)
```



```go
// Attrs is the structure of the sprite OAM attribute, it includes the 3 seperate attributes
// used for controlling a sprite, including size, location, color mode and others
type Attrs struct {
    // Attr0 has the following format
    //
    // [0 - 7] Y Position - y position of the top left corner of the sprite (0 - 255)
    //
    // [8 - 9] Sprite Mode - Set the draw mode of the sprite
    //   - Normal - Sprite is rendered normally (default)
    //   - Affine - Sprite is affine and rendered using the affine matrix
    //   - Hide - The Sprite is not drawn
    //   - AffineDBL - Affine sprite using double rendering area
    //
    // [A - B] Sprite Effect - Set the sprite draw effect
    //   - Normal - Sprite is rendered normally (default)
    //   - Blend - Sprite is rendered with alpha blending
    //   - Window - Sprite is used as a mask
    //
    // [C] Mosiac - Enables the Mosiac graphical effect
    //   - Mosaic - Sprite is rendered using the mosaic effect
    //
    // [D] Color Mode - Sets the sprite color mode
    //   - Color16 - use one of the 16, 16 color palettes when rendering (default)
    //   - Color256 - use the 256 color palette when rendering
    //
    // [E - F] Sprite Shape - Sets the shape of the sprite, combined with the sprite size to get the final sprite size
    //   - Square - the sprite is a square sprite
    //   - Wide - the sprite is wider than it is tall
    //   - Tall - this sprite is taller than it is wide
    //
    // Sprite shape and size in pixels are determinded by both their size and shape attributes
    //   Square/ Small - 8  x 8
    //   Wide  / Small - 16 x 8
    //   Tall  / Small - 8  x 16
    //
    //   Square/ Med - 16 x 16
    //   Wide  / Med - 32 x 8
    //   Tall  / Med - 8  x 32
    //
    //   Square/ Large - 32 x 32
    //   Wide  / Large - 32 x 16
    //   Tall  / Large - 16 x 32
    //
    //   Square/ XL - 64 x 64
    //   Wide  / XL - 64 x 32
    //   Tall  / XL - 32 x 64
    Attr0 Attr0
```

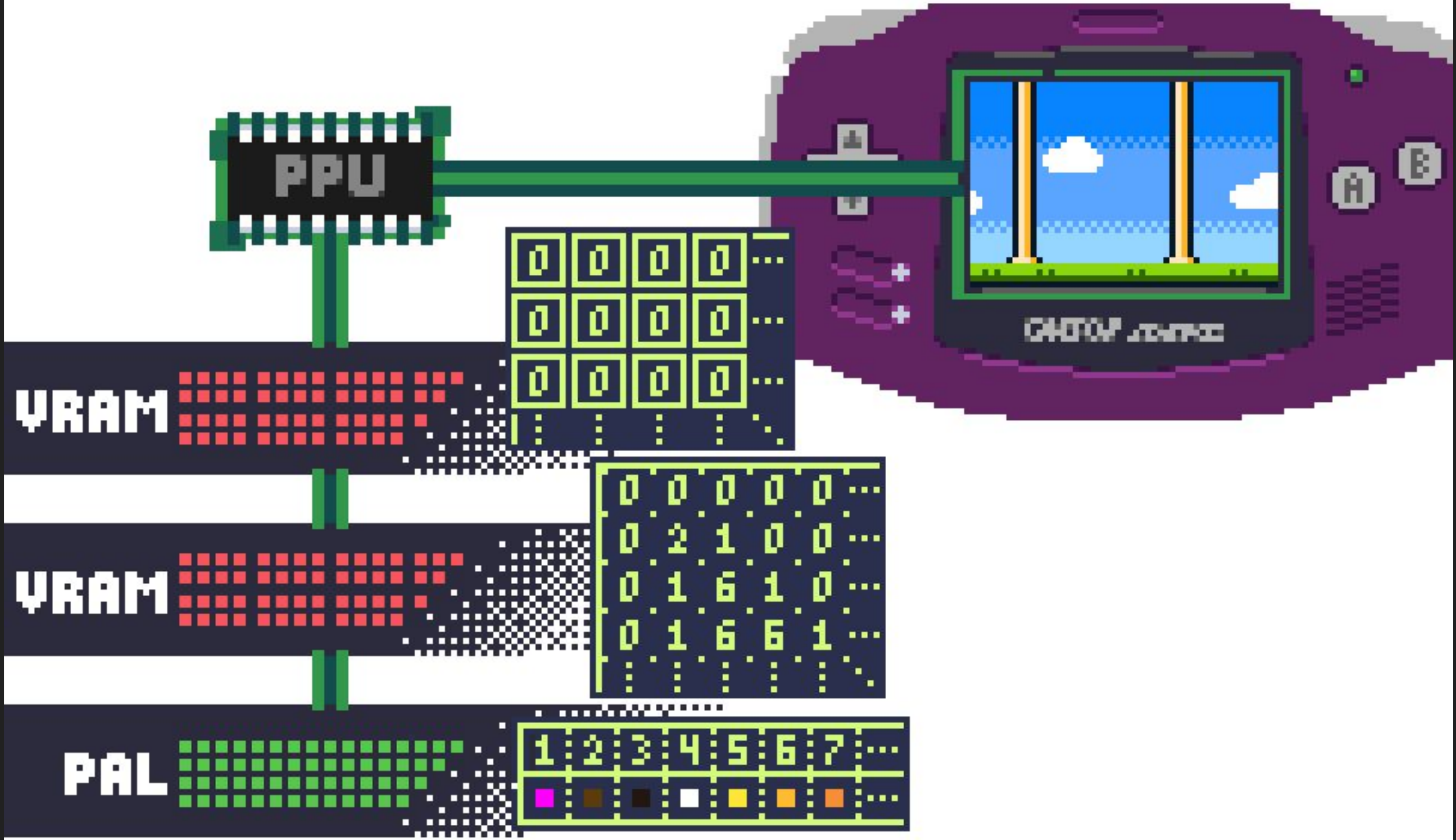


```
    // Attr1 has the following format
    //
    // [0 - 8] X - The position of the top left corner of the sprite (0 - 511)
    //
    // [9 - D] Affine Index - The index for affine sprite data (0 - 32), only used if Attr0 is set to use affine attributes
    //
    // [C] Horizontal Mirrior - if set, the sprite is mirriored horizontally
    //
    // [D] Vertical Mirrior - if set, the sprite is mirriored vertically
    //
    // [E - F] Sprite Size - the size of the sprite, combined with the sprite size to get the final sprite size
    //   - Small - a small sprite, 8px to 16px in width and height
    //   - Medium - a medium sprite, 8px to 32px in width and height
    //   - Large - a large sprite, 16px to 32px in width and height
    //   - XL - an extra large sprite, 32px to 64px in width and height
    //
    // Sprite shape and size in pixels are determinded by both their size and shape attributes
    //   Square/ Small - 8  x 8
    //   Wide  / Small - 16 x 8
    //   Tall  / Small - 8  x 16
    //
    //   Square/ Med - 16 x 16
    //   Wide  / Med - 32 x 8
    //   Tall  / Med - 8  x 32
    //
    //   Square/ Large - 32 x 32
    //   Wide  / Large - 32 x 16
    //   Tall  / Large - 16 x 32
    //
    //   Square/ XL - 64 x 64
    //   Wide  / XL - 64 x 32
    //   Tall  / XL - 32 x 64
    Attr1 Attr1
```



```
	// Attr2 has the following format
	//
	// [0 - 9] Tile Index - the index of the base tile for the sprite, starts at 512 in bit map modes(0 - 1024)
	//
	// [A - B] Priority - sets the priority/ layer of the sprites
	//   - priority0 - the highest sprite priorty, will be drawn above all other sprites
	//   - Priority1 - priority 1, will be drawn above priority 2 & 3 and below prioritiy 0
	//   - Priority2 - priority 2, will be drawn above priorities 3 and below priorities 0 & 1
	//   - Priority3 - the lowest priority, will be drawing below all other sprites
	//
	// [C - F] Palette Bank - the index of the 16 bit palette to use for the sprite, this will be ignored if the sprite is using 256 colors (0 - 16)
	Attr2 Attr2

	// _ is used for struct spacing because regular and affine OAM data is interlaced
	_ memmap.OAMValue
}
```

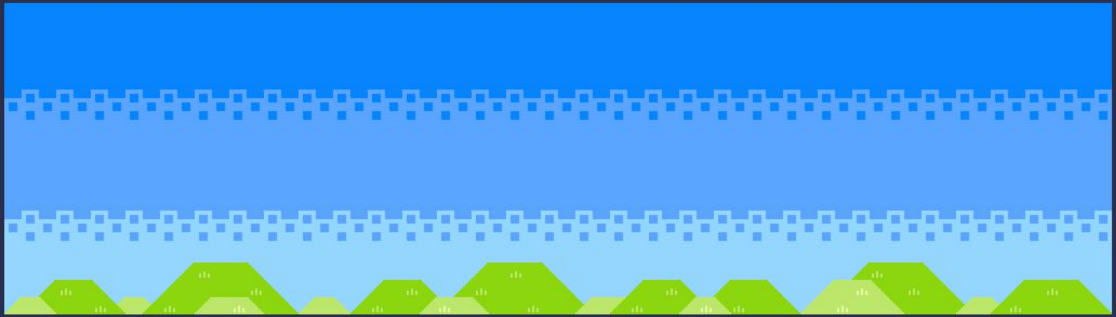
DRAWING BACKGROUNDS



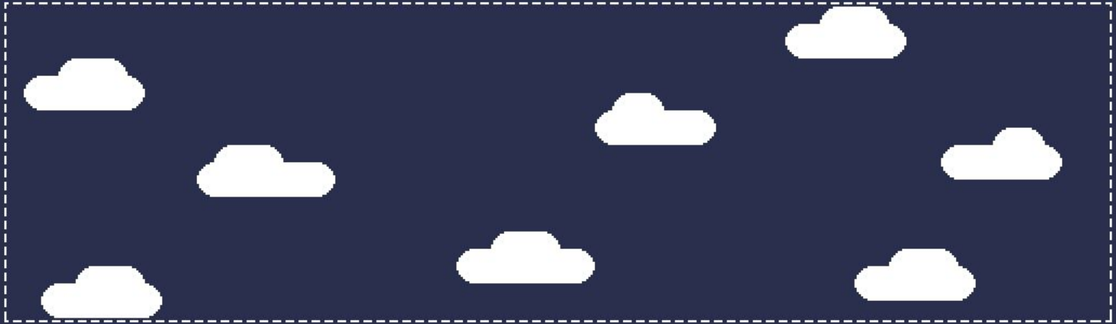
```
0 0 0 0 ...
0 0 0 0 ...
0 0 0 0 ...
```

```
0 0 0 0 0 ...
0 2 1 0 0 ...
0 1 6 1 0 ...
0 1 6 6 1 ...
```

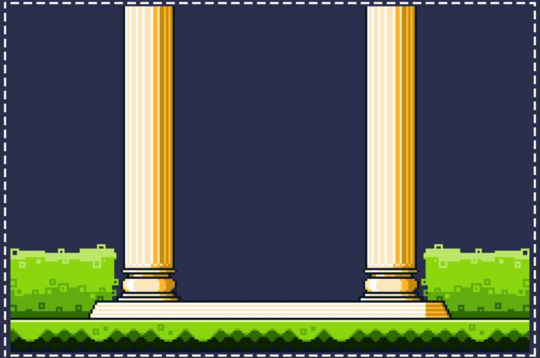
1	2	3	4	5	6	7	...
🟪	🟫	⬛	⬜	🟨	🟧	🟧	...



BG 0
512×256

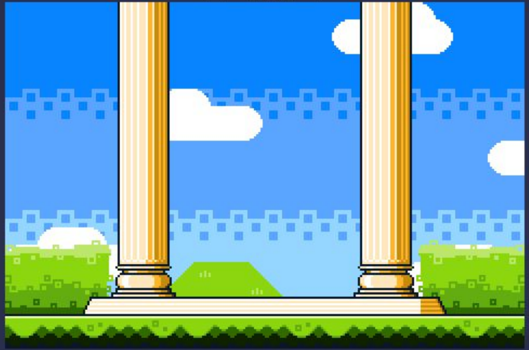


BG 1
512×256

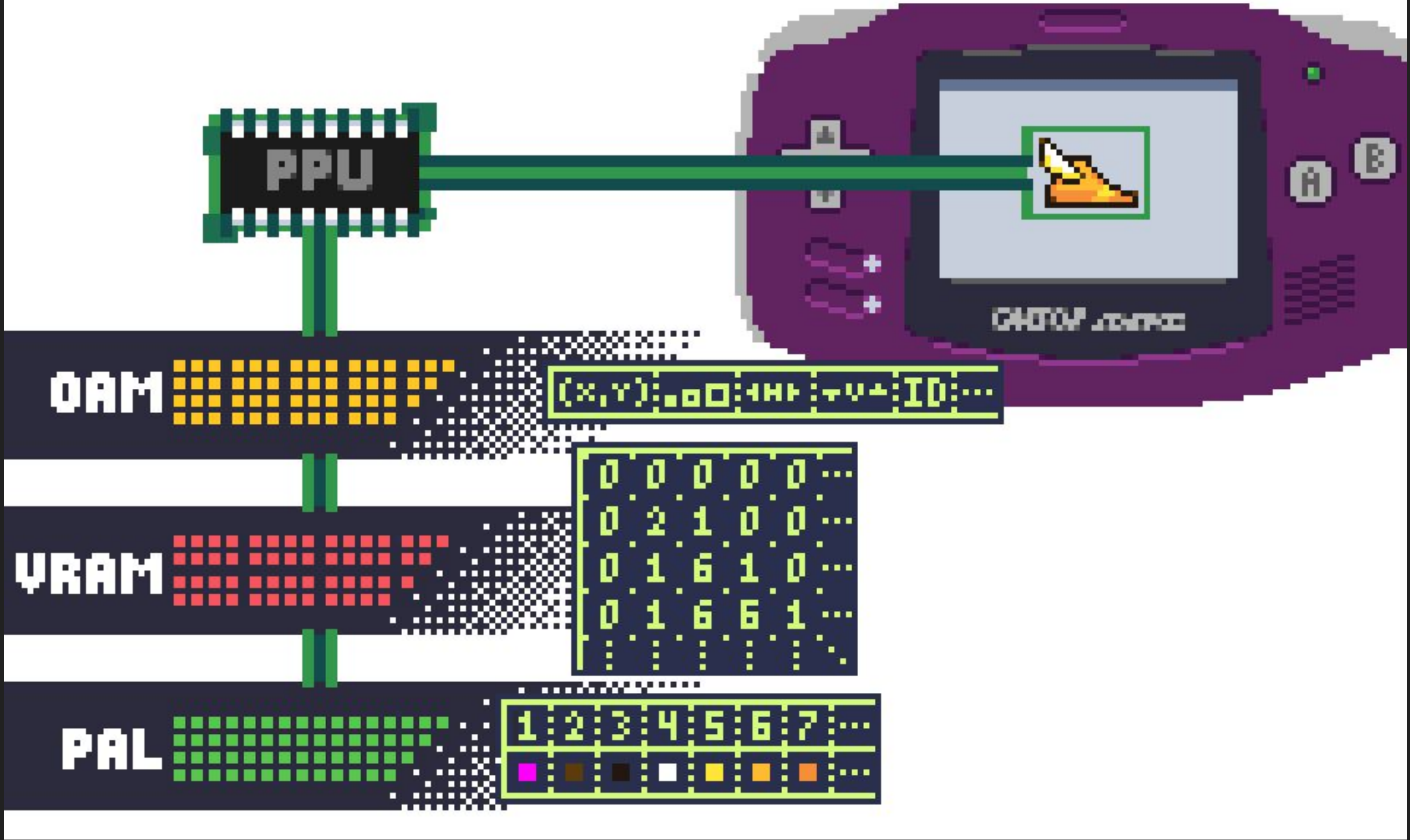


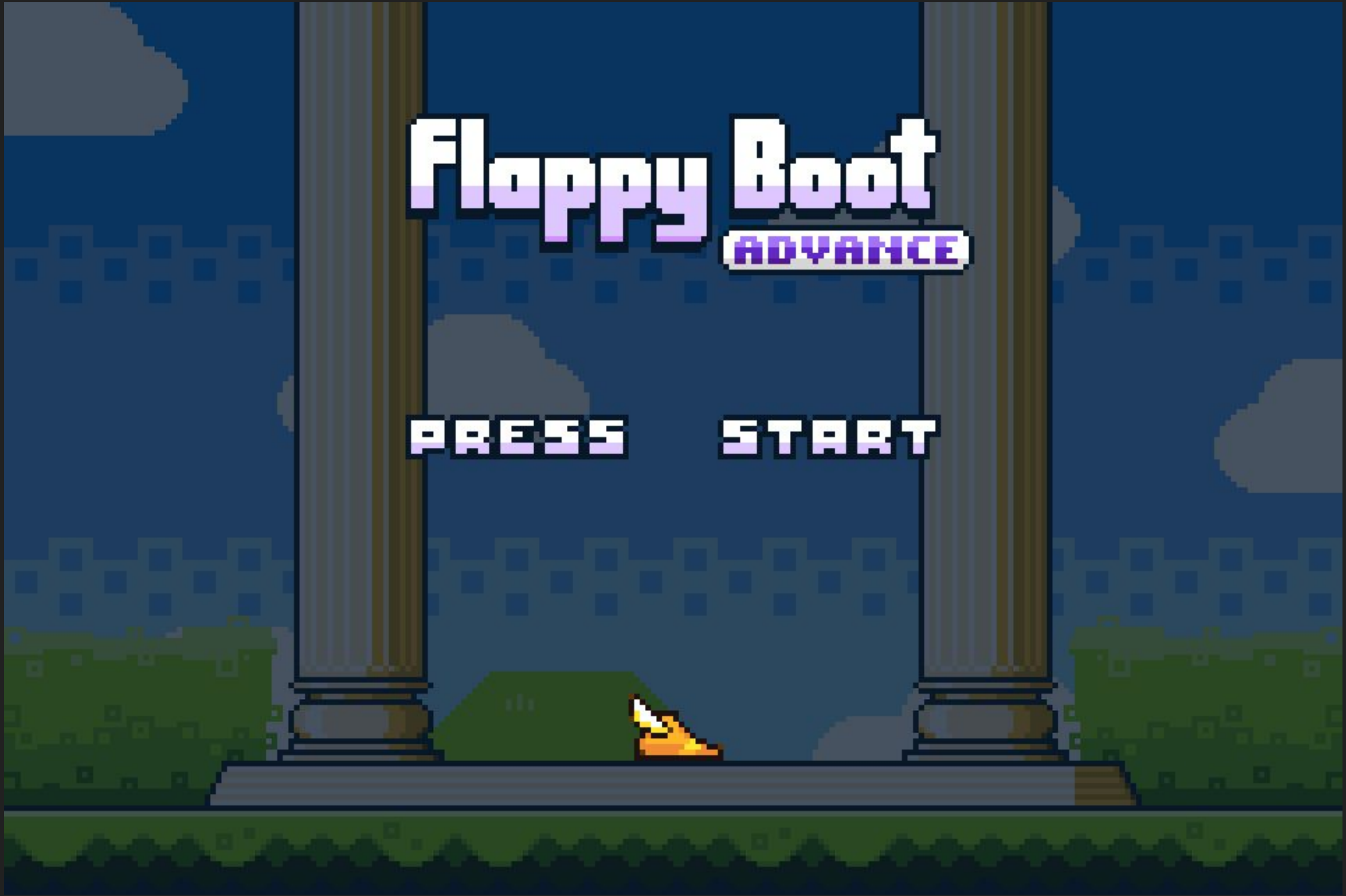
BG 2
256×256

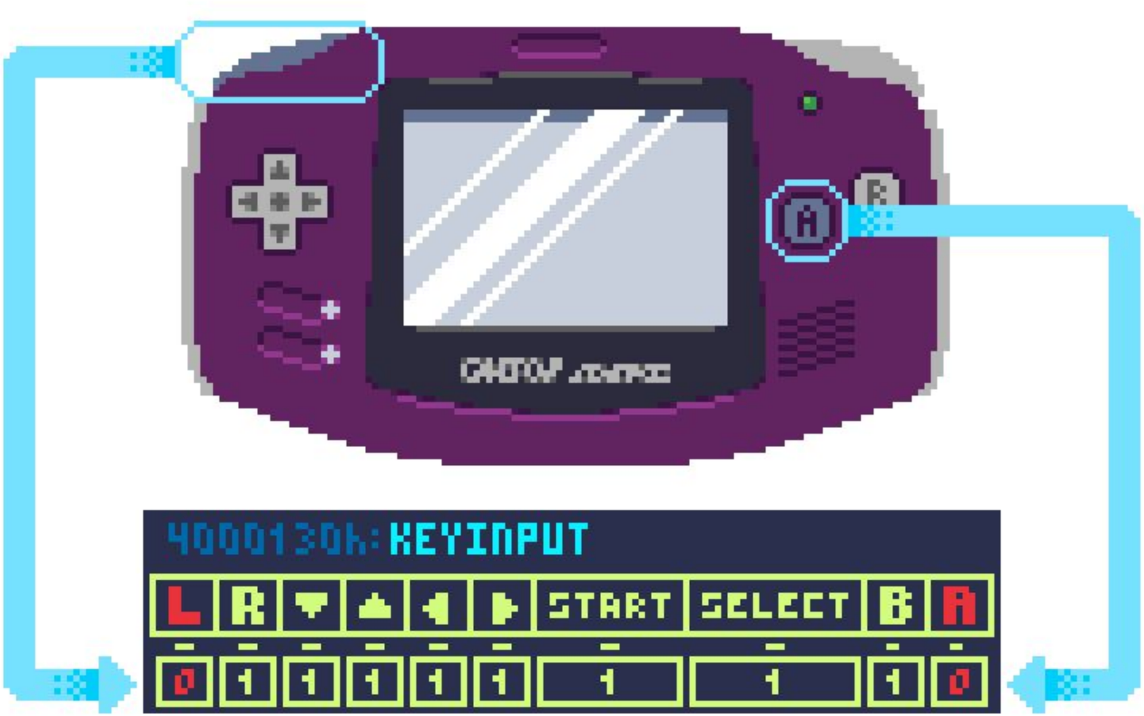
FINAL



DRAWING SPRITES

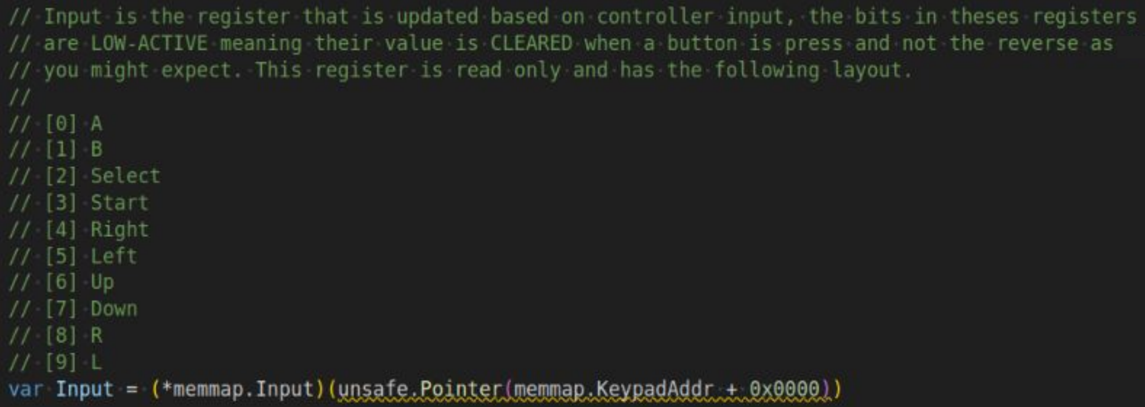



Floppy Boot
ADVANCE
PRESS START

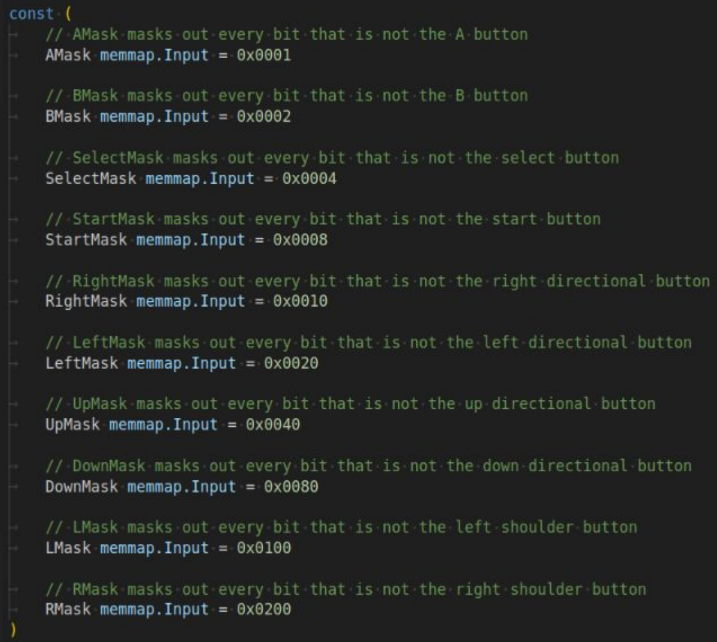


4000130h:KEYINPUT

L	R	▼	▲	◀	▶	START	SELECT	B	A
0	1	1	1	1	1	1	1	1	0



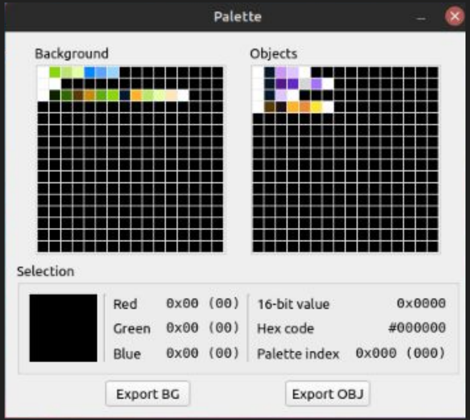
```go
// Input is the register that is updated based on controller input, the bits in theses registers
// are LOW-ACTIVE meaning their value is CLEARED when a button is press and not the reverse as
// you might expect. This register is read only and has the following layout.
//
// [0] A
// [1] B
// [2] Select
// [3] Start
// [4] Right
// [5] Left
// [6] Up
// [7] Down
// [8] R
// [9] L
var Input = (*memmap.Input)(unsafe.Pointer(memmap.KeypadAddr + 0x0000))
```



```go
const (
	// AMask masks out every bit that is not the A button
	AMask memmap.Input = 0x0001

	// BMask masks out every bit that is not the B button
	BMask memmap.Input = 0x0002

	// SelectMask masks out every bit that is not the select button
	SelectMask memmap.Input = 0x0004

	// StartMask masks out every bit that is not the start button
	StartMask memmap.Input = 0x0008

	// RightMask masks out every bit that is not the right directional button
	RightMask memmap.Input = 0x0010

	// LeftMask masks out every bit that is not the left directional button
	LeftMask memmap.Input = 0x0020

	// UpMask masks out every bit that is not the up directional button
	UpMask memmap.Input = 0x0040

	// DownMask masks out every bit that is not the down directional button
	DownMask memmap.Input = 0x0080

	// LMask masks out every bit that is not the left shoulder button
	LMask memmap.Input = 0x0100

	// RMask masks out every bit that is not the right shoulder button
	RMask memmap.Input = 0x0200
)
```

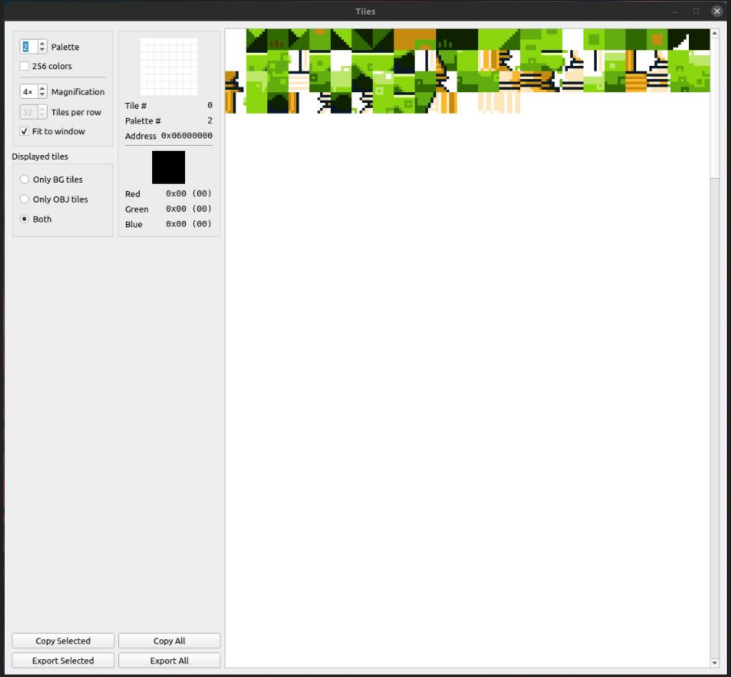
EMULATION

mGBA - 0.10.0
File
Emulation
Audio/Video
Tools
mGBA
POWER
START
SELECT
B
A



Selection

Red 0x00 (00)
Green 0x00 (00)
Blue 0x00 (00)
16-bit value 0x0000
Hex code #000000
Palette index 0x000 (000)

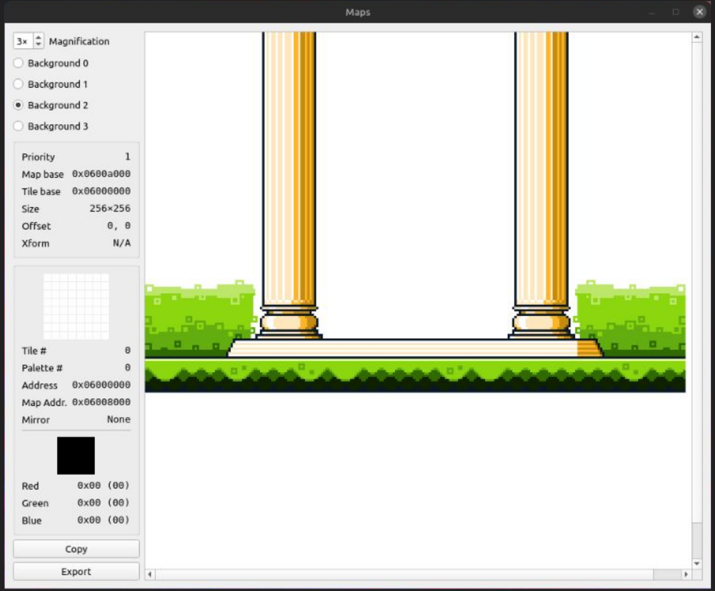


2 Palette
☐ 256 colors
4× Magnification
32 Tiles per row
☑ Fit to window

Displayed tiles
○ Only BG tiles
○ Only OBJ tiles
● Both

Tile # 0
Palette # 2
Address 0x06000000

Red 0x00 (00)
Green 0x00 (00)
Blue 0x00 (00)

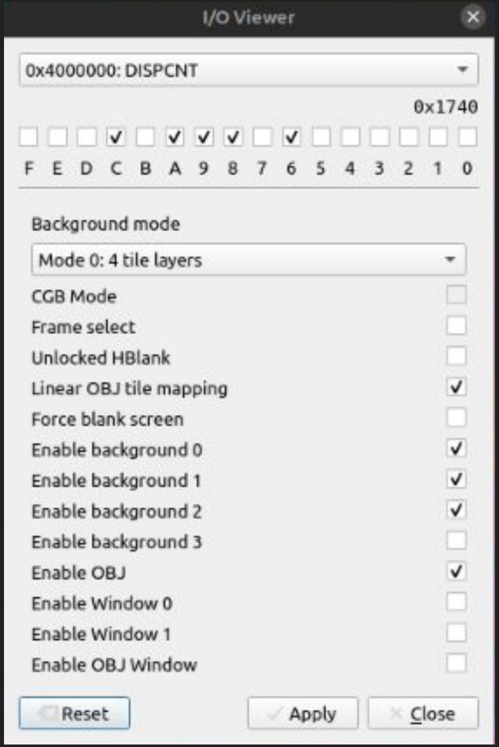


3× Magnification
○ Background 0
○ Background 1
● Background 2
○ Background 3

Priority 1
Map base 0x0600a000
Tile base 0x06000000
Size 256×256
Offset 0, 0
Xform N/A

Tile # 0
Palette # 0
Address 0x06000000
Map Addr. 0x06008000
Mirror None

Red 0x00 (00)
Green 0x00 (00)
Blue 0x00 (00)



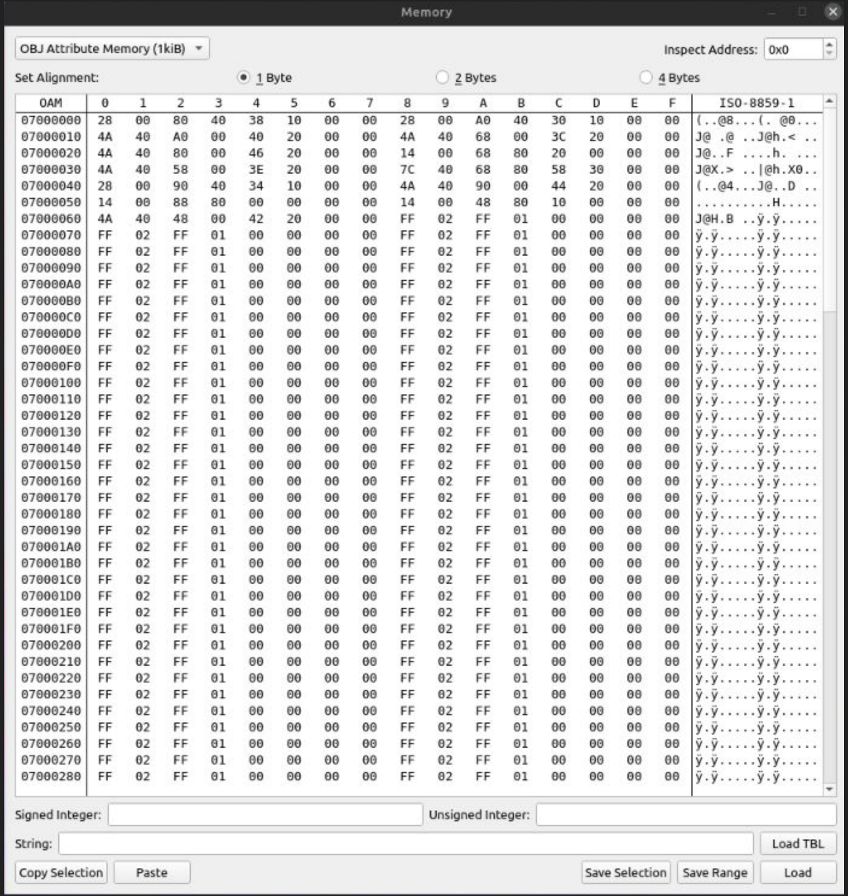
0x4000000: DISPCNT

0x1740

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
□	□	□	☑	□	☑	☑	☑	□	☑	□	□	□	□	□	□

Background mode: Mode 0: 4 tile layers

CGB Mode	□
Frame select	□
Unlocked HBlank	□
Linear OBJ tile mapping	☑
Force blank screen	□
Enable background 0	☑
Enable background 1	☑
Enable background 2	☑
Enable background 3	□
Enable OBJ	☑
Enable Window 0	□
Enable Window 1	□
Enable OBJ Window	□



OBJ Attribute Memory (1kiB)

Inspect Address: 0x0

Set Alignment: ● 1 Byte ○ 2 Bytes ○ 4 Bytes

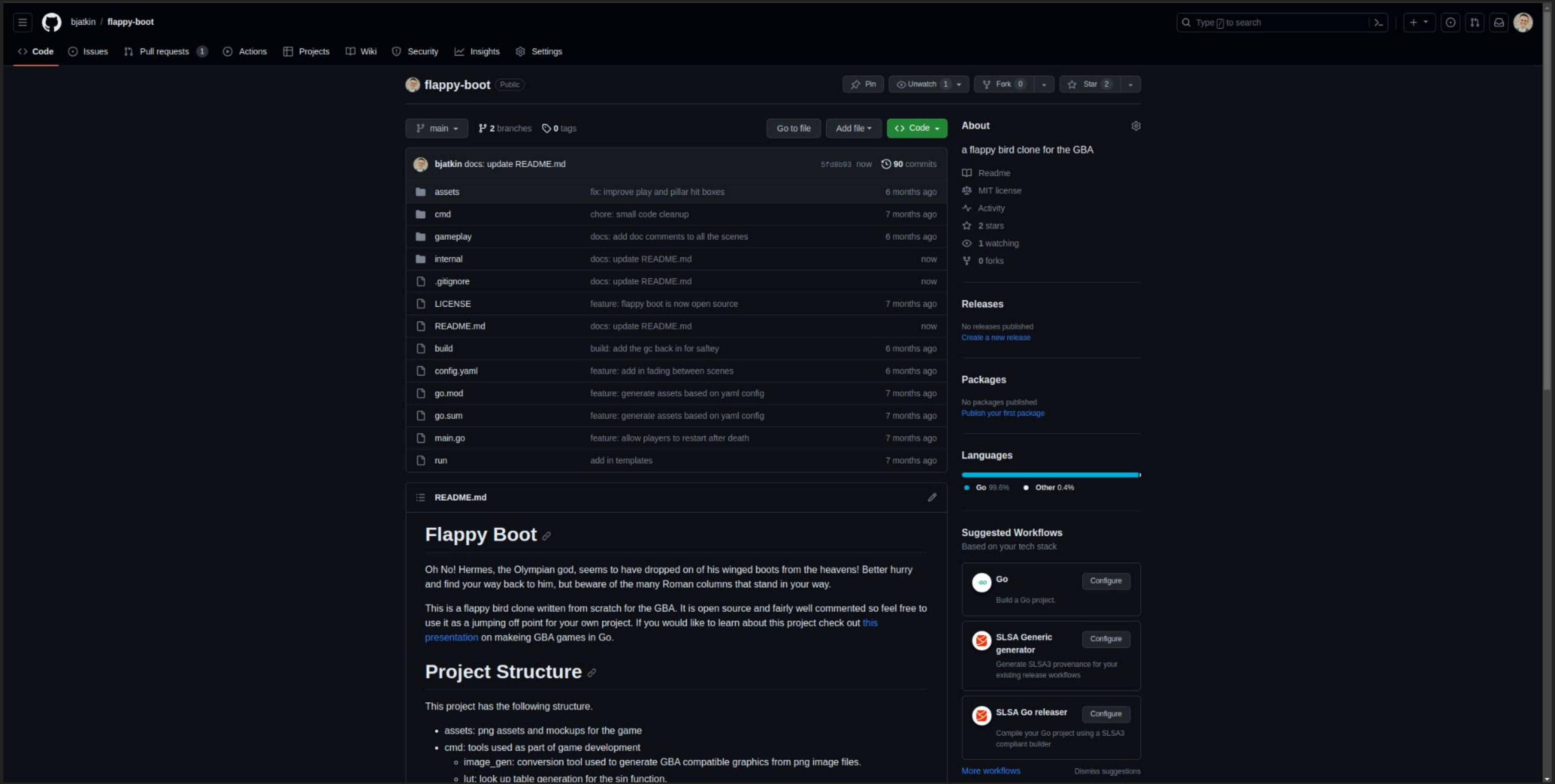
```
OAM       0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   ISO-8859-1
07000000  28 00 80 40 38 10 00 00 28 00 A0 40 30 10 00 00  (..@8...(. @0...
07000010  4A 40 A0 00 40 20 00 00 4A 40 68 00 3C 20 00 00  J@ .@ ..J@h.< ..
07000020  4A 40 80 00 46 20 00 00 14 00 68 80 20 00 00 00  J@..F ....h. ...
07000030  4A 40 58 00 3E 20 00 00 7C 40 68 80 58 30 00 00  J@X.> ..|@h.X0..
07000040  28 00 90 40 34 10 00 00 4A 40 90 00 44 20 00 00  (..@4...J@..D ..
07000050  14 00 88 80 00 00 00 00 14 00 48 80 10 00 00 00  ..........H.....
07000060  4A 40 48 00 42 20 00 00 FF 02 FF 01 00 00 00 00  J@H.B ..ÿ.ÿ.....
07000070  FF 02 FF 01 00 00 00 00 FF 02 FF 01 00 00 00 00  ÿ.ÿ.....ÿ.ÿ.....
07000080  FF 02 FF 01 00 00 00 00 FF 02 FF 01 00 00 00 00  ÿ.ÿ.....ÿ.ÿ.....
07000090  FF 02 FF 01 00 00 00 00 FF 02 FF 01 00 00 00 00  ÿ.ÿ.....ÿ.ÿ.....
070000A0  FF 02 FF 01 00 00 00 00 FF 02 FF 01 00 00 00 00  ÿ.ÿ.....ÿ.ÿ.....
070000B0  FF 02 FF 01 00 00 00 00 FF 02 FF 01 00 00 00 00  ÿ.ÿ.....ÿ.ÿ.....
070000C0  FF 02 FF 01 00 00 00 00 FF 02 FF 01 00 00 00 00  ÿ.ÿ.....ÿ.ÿ.....
070000D0  FF 02 FF 01 00 00 00 00 FF 02 FF 01 00 00 00 00  ÿ.ÿ.....ÿ.ÿ.....
070000E0  FF 02 FF 01 00 00 00 00 FF 02 FF 01 00 00 00 00  ÿ.ÿ.....ÿ.ÿ.....
070000F0  FF 02 FF 01 00 00 00 00 FF 02 FF 01 00 00 00 00  ÿ.ÿ.....ÿ.ÿ.....
07000100  FF 02 FF 01 00 00 00 00 FF 02 FF 01 00 00 00 00  ÿ.ÿ.....ÿ.ÿ.....
07000110  FF 02 FF 01 00 00 00 00 FF 02 FF 01 00 00 00 00  ÿ.ÿ.....ÿ.ÿ.....
07000120  FF 02 FF 01 00 00 00 00 FF 02 FF 01 00 00 00 00  ÿ.ÿ.....ÿ.ÿ.....
07000130  FF 02 FF 01 00 00 00 00 FF 02 FF 01 00 00 00 00  ÿ.ÿ.....ÿ.ÿ.....
07000140  FF 02 FF 01 00 00 00 00 FF 02 FF 01 00 00 00 00  ÿ.ÿ.....ÿ.ÿ.....
07000150  FF 02 FF 01 00 00 00 00 FF 02 FF 01 00 00 00 00  ÿ.ÿ.....ÿ.ÿ.....
07000160  FF 02 FF 01 00 00 00 00 FF 02 FF 01 00 00 00 00  ÿ.ÿ.....ÿ.ÿ.....
07000170  FF 02 FF 01 00 00 00 00 FF 02 FF 01 00 00 00 00  ÿ.ÿ.....ÿ.ÿ.....
07000180  FF 02 FF 01 00 00 00 00 FF 02 FF 01 00 00 00 00  ÿ.ÿ.....ÿ.ÿ.....
07000190  FF 02 FF 01 00 00 00 00 FF 02 FF 01 00 00 00 00  ÿ.ÿ.....ÿ.ÿ.....
070001A0  FF 02 FF 01 00 00 00 00 FF 02 FF 01 00 00 00 00  ÿ.ÿ.....ÿ.ÿ.....
070001B0  FF 02 FF 01 00 00 00 00 FF 02 FF 01 00 00 00 00  ÿ.ÿ.....ÿ.ÿ.....
070001C0  FF 02 FF 01 00 00 00 00 FF 02 FF 01 00 00 00 00  ÿ.ÿ.....ÿ.ÿ.....
070001D0  FF 02 FF 01 00 00 00 00 FF 02 FF 01 00 00 00 00  ÿ.ÿ.....ÿ.ÿ.....
070001E0  FF 02 FF 01 00 00 00 00 FF 02 FF 01 00 00 00 00  ÿ.ÿ.....ÿ.ÿ.....
070001F0  FF 02 FF 01 00 00 00 00 FF 02 FF 01 00 00 00 00  ÿ.ÿ.....ÿ.ÿ.....
07000200  FF 02 FF 01 00 00 00 00 FF 02 FF 01 00 00 00 00  ÿ.ÿ.....ÿ.ÿ.....
07000210  FF 02 FF 01 00 00 00 00 FF 02 FF 01 00 00 00 00  ÿ.ÿ.....ÿ.ÿ.....
07000220  FF 02 FF 01 00 00 00 00 FF 02 FF 01 00 00 00 00  ÿ.ÿ.....ÿ.ÿ.....
07000230  FF 02 FF 01 00 00 00 00 FF 02 FF 01 00 00 00 00  ÿ.ÿ.....ÿ.ÿ.....
07000240  FF 02 FF 01 00 00 00 00 FF 02 FF 01 00 00 00 00  ÿ.ÿ.....ÿ.ÿ.....
07000250  FF 02 FF 01 00 00 00 00 FF 02 FF 01 00 00 00 00  ÿ.ÿ.....ÿ.ÿ.....
07000260  FF 02 FF 01 00 00 00 00 FF 02 FF 01 00 00 00 00  ÿ.ÿ.....ÿ.ÿ.....
07000270  FF 02 FF 01 00 00 00 00 FF 02 FF 01 00 00 00 00  ÿ.ÿ.....ÿ.ÿ.....
07000280  FF 02 FF 01 00 00 00 00 FF 02 FF 01 00 00 00 00  ÿ.ÿ.....ÿ.ÿ.....
```

Signed Integer:

Unsigned Integer:

String:

PUBLISHING



bjatkin docs: update README.md 5fd8b93 now 90 commits

assets	fix: improve play and pillar hit boxes	6 months ago
cmd	chore: small code cleanup	7 months ago
gameplay	docs: add doc comments to all the scenes	6 months ago
internal	docs: update README.md	now
.gitignore	docs: update README.md	now
LICENSE	feature: flappy boot is now open source	7 months ago
README.md	docs: update README.md	now
build	build: add the gc back in for saftey	6 months ago
config.yaml	feature: add in fading between scenes	6 months ago
go.mod	feature: generate assets based on yaml config	7 months ago
go.sum	feature: generate assets based on yaml config	7 months ago
main.go	feature: allow players to restart after death	7 months ago
run	add in templates	7 months ago

README.md

Flappy Boot

Oh No! Hermes, the Olympian god, seems to have dropped on of his winged boots from the heavens! Better hurry and find your way back to him, but beware of the many Roman columns that stand in your way.

This is a flappy bird clone written from scratch for the GBA. It is open source and fairly well commented so feel free to use it as a jumping off point for your own project. If you would like to learn about this project check out this presentation on makeing GBA games in Go.

Project Structure

This project has the following structure.

- assets: png assets and mockups for the game
- cmd: tools used as part of game development
  - image_gen: conversion tool used to generate GBA compatible graphics from png image files.
  - lut: look up table generation for the sin function.

About

a flappy bird clone for the GBA

- Readme
- MIT license
- Activity
- 2 stars
- 1 watching
- 0 forks

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

Languages

- Go 99.6%
- Other 0.4%

Suggested Workflows

Based on your tech stack

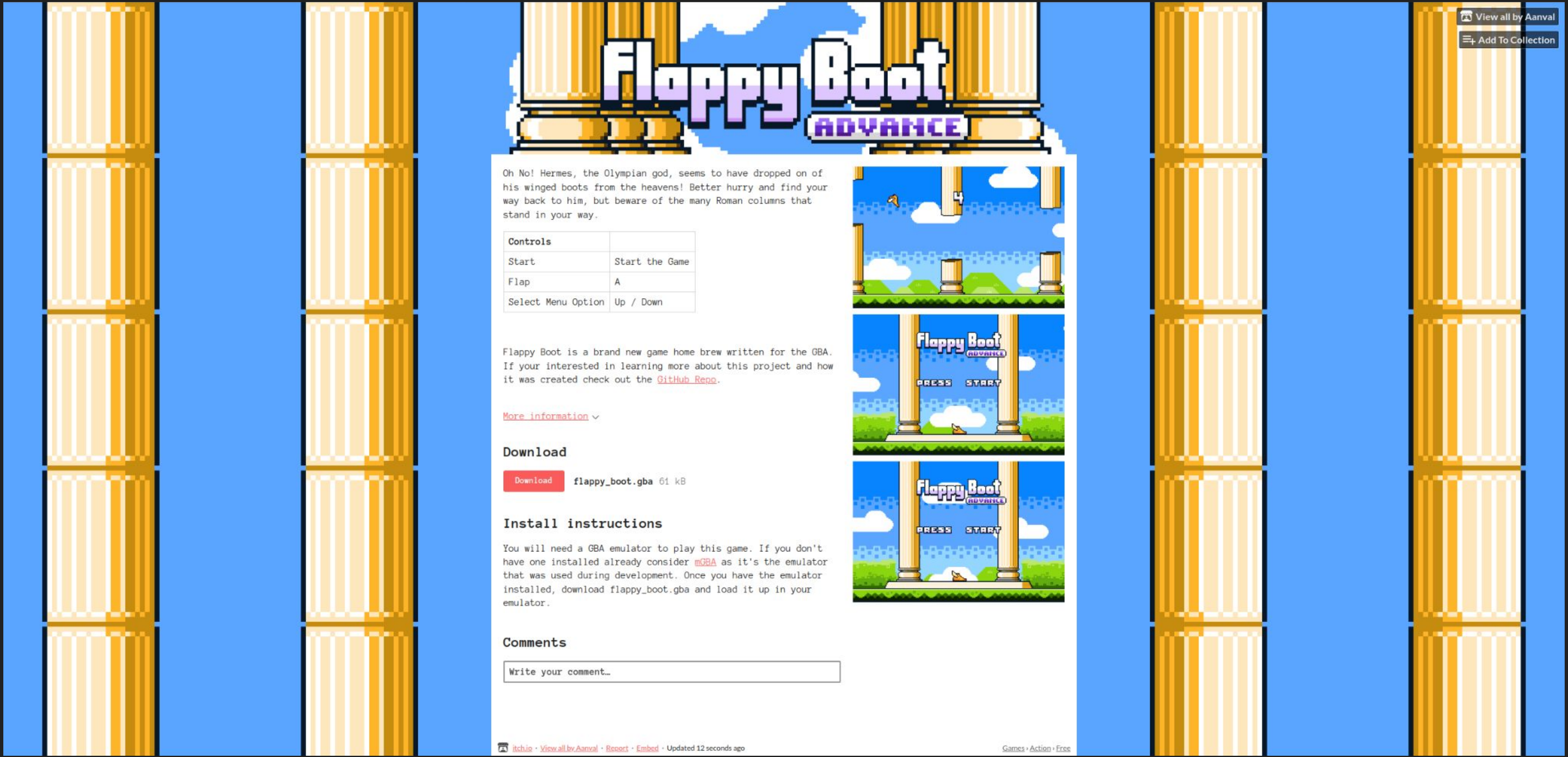
Go — Configure
Build a Go project.

SLSA Generic generator — Configure
Generate SLSA3 provenance for your existing release workflows

SLSA Go releaser — Configure
Compile your Go project using a SLSA3 compliant builder

More workflows
Dismiss suggestions

https://github.com/bjatkin/flappy-boot



Oh No! Hermes, the Olympian god, seems to have dropped on of his winged boots from the heavens! Better hurry and find your way back to him, but beware of the many Roman columns that stand in your way.

Controls	
Start	Start the Game
Flap	A
Select Menu Option	Up / Down

Flappy Boot is a brand new game home brew written for the GBA. If your interested in learning more about this project and how it was created check out the GitHub Repo.

More information

Download

Download flappy_boot.gba 61 kB

Install instructions

You will need a GBA emulator to play this game. If you don't have one installed already consider mGBA as it's the emulator that was used during development. Once you have the emulator installed, download flappy_boot.gba and load it up in your emulator.

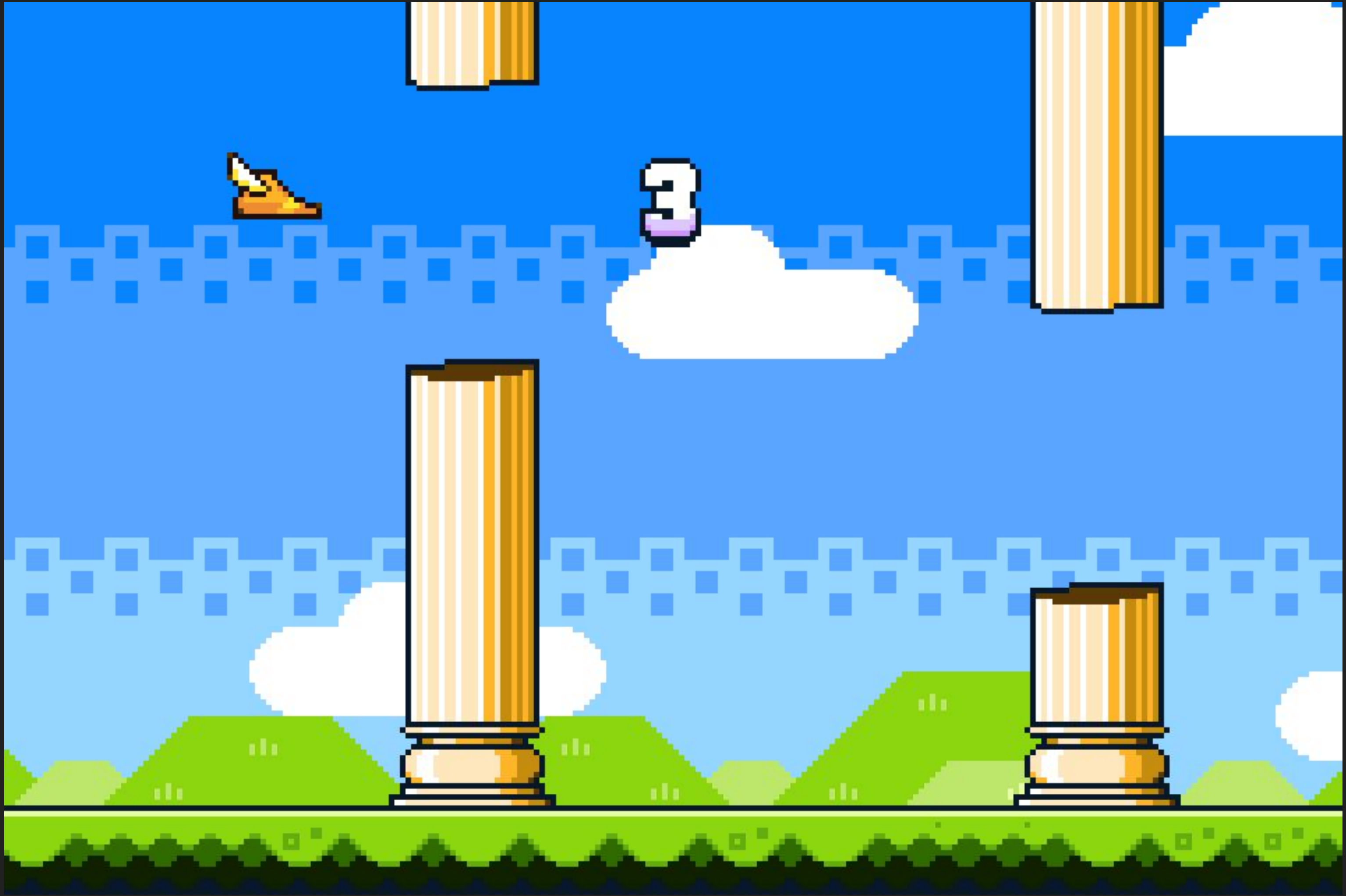
Comments

Write your comment…

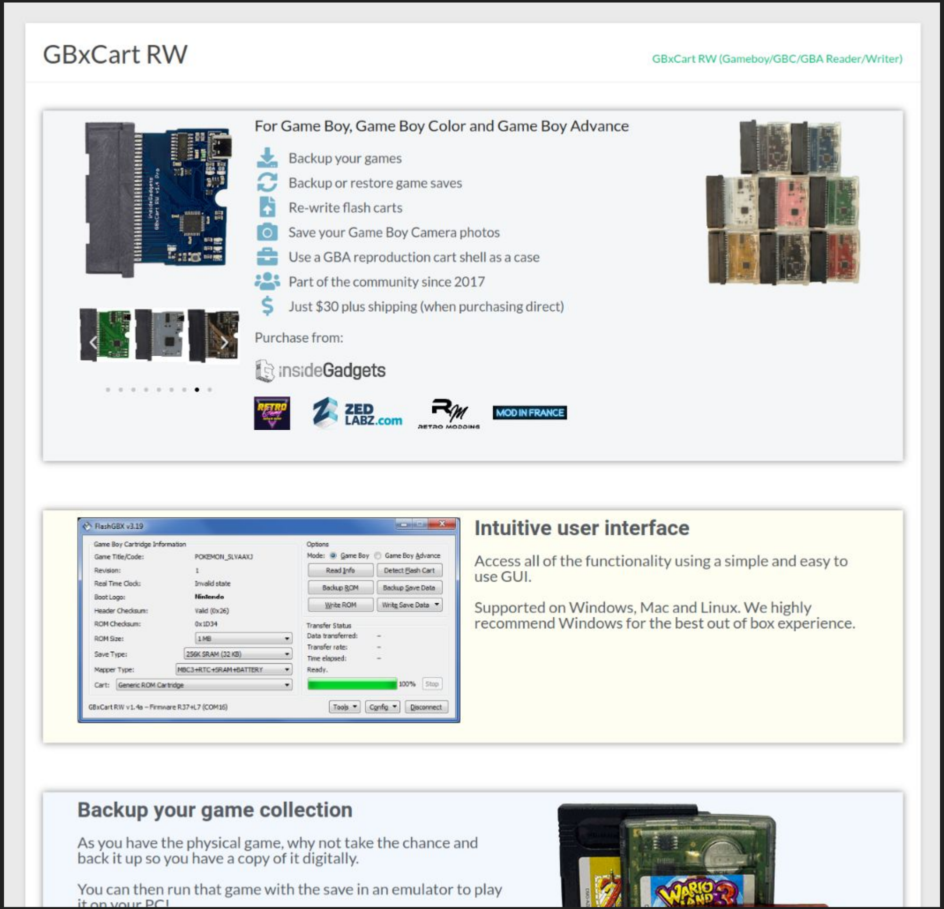
itch.io · View all by Aanval · Report · Embed · Updated 12 seconds ago

Games › Action › Free

https://aanval.itch.io/flappy-boot-advance


3

FlappyBoot
A
B



GBxCart RW

GBxCart RW (Gameboy/GBC/GBA Reader/Writer)

For Game Boy, Game Boy Color and Game Boy Advance

- Backup your games
- Backup or restore game saves
- Re-write flash carts
- Save your Game Boy Camera photos
- Use a GBA reproduction cart shell as a case
- Part of the community since 2017
- Just $30 plus shipping (when purchasing direct)

Purchase from:

insideGadgets

FlashGBX v3.19

Game Boy Cartridge Information

Game Title/Code: POKEMON_SLVAAXJ
Revision: 1
Real Time Clock: Invalid state
Boot Logo: **Nintendo**
Header Checksum: Valid (0x26)
ROM Checksum: 0x1D34
ROM Size: 1 MB
Save Type: 256K SRAM (32 KB)
Mapper Type: MBC3+RTC+SRAM+BATTERY
Cart: Generic ROM Cartridge

Options

Mode: ● Game Boy ○ Game Boy Advance

Transfer Status

Data transferred: –
Transfer rate: –
Time elapsed: –
Ready.

100%

GBxCart RW v1.4a – Firmware R37+L7 (COM16)

Intuitive user interface

Access all of the functionality using a simple and easy to use GUI.

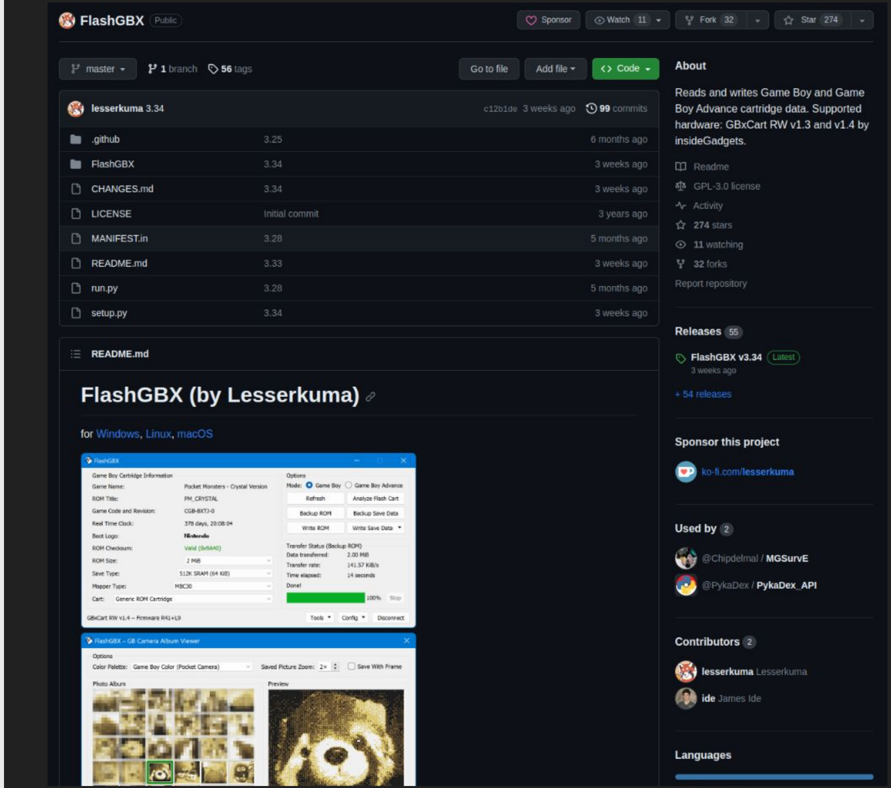
Supported on Windows, Mac and Linux. We highly recommend Windows for the best out of box experience.

Backup your game collection

As you have the physical game, why not take the chance and back it up so you have a copy of it digitally.

You can then run that game with the save in an emulator to play it on your PC!

https://www.gbxcart.com/



FlashGBX Public

1 branch 56 tags

lesserkuma 3.34 c12b1de 3 weeks ago 99 commits

.github	3.25	6 months ago
FlashGBX	3.34	3 weeks ago
CHANGES.md	3.34	3 weeks ago
LICENSE	Initial commit	3 years ago
MANIFEST.in	3.28	5 months ago
README.md	3.33	3 weeks ago
run.py	3.28	5 months ago
setup.py	3.34	3 weeks ago

README.md

FlashGBX (by Lesserkuma)

for Windows, Linux, macOS

FlashGBX

Game Boy Cartridge Information

Game Name: Pocket Monsters - Crystal Version
ROM Title: PM_CRYSTAL
Game Code and Revision: CGB-BXTJ-0
Real Time Clock: 378 days, 20:08:04
Boot Logo: **Nintendo**
ROM Checksum: Valid (0x9A40)
ROM Size: 2 MiB
Save Type: 512K SRAM (64 KiB)
Mapper Type: MBC30
Cart: Generic ROM Cartridge

Options

Mode: ● Game Boy ○ Game Boy Advance

Transfer Status (Backup ROM)

Data transferred: 2.00 MiB
Transfer rate: 141.57 KiB/s
Time elapsed: 14 seconds
Done!

100%

GBxCart RW v1.4 – Firmware R41+L9

FlashGBX – GB Camera Album Viewer

Options

Color Palette: Game Boy Color (Pocket Camera) Saved Picture Zoom: 2× □ Save With Frame

Photo Album

Preview

About

Reads and writes Game Boy and Game Boy Advance cartridge data. Supported hardware: GBxCart RW v1.3 and v1.4 by insideGadgets.

Readme
GPL-3.0 license
Activity
274 stars
11 watching
32 forks
Report repository

Releases 55

FlashGBX v3.34 Latest
3 weeks ago

+ 54 releases

Sponsor this project

ko-fi.com/lesserkuma

Used by 2

@Chipdelmal / MGSurvE
@PykaDex / PykaDex_API

Contributors 2

lesserkuma Lesserkuma
ide James Ide

Languages

https://github.com/lesserkuma/FlashGBX

Questions?