Reference.

Official Documentation of the Gamebuino Meta.

For use this document, click on the page number in this summary for going directly to the page. For return to the summary, click on the Gamebuino logo at the bottom of the page.

Summary.

Technical Specs.

Technical Specification.

Specifications

The Gamebuino META lets gamers to play the great pixelated games they loved on a compact device and learn programming to make their own games. The Gamebuino META fits in a pocket, packs plenty of free exclusive games, and the battery will last through a continuous day of gaming. An optional microSD card can fit more games than you can play, and players can switch between games in a matter of seconds.

Microcontroller
ATSAMD21, 32bit ARM Cortex M0+, 256KB flash, 32KB RAM, same as Arduino Zero

Display
1.8", 80x64 and 160x128 pixels in full RGB565 16-bit colors. 25 fps refresh rate by default, up to 50 fps.

Battery
LiPo 900mAh, charged through micro USB B port (like most phones).

Sound
10bit DAC, mutli-channel 8bit WAV playback, 2.5W class-D audio amp, audio jack 3.5mm, 1W speaker.

Back lights
8 independently controllable RGB LEDs for light effects.

Buttons
D-pad, A, B, Menu, Home.

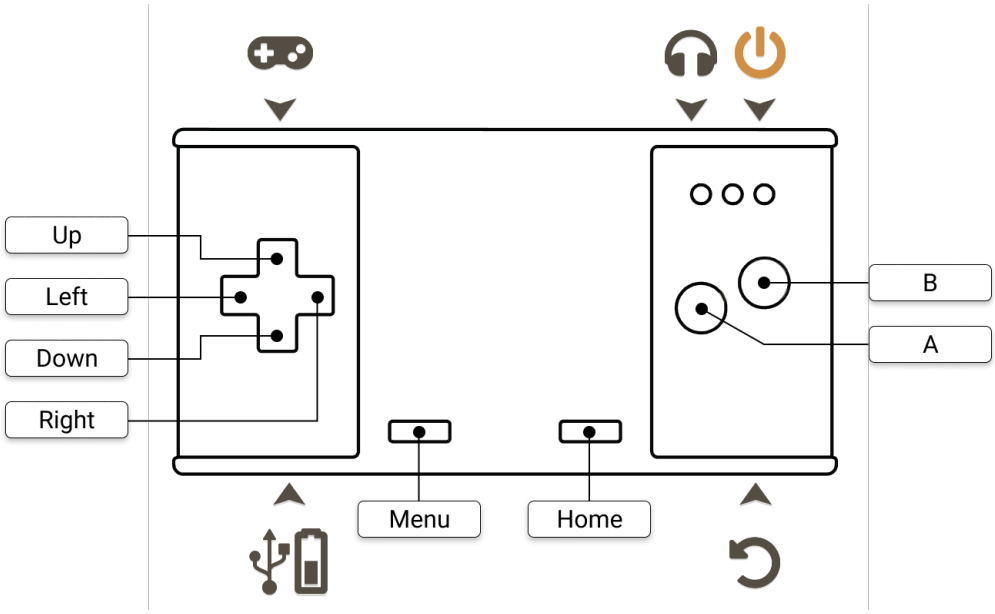
Backpacks (aka shields)
All GPIOs you have on an Arduino Zero are free to use on the back of the console. The developer backpack is compatible with Arduino Shields.

Warning : Gamebuino runs at 3.3V, it's not 5V compatible. The DAC output is used for sound. SPI slaves are present on the bus, but won't prevent you from using it.

MicroSD card socket
FAT and FAT32 compatible. Optional microSD card with plenty of games provided with the *deluxe early bird* and *deluxe* perks. Change between games on the go, not computer required!

Technical Specification.

How to use your Gamebuino



- **Power** Slide switch to turn your Gamebuino on or off.
- **SD card** Where all your games go. You can use the provided micro-USB adapter to add games by following this method.
- **Audio jack:** The 3.5mm audio jack to plug in headphones or external speakers.
- **USB:** A micro-USB port used to charge the battery and develop game as well. Nearly all standard microUSB cables are compatible.
- **Reset** Single tap to reset. Double tap to flash the loader from the SD card if available, otherwise, the console will enter the bootloader.
- **A button** Select / Primary action
- **B button** Cancel / Secondary action
- **Menu** Game-dependent menu, can be used to open inventory, see the map, etc.
- **Home** Exit a game, adjust sound volume & light intensity, take screenshots and screen recordings.

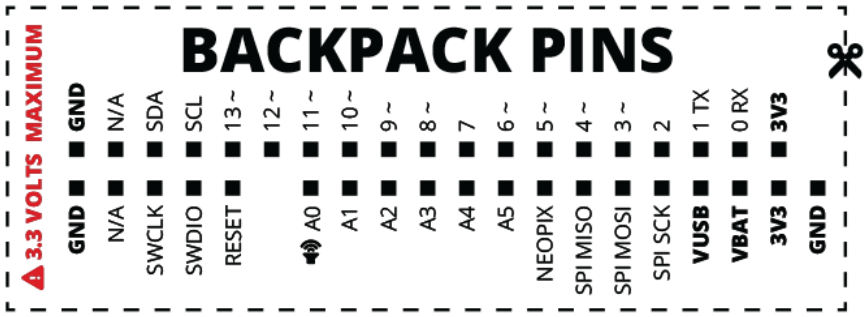
Programming

Your console was not only designed as a gaming console. We took the time to turn this product into a creation tool. You can create your own games in C++. Setting up the environment is simple, and the possibilities of games are endless. Making games on the Gamebuino is for all levels, beginners and experts alike.

Backpacks

On the back of your Gamebuino, there are a series of input and output pins. These are all the Arduino pins. These allow you to do electronics with your console.

This is the layout of the pins:



If this interests you, go ahead and print it, it is of great help.

Core.

Core.

gb.begin

Description

```
void gb.begin()
```

gb.begin initializes the Gamebuino. It should be called once at the beginning of the setup() function.

Parameters

none

Returns

none

Example

```
#include <Gamebuino-Meta.h>

void setup(){
  // initialize the Gamebuino object
  gb.begin();
}

void loop(){
  while(!gb.update());
  gb.display.clear();
  gb.display.println("Hello world");
}
```

Core.

gb.waitForUpdate

Description

```
void gb.waitForUpdate()
```

gb.waitForUpdate waits until a new screen update is needed. This also handles sound updates, key press readings....everything!

Parameters

none

Returns

none

Example

```
#include <Gamebuino-Meta.h>
void setup() {
  gb.begin();
}
void loop() {
  // let's wait for the next update to have a stable framerate
  gb.waitForUpdate();
  gb.display.clear();
  gb.display.print("Hello World!");
}
```

Core.

gb.update

Description

```
bool gb.update()
```

gb.update makes sure that the Gamebuino Meta runs at a constant framerate. It will return true whenever a next frame is ready to render. Default framerate: 25 fps, settable via gb.setFrameRate()

It also takes care of updating buttons, sound, sending out the screen, etc.

Parameters

none

Returns

bool: true if it is time to render the next frame

Example

```
#include <Gamebuino-Meta.h>

void setup(){
  // initialize the Gamebuino object
  gb.begin();
}

void loop(){
  while(!gb.update());
  gb.display.clear();
  gb.display.println("Hello world");
}
```

Core.

gb.getCpuLoad

Description

```
uint8_t gb.getCpuLoad()
```

gb.getCpuLoad returns the current load of the CPU in percent. So, if it returns 50, that means it took 50% of the time it had to create a frame. CPU usage above 100% means that your game is running at a slower framerate than set. And for values higher than 255%, gb.getCpuLoad returns incorrect values (uint8_t overflow).

Parameters

none

Returns

uint8_t: percentage of CPU usage

Example

```
#include <Gamebuino-Meta.h>

void setup() {
  gb.begin();
}

void loop() {
  while(!gb.update());
  gb.display.clear();

  // here we fetch the CPU usage percentage
  uint8_t load = gb.getCpuLoad();
  gb.display.print("CPU:");
  gb.display.print(load);
  gb.display.println("%");
}
```

Core.

gb.getFreeRam

Description

```
uint16_t gb.getFreeRam()
```

gb.getFreeRam returns the amount of free RAM in bytes.

Parameters

none

Returns

uint16_t: free RAM in bytes

Example

```
#include <Gamebuino-Meta.h>

void setup() {
  gb.begin();
}

void loop() {
  while(!gb.update());
  gb.display.clear();

  // fetch the amount of free RAM
  uint16_t ram = gb.getFreeRam();
  gb.display.print("RAM:");
  gb.display.println(ram);
}
```

Core.

gb.getDefaultName

Description

```
void gb.getDefaultName( char* string )
```

gb.getDefaultName fetches the default player name into a string, as configurable in loader.bin
This function is slow, do not call it in a continuous loop!

Parameters

char* string: the string in which to fetch the default name, it must be char[13] or larger

Returns

none

Example

```
#include <Gamebuino-Meta.h>

char name[13];
void setup() {
  gb.begin();

  // load the default name
  gb.getDefaultName(name);
}

void loop() {
  while(!gb.update());
  gb.display.clear();
  gb.display.println(name);
}
```

Core.

gb.createColor

Description

```
Color gb.createColor( uint8_t red , uint8_t green , uint8_t blue )
```

gb.createColor creates a Color object value based on common RGB565 values.

Parameters

- uint8_t red: proportion of red (min: 0, max: 255)
- uint8_t green: proportion of green (min: 0, max: 255)
- uint8_t blue: proportion of blue (min: 0, max: 255)

Returns

Color: the RGB565 color which is closest to the one provided

Example

```
#include <Gamebuino-Meta.h>

void setup() {
  gb.begin();
}

void loop() {
  while(!gb.update());
  gb.display.clear();

  // create the color (#FFC526, some shade of orange)
  Color c = gb.createColor(255, 197, 38);
  gb.display.setColor(c);
  gb.display.println("Hello World");
}
```

Core.

gb.setScreenRotation

Description

```
void gb.setScreenRotation( Rotation r )
```

gb.setScreenRotation sets the rotation of the screen. Default is down. After rotating the screen it is highly recommended to clear the display with gb.display.clear().

Parameters

- Rotation r: Rotation what to set the screen to. Possible values are: ROTATION_LEFT, ROTATION_UP, ROTATION_RIGHT, ROTATION_DOWN

Returns

none

Example

```
#include <Gamebuino-Meta.h>

void setup() {
  gb.begin();

  // rotate the screen right
  gb.setScreenRotation(ROTATION_RIGHT);
}

void loop() {
  while(!gb.update());
  gb.display.clear();
  gb.display.println("Hello World");
}
```

Core.

getScreenRotation

Description

```
Rotation gb.getScreenRotation()
```

gb.getScreenRotation returns the current rotation of the screen.

Parameters

none

Returns

Rotation - current rotation of the screen

Example

N/A

Core.

gb.frameCount

Description

```
Random gb.pickRandomSeed
```

gb.pickRandomSeed lorem ipsum.

Parameters

Lorem ipsum

Returns

Lorem ipsum

Example

```
#include <Gamebuino-Meta.h>

void setup() {
  gb.begin();
}

void loop() {
  while(!gb.update());
  // check if A was pressed
 gb.pickRandomSeed
}
```

Core.

gb.pickRandomSeed

Description

```
Lorem gb.pickRandomSeed
```

gb.pickRandomSeed lorem ipsum.

Parameters

- Lorem ipsum

Returns

Lorem ipsum

Example

```
#include <Gamebuino-Meta.h>

void setup() {
  gb.begin();
}

void loop() {
  while(!gb.update());
  // check if A was pressed
 gb.pickRandomSeed
}
```

Buttons.

Buttons.

Buttons names

Description

- BUTTON_A
- BUTTON_B
- BUTTON_MENU (used to be BUTTON_C)
- BUTTON_UP
- BUTTON_RIGHT
- BUTTON_DOWN
- BUTTON_LEFT

Parameters

- Lorem Ipsum

Returns

Lorem Ipsum

Example

```
#include <Gamebuino-Meta.h>

void setup() {
  gb.begin();
}

void loop() {
  while(!gb.update());
  // check if A was pressed
  if (gb.buttons.pressed(BUTTON_A)) {
  // do something
  }
}
```

Buttons.

gb.buttons.pressed

Description

```
bool gb.buttons.pressed( Button button )
```

gb.buttons.pressed returns true once you pressed down that button. It will not return true until the button is released and then pressed again.

Parameters

- Button button: the button to check

Returns

bool: true if the button was pressed down just now

Example

```
#include <Gamebuino-Meta.h>

void setup() {
  gb.begin();
}

void loop() {
  while(!gb.update());
  // check if A was pressed
  if (gb.buttons.pressed(BUTTON_A)) {
  // do something
  }
}
```

Buttons.

gb.buttons.released

Description

```
bool gb.buttons.released( Button button )
```

gb.buttons.released returns true if the button has been released just now, else false.

Parameters

- Button button: the button to check

Returns

bool: true if the button was released just now

Example

```
#include <Gamebuino-Meta.h>

void setup() {
  gb.begin();
}

void loop() {
  while(!gb.update());
  // check if the A button was released
  if (gb.buttons.released(BUTTON_A)) {
  // The button was released, handle it
  }
}
```

Buttons.

gb.buttons.held

Description

```
bool gb.buttons.held( Button button , uint16_t time )
```

gb.buttons.held returns true once the button has been held for time amount of frames. It will not return true again until you release the button and hold it down again that long.

Parameters

- Button button: the button to check
- uint16_t time: the number of frames you need to hold that button down

Returns

bool: true if the button has been hold down exactly time frames

Example

```
#include <Gamebuino-Meta.h>

void setup() {
  gb.begin();
}

void loop() {
  while(!gb.update());
  // check if the A button was held down for 25 frames (one second)
  if (gb.buttons.held(BUTTON_A, 25)) {
  // handle it
  }
}
```

Buttons.

gb.buttons.repeat

Description

```
bool gb.buttons.repeat( Button button , uint16_t period )
```

gb.buttons.repeat returns true if the button has been held down for period amount of frames, repetitively. A period of 0 will check if the button is pressed down in this instant

Parameters

- Button button: button to check
- uint16_t period: repetitive period the button needs to be held

Returns

bool: true if the button has been held repetitive for period frames

Example

```
#include <Gamebuino-Meta.h>

void setup() {
  gb.begin();
}

void loop() {
  while(!gb.update());
  // check if the A button is held right now
  if (gb.buttons.repeat(BUTTON_A, 0)) {
  // handle it
  }

  // check if the B button is held for 4 frames, repetitively
  if (gb.buttons.repeat(BUTTON_B, 4)) {
  // handle it
  }
}
```

Buttons.

gb.buttons.timeHeld

Description

```
uint16_t gb.buttons.timeHeld( Button button )
```

gb.buttons.timeHeld returns the number of frames a button has been held down already, or 0xFFFF if the button has just been released.

Parameters

- Button button: the button to check

Returns

uint16_t: number of frames the button has been held down

Example

```
#include <Gamebuino-Meta.h>
void setup() {
  gb.begin();
}
void loop() {
  while(!gb.update());
  gb.display.clear();
  //fetch how long the A button has been held down
  uint16_t time = gb.buttons.timeHeld(BUTTON_A);
  gb.display.println(time);
}
```

Sound.

Sound

gb.sound.play

Description

```
// wav file:
int8_t gb.sound.play(
    [const] char* filename
    [, bool loop = false])

// pattern:
int8_t gb.sound.play(
    [const] uint16_t* pattern
    [, bool loop = false])

// raw buffer:
int8_t gb.sound.play(
    [const] uint8_t* buffer
    [, uint32_t length] [, bool loop = false])

// custom handler:
int8_t gb.sound.play(Sound_Handler* handler [, bool loop = false])
```

gb.sound.play will start playing sound from various sources. They return a track identifier or -1 if failed (no available channel).

- **Wav file** Plays an 8-bit unsigned wav file with a bitrate of 44.1kHz (max. sound frequency of 22050Hz)
- **Pattern** Plays a pattern
- **Raw Buffer** Plays a raw unsigned 8-bit buffer. The length of the buffer is determined automatically. If that fails (as in, your program won't compile) manually specify the length.
- **Custom Handler** Play a custom sound handler

Parameters

- (mixed) source: see above
- bool loop: if true the sound will loop over and over forever

Returns

int8_t: track identifier on success or -1 on failure

Example

```
#include <Gamebuino-Meta.h>
void setup() {
  gb.begin();
}

void loop() {
  while(!gb.update());
  if (gb.buttons.pressed(BUTTON_A)) {
  // let's play TEST.WAV from the sd card!
  gb.sound.play("TEST.WAV");
 }
}
```

Sound.

gb.sound.tone

Description

```
int8_t gb.sound.tone( uint32_t frequency [, int32_t duration = 0] )
```

gb.sound.tone plays a tone of a certain frequency for duration milliseconds. If duration is zero then it'll play indefinitally. It returns the track identifier or -1 on failure

Parameters

- uint32_t frequency: frequency of the tone to play
- int32_t duration (optional): duration in milliseconds for the tone to play

Returns

int8_t: track identifier or -1 on failure

Example

```
#include <Gamebuino-Meta.h>
void setup() {
  gb.begin();
}
void loop() {
  while(!gb.update());
  if (gb.buttons.pressed(BUTTON_A)) {
  //let's play a tone with a frequency of 10kHz for one second!
  gb.sound.tone(100000, 1000);
  }
}
```

Sound.

gb.sound.playOK

Description

```
int8_t gb.sound.playOK()
```

gb.sound.playOK plays an "OK" sound and returns a track identifier, or -1 on failure

Parameters

none

Returns

int8_t: track identifier or -1 on failure

Example

```
#include <Gamebuino-Meta.h>

void setup() {
  gb.begin();
}

void loop() {
  while(!gb.update());
  if (gb.buttons.pressed(BUTTON_A)) {
  //play the OK sound
  gb.sound.playOK();
  }
}
```

Sound.

gb.sound.playCancel

Description

```
int8_t gb.sound.playCancel()
```

gb.sound.playCancel plays a "Cancel" sound and returns a track identifier, or -1 on failure

Parameters

none

Returns

int8_t: track identifier or -1 on failure

Example

```
#include <Gamebuino-Meta.h>

void setup() {
  gb.begin();
}

void loop() {
  while(!gb.update());
  if (gb.buttons.pressed(BUTTON_A)) {
  //play the Cancel sound
  gb.sound.playCancel();
  }
}
```

Sound.

gb.sound.playTick

Description

```
int8_t gb.sound.playTick()
```

gb.sound.playTick plays a "Tick" sound and returns a track identifier, or -1 on failure

Parameters

none

Returns

int8_t: track identifier or -1 on failure

Example

```
#include <Gamebuino-Meta.h>

void setup() {
  gb.begin();
}

void loop() {
  while(!gb.update());
  if (gb.buttons.pressed(BUTTON_A)) {
  //play the Tick sound
  gb.sound.playTick();
  }
}
```

Sound.

gb.sound.isPlaying

Description

```
bool gb.sound.isPlaying( int8_t trackIdentifier )
```

gb.sound.isPlaying returns true if the track identifier is playing

Parameters

- int8_t trackIdentifier: track to check

Returns

bool: true if the track is playing, otherwise false

Example

```
#include <Gamebuino-Meta.h>

void setup() {
  gb.begin();
}

int8_t track = -1;

void loop() {
  while(!gb.update());
  gb.display.clear();

// check if the track is playing
  if (gb.sound.isPlaying(track)) {
    gb.display.println("Playing");
  }

  if (gb.buttons.pressed(BUTTON_A)) {
    track = gb.sound.playOK();
  }
}
```

Sound.

gb.sound.stop

Description

```
void gb.sound.stop( int8_t trackIdentifier )
```

gb.sound.stop will stop a given channel, as told via the track identifier.

Parameters

- int8_t trackIdentifier: track to stop

Returns

none

Example

```
#include <Gamebuino-Meta.h>

int8_t track = -1;
void setup() {
  gb.begin();
  track = gb.sound.play("TEST.WAV", true);
}

void loop() {
  while(!gb.update());
  if (gb.buttons.pressed(BUTTON_A)) {
// stop our track
    gb.sound.stop(track);
  }
}
```

Sound.

gb.sound.getPos

Description

```
uint32_t gb.sound.getPos( int8_t trackIdentifier )
```

gb.sound.getPos returns the position of a playing track, or 0xFFFFFFFF on failure

Parameters

- int8_t trackIdentifier: track to check

Returns

uint32_t: position of the track playing, or 0xFFFFFFFF if channel not found / not playing / not supporting this

Example

N/A

Sound.

gb.sound.fx

Description

```
void gb.sound.fx( const Sound_FX * const fx )
```

gb.sound.fx will start playing a specified sound effect. As it does so, if another sound effect is currently playing that one is stopped. Sound effects do not use up a channel for playing normal audio. The data itself can easily be created using this tool.

If you want to know more about what a sound_FX is composed of, check out this tutorial.

Parameters

- const Sound_FX * const fx: pointer to the sound pattern to play

Returns

none

Example

```
#include <Gamebuino-Meta.h>

// define your sound effect
const Gamebuino_Meta::Sound_FX my_sfx[] = {
  {Gamebuino_Meta::Sound_FX_Wave::NOISE,1,70,0,0,240,1},
  {Gamebuino_Meta::Sound_FX_Wave::SQUARE,1,0,0,-3,50,5},
  {Gamebuino_Meta::Sound_FX_Wave::NOISE,0,70,0,0,224,1},
};

void setup() {
  gb.begin();
}

void loop() {
  while(!gb.update());

  if (gb.buttons.pressed(BUTTON_A)) {
// play the sound effect
    gb.sound.fx(my_sfx);
  }
```

Save.

Save.

gb.save.config

Description

```
void gb.save.config([uint16_t blocks,] const SaveDefaults* defaults [, uint16_t numDefaults])
```

gb.save.config will configure gb.save with defaults. You can also specify the number of blocks the savefile should have. The length of the defaults array is autodetermined, if possible. If it fails to compile specify the length manually.

Parameters

- uint16_t blocks (optional): number of blocks
- const SaveDefaults* defaults: defaults array
- uint16_t numDefaults (optional): number of defaults

Returns

none

Example

```
#include <Gamebuino-Meta.h>

const SaveDefaults defaults[] = {
  {0, SAVETYPE_INT, 9999, 0},
};

void setup() {
  gb.begin();

// set the save defaults
  gb.save.config(defaults);
}

void loop() {

}
```

Save.

gb.save.get

Description

```
//number:
int32_t gb.save.get(uint16_t block)
//buffer:
void gb.save.get(uint16_t block , void* buffer , uint8_t buffersize)
//object:
void gb.save.get(uint16_t , T object)
```

gb.save.get gets a variable off of the savefile. This operation is slow!

- **Number** You just specify the block number and it returns the integer.
- **Buffer** You need to specify a pointer and the size of the buffer
- **Object** You can specify an arbitrary object which will get populated.

Parameters

- uint16_t block: saveblock to fetch
- for the others: see above

Returns

see above

Example

```
#include <Gamebuino-Meta.h>

void setup() {
  gb.begin();

// fetch a number
  int32_t number = gb.save.get(0);

// fetch a buffer
  uint8_t buffer[10];
  gb.save.get(1, buffer, 10);
}

void loop() {

}
```

Save.

gb.save.del

Description

```
void gb.save.del(uint16_t block)
```

gb.save.del deletes a given block. This operation is slow!

Parameters

- uint16_t block: block to delete

Returns

none

Example

```
#include <Gamebuino-Meta.h>

void setup() {
  gb.begin();

// delete block 0
  gb.save.del(0);
}
void loop() {

}
```

GUI.

GUI.

gb.gui.keyboard

Description

```
// strings:
void gb.gui.keyboard(const char* title , char* text [, uint8_t length])
// Multilang:
void gb.gui.keyboard(const MultiLang* title , char* text [, uint8_t length]
[,uint8_t numLang])
```

gb.gui.keyboard displays a keyboard in which you can enter a message. You can also provide a default text.

Parameters

- const char* | const MultiLang* title: title which to display
- char* text: this acts both as the default text and as the output buffer
- uint8_t length (optional): the length **in characters** of the buffer, meaning if this is e.g. 12 your text buffer should be char[13] due to the added '\0'. If not provided, it is attempted to be autodetermined
- uint8_t numLang (optional): number language entries for MultiLang title. If not provided, it is attempted to autodetermine it

Returns

none

Example

```
#include <Gamebuino-Meta.h>

void setup() {
  gb.begin();

  // ask the player for their favorite animal
  char text[12] = "Foxies";
  gb.gui.keyboard("Favorite Animal?", text);

  gb.display.clear();
  gb.display.println("You entered:");
  gb.display.print(text);
}

void loop() {
  while(!gb.update());
}
```

GUI.

gb.gui.menu

Description

```
// strings:
uint8_t gb.gui.menu(const char* title , const char** items [, uint8_t length])
// Multilang:
uint8_t gb.gui.menu(const MultiLang* title , const MultiLang** items [, uint8_t
length , uint8_t numLang])
```

gb.gui.menu provides a nice graphical menu where you can pick items. It returns the index of the item that was picked by the user.

Parameters

- const char* | const MultiLang* title: Title to display above the menu entries
- const char** | const MultiLang** items: Array of the items of the menu
- uint8_t length (optional): amount of entries in the menu. If not specified, it is attempted to autodetermine it
- uint8_t numLang (optional): number language entries for MultiLang things. If not specified, it is attempted to autodetermine it

Returns

uint8_t: index of the selected entry

Example

```
#include <Gamebuino-Meta.h>
const char* entries[] = {
  "Pizza",
  "Spaghetti",
  "Noodles",
  "Ice Cream",
};
void setup() {
  gb.begin();

// display the menu
  uint8_t entry = gb.gui.menu("Best food?", entries);

  gb.display.clear();
  gb.display.println("You picked:");
  gb.display.print(entries[entry]);
}

void loop() {
  while(!gb.update());
}
```

GUI.

gb.gui.popup

Description

```
// strings:
void gb.gui.popup(const char* text , uint8_t duration)
// Multilang:
void gb.gui.popup(const MultiLang* text , uint8_t duration [, uint8_t numLang])
```

gb.gui.popup displays a popup for ***duration*** frames. Please note that the popup only functions correctly if, during the entirety of its display time, the entire screen is been re-drawn each frame.

Parameters

- const char* | const MultiLang* text: text content of the popup
- uint8_t duration: Duration the popup is displayed (in frames)
- uint8_t numLang (optional): number language entries for MultiLang text. If not provided, it is attempted to autodetermine it

Returns

none

Example

```
#include <Gamebuino-Meta.h>

void setup() {
  gb.begin();
}

const MultiLang fox[] = {
  {LANG_EN, "Fox"},
  {LANG_DE, "Fuchs"},
};

void loop() {
  while(!gb.update());
  gb.display.fill(RED);

  if (gb.buttons.pressed(BUTTON_A)) {
  gb.gui.popup(fox, 50);
  }
```

Language.

Language.

gb.language.get

Description

```
const char* gb.language.get(const MultiLang* lang [, uint8_t length])
```

gb.language.get fetches the pointer off a MultiLang array "lang" and returns it.

Parameters

- const MultiLang* lang: multi-language array
- uint8_t length (optional): length of the array

Returns

const char* - pointer to the localized string

Example

```
#include <Gamebuino-Meta.h>

const MultiLang[] lang_fox = {
  {LANG_EN, "fox"},
  {LANG_DE, "Fuchs"},
};

void setup() {
  gb.begin();
}

void loop() {
  while(!gb.update());
  gb.display.clear();

  // fetch the pointer for the string
  const char* fox = gb.language.get(lang_fox);

  gb.display.println(fox);
}
```

Reference, DangerZone.

All these function should not be used as they are internal and might change with no backward compatibility. Use at your own risk.

Reference,DangerZone.

gb.getTimePerFrame

Description

```
uint8_t gb.getTimePerFrame()
```

gb.getTimePerFrame returns how long each frame is displayed, in milliseconds.

So, for the default of 25 fps that would be 40 milliseconds (40*25=1000)

Parameters

none

Returns

uint8_t: milliseconds per frame

Example

N/A

Reference,DangerZone.

gb.checkHomeMenu

Description

```
void gb.checkHomeMenu()
```

gb.checkHomeMenu checks if the BUTTON_HOME is pressed or our time is up recording and calls the home menu accordingly.

Do not call this manually, gb.update already does this for you.

Parameters

none

Returns

none

Example

N/A

Reference,DangerZone.

gb.homeMenu

Description

```
void gb.homeMenu()
```

gb.homeMenu opens the home menu, which allows you to adjust volume / change contrast / switch games etc.

Do not call this manually, gb.checkHomeMenu does this for you!

Parameters

none

Returns

none

Example

N/A

Reference,DangerZone.

gb.changeGame

Description

```
void gb.changeGame()
```

gb.changeGame triggers the loading of loader.bin off of the SD card, thus allowing you to load other games.

This function is automatically called in gb.homeMenu if the user selects that option, so you will probably not need this.

This function does not return, no code after this will execute!

Parameters

none

Returns

none

Example

```
#include <Gamebuino-Meta.h>

void setup() {
  gb.begin();
}

void loop() {
  if (gb.update()) {
    if (gb.buttons.pressed(BUTTON_A)) {
    // trigger game changing
    gb.changeGame();
    }
  }
}
```

Reference,DangerZone.

gb.titleScreen

Description

```
void gb.titleScreen()
```

gb.titleScreen displays the beginning title with a flashing "A to start". If you call this method manually you are probably doing something wrong! gb.begin calls it automatically on hard power-ons.

Parameters

none

Returns

none

Example

N/A

Reference,DangerZone.

gb.updateDisplay

Description

```
void gb.updateDisplay()
```

gb.updateDisplay draws gb.display onto gb.tft and thus updating the live display.

Do not call this function yourself, gb.update already does so for you.

Parameters

none

Returns

none

Example

```
#include <Gamebuino-Meta.h>

void setup() {
  gb.begin();

  gb.display.clear();
  gb.display.println("Hello World");

  // now update the actual screen
  gb.updateDisplay();
}

void loop() {
}
```

Reference,DangerZone.

gb.buttons.begin

Description

```
void gb.buttons.begin()
```

gb.buttons.begin initializes the buttons.

Do not call this manually, gb.begin does this for you!

Parameters

none

Returns

none

Example

N/A

Reference,DangerZone.

gb.buttons.update

Description

```
void gb.buttons.update()
```

gb.buttons.update updates the internal button state.

Do not call this manually, gb.update already does this for you!

Parameters

none

Returns

none

Example

N/A

Reference,DangerZone.

gb.sound.startEfxOnly

Description

```
void gb.sound.startEfxOnly()
```

gb.sound.startEfxOnly puts sound into effects-only mode. This is used internally for the home menu. I cannot think of any other use for this. So if you are using this for your game you are maybe doing something wrong....

Parameters

none

Returns

none

Example

N/A

Reference,DangerZone.

gb.sound.stopEfxOnly

Description

```
void gb.sound.stopEfxOnly()
```

gb.sound.stopEfxOnly puts sound into normal mode. This is used internally for the home menu. I cannot think of any other use for this. So if you are using this for your game you are maybe doing something wrong...

Parameters

none

Returns

none

Example

N/A

Reference,DangerZone.

gb.sound.begin

Description

```
void gb.sound.begin()
```

gb.sound.begin initializes sound stuff.

Do not call this manually, gb.begin does this for you!

Parameters

none

Returns

none

Example

N/A

Reference,DangerZone.

gb.sound.mute

Description

```
void gb.sound.mute()
```

gb.sound.mute will mute all sounds immediately.

If you use it, it is recommended to fetch the mute-state via gb.sound.isMute first, to know if you have to unmute or not.

If you are planning to add a simple mute/unmute switch to your game: Don't. You can already do that via the home menu!

Parameters

none

Returns

none

Example

N/A

Reference,DangerZone.

gb.sound.unmute

Description

```
void gb.sound.unmute()
```

gb.sound.unmute will unmute all sounds immediately.

If you use it, it is recommended to fetch the mute-state via gb.sound.isMute before mute-ing, to know if you have to unmute or not.

If you are planning to add a simple mute/unmute switch to your game: Don't. You can already do that via the home menu!

Parameters

none

Returns

none

Example

N/A

Reference,DangerZone.

gb.sound.isMute

Description

```
bool gb.sound.isMute()
```

gb.sound.isMute returns if the sound is currently muted.

Parameters

none

Returns

bool: is the sound muted?

Example

N/A

Reference,DangerZone.

gb.sound.setVolume

Description

```
void gb.sound.setVolume( uint8_t volume )
```

gb.sound.setVolume sets the current volume output. volume may be anything from 0 to 8.

If you are using this to have the user being able to set the output volume....don't. They can already do that via the home menu!

Parameters

- uint8_t volume: volume to set to, 0-8

Returns

none

Example

N/A

Reference,DangerZone.

gb.sound.getVolume

Description

```
uint8_t gb.sound.getVolume()
```

gb.sound.getVolume returns the current volume set.

If you are using this to have the user being able to set the output volume....don't. They can already do that via the home menu!

Parameters

none

Returns

uint8_t: current volume set

Example

N/A

Reference,DangerZone.

gb.language.setCurrentLang

Description

```
void gb.language.setCurrentLang( LangCode code )
```

gb.language.setCurrentLang sets the current language.

If you are wanting to add to your game that the player can change the language....don't. They can already do so via loader.bin

This is used internally for the home menu. I cannot think of any other use for this. So if you are using this for your game you are maybe doing something wrong....

Parameters

- LangCode code: the language code of the langauge to set to

Returns

none

Example

N/A

Reference,DangerZone.

gb.language.getCurrentLang

Description

```
LangCode gb.language.getCurrentLang()
```

gb.language.getCurrentLang fetchs the currently set language code.

If you are wanting to add to your game that the player can change the language....don't. They can already do so via loader.bin

This is used internally for the home menu. I cannot think of any other use for this. So if you are using this for your game you are maybe doing something wrong....

Parameters

none

Returns

LangCode: current language code

Example

N/A

Reference,DangerZone.

gb.language._get

Description

```
const char* gb.language._get( const MultiLang* lang )
```

gb.language._get fetches the pointer off a MultiLang array "lang" and returns the correct one, according to the current langauge set. The length of the array is autodetermined, if not possible you will have to use gb.language.get .

As opposed to gb.language.get this isn't effected by config-gamebuino.h "LANGUAGE_DEFAULT_SIZE"

You will typically not need to use this at all. If you are using this you are probably doing something wrong. This exists for internal library stuff.

Parameters

- const MultiLang* lang: multilang array

Returns

const char*: pointer to the correct string

Example

N/A

Reference,DangerZone.

gb.bootloader.version

Description

```
uint32_t gb.bootloader.version()
```

gb.bootloader.version returns the version of the bootloader. The uppermost 16-bit are major, the next 8 bit are minor and the last 8 bit are patch.

For example: bootloader 1.0.1 is 0x00010001

Parameters

none

Returns

uint32_t: version

Example

N/A

Reference,DangerZone.

gb.bootloader.game

Description

```
void gb.bootloader.game([const] char* filename)
```

gb.bootloader.game will flash a filename and load it.

This will load the file, regardless if it is a .bin or not!!!!

This function does not return

Parameters

none

Returns

none

Example

N/A

Reference,DangerZone.

gb.bootloader.loader

Description

```
void gb.bootloader.loader()
```

gb.bootloader.loader loads the loader.

There are only very rare occasions to use this, you'll probably want to use gb.changeGame instead!

This function does not return!

Parameters

none

Returns

none

Example

N/A

Reference,DangerZone.

gb.bootloader.enter

Description

```
void gb.bootloader.enter()
```

gb.bootloader.enter will enter bootloader mode.

This function does not return.

Parameters

none

Returns

none

Example

N/A

Reference,DangerZone.

gb.bootloader.enter

Description

```
void gb.bootloader.enter()
```

gb.bootloader.enter will enter bootloader mode.

This function does not return.

Parameters

none

Returns

none

Example

N/A

Reference,DangerZone.

gb.tft.initB

Description

```
void gb.tft.initB()
```

gb.tft.initB is the init process for ST7735B TFTs.

Parameters

none

Returns

none

Example

N/A

Reference,DangerZone.

gb.tft.initR

Description

```
void gb.tft.initR(uint8_t options = INITR_GREENTAB)
```

gb.tft.initR initializes ST7735R TFTs.

Parameters

- uint8_t options: INITR_GREENTAB, INITR_144GREENTAB, INITR_BLACKTAB

Returns

none

Example

N/A

Reference,DangerZone.

gb.tft.setAddrWindow

Description

```
void gb.tft.setAddrWindow(uint8_t x0 , uint8_t y0 , uint8_t x1 , uint8_t y1)
```

gb.tft.setAddrWindow sets the current window where sent data will be drawn to.

Parameters

- uint8_t x0: from this x-address
- uint8_t y0: from this y-address
- uint8_t x1: to this x-address
- uint8_t y1: to this y-address

Returns

none

Example

N/A

Reference,DangerZone.

gb.tft.pushColor

Description

```
void gb.tft.pushColor(uint16_t c)
```

gb.tft.pushColor pushes two bytes, 16-bit, typically one color, to the screen (and thus typically one pixel).

Parameters

- uint16_t c: 16-bits to push (typically a color)

Returns

none

Example

N/A

Reference,DangerZone.

gb.tft.drawBuffer

Description

```
void gb.tft.drawBuffer(int16_t x , int16_t y , uint16_t* buffer , uint16_t w , uint16_t h)
```

gb.tft.drawBuffer draws a buffer at (x, y) with a width w and height h to the tft, using DMA to speed up the process.

Parameters

- int16_t x: x-coordinate of the window
- int16_t y: y-coordinate of the window
- uint16_t* buffer: buffer to send
- uint16_t w: width of the window
- uint16_t h: height of the window

Returns

none

Example

N/A

Reference,DangerZone.

gb.tft.sendBuffer

Description

```
void gb.tft.sendBuffer(uint16_t* buffer , uint16_t n)
```

gb.tft.sendBuffer starts a DMA transaction to send a buffer of the size n. You have to check *extern bool* Gamebuino_Meta::transfer_is_done to check if the transfer is done.

Parameters

- uint16_t* buffer: buffer to send
- uint16_t n: size of the word-buffer (Important! NOT bytes!)

Returns

none

Example

N/A

Reference,DangerZone.

gb.tft.dataMode

Description

```
void gb.tft.dataMode()
```

gb.tft.dataMode puts the TFT into data mode and selects it on the SPI bus

Parameters

none

Returns

none

Example

N/A

Reference,DangerZone.

gb.tft.commandMode

Description

```
void gb.tft.commandMode()
```

gb.tft.commandMode puts the TFT into command mode and selects it on the SPI bus

Parameters

none

Returns

none

Example

N/A

Reference,DangerZone.

gb.tft.idleMode

Description

```
void gb.tft.idleMode()
```

gb.tft.idleMode de-selects the TFT on the SPI bus

Parameters

none

Returns

none

Example

N/A

Reference,DangerZone.

gb.tft.setRotation

Description

```
void gb.tft.setRotation(uint8_t r)
```

gb.tft.setRotation sets the rotation the TFT is installed in.

Parameters

- uint8_t r: rotation direction (IDK which is which, look in the DS)

Returns

none

Example

N/A

Reference,DangerZone.

gb.tft.invertDisplay

Description

```
void gb.tft.invertDisplay(bool i)
```

gb.tft.invertDisplay will invert the colors of the display / set it to normal

Parameters

- bool i: invert? true - yes

Returns

none

Example

N/A

Graphics.

Introduction

Graphic functions can be used on the display, lights or images.

For example Graphics::fill() fills with one color. You can do the following:

- gb.display.fill() to fill the display with one color.
- gb.lights.fill() to set all the lights to the same color.
- myImage.fill() to fill your Image with one color.

You can use graphic functions directly to the screen and bypass the gb.display buffer by using gb.tft (not advised!).

Entries with an asterix (*) are not available for gb.tft.

Related.

Related.

gb.createColor

Description

```
Color gb.createColor(uint8_t red , uint8_t green , uint8_t blue)
```

gb.createColor creates a Color object value based on common RGB565 values.

Parameters

- uint8_t red: proportion of red (min: 0, max: 255)
- uint8_t green: proportion of green (min: 0, max: 255)
- uint8_t blue: proportion of blue (min: 0, max: 255)

Returns

Color: the RGB565 color which is closest to the one provided

Example

```
#include <Gamebuino-Meta.h>

void setup() {
  gb.begin();
}

void loop() {
  while(!gb.update());
  gb.display.clear();

  // create the color (#FFC526, some shade of orange)
  Color c = gb.createColor(255, 197, 38);
  gb.display.setColor(c);
  gb.display.println("Hello World");
}
```

Related.

Image tutorial

General Idea

The general Idea of images is to have one standard way of drawing graphics onto the screen, on the lights, or on other images.

An image can have multiple different resources (such as, SD card BMP file, flash array, or ram array). An Image can also have either the color mode RGB565 or be indexed.

On an Image itself you can draw lines, shape, text, and more importantly: other images! The `gb.display` itself is an image, meaning you can do all those actions with `gb.display`

Now, let's just pretend we'd already have an image...

```
#include <Gamebuino-Meta.h>

Image myImg; // let's just pretend this was a correctly initialized image (which it is not)

void setup() {
gb.begin();
}

void loop() {
  while(!gb.update());
  gb.display.clear();
  // now we can draw the image to the display, at x=10 and y=0
  gb.display.drawImage(10, 0, myImg);
}
```

As you can see, drawing an image is pretty straight-forward.

Image Sources

As already mentioned, there are different kind of image sources, which result in different constructors for the Image object.

RAM Image

Image(uint16_t width , uint16_t height [, ColorMode col = (RGB565|Index)] , [uint16_t frames = 1 [, uint8_t fl = 1]])

A RAM image is simply an image with a blank buffer in RAM. They are only two parameters needed to initialize it: the width and the height. Both are of type `uint16_t`.

The optional other parameters are explained below.

For example, gb.display is such a RAM image, and is, by default, initialized with `Image(80, 64)`

Flash Image

Now, if you want to save some assets, like sprites, in your game, you typically don't want to have them in RAM, so, you can save Images in Flash, too!

You don't have to code the images yourself, use the Image to code converter!

They are either a `uint8_t` array, or a `uint16_t` array. The first one is recommended to use for an indexed image, the second one for an rgb565 image. In the indexed case there is two pixels per byte. In the rgb565 case one pixel per two bytes, so one pixel per array entry.

Both have a header, which are slightly different:

For uint8_t data

```
<width>, <height>,
<frames-lower-8-bit>, <frames-upper-8-bit>,
<frame_loop>,
<transparent_color>,
<color_mode>,
```

For uint16_t data:

```
<width>, <height>,
<frames>,
<frame_loop>,
<transparent_color>,
<color_mode>,
```

For an rgb565 image the color mode is `0`, for an indexed image it is `1`.

The transparent color is which color should be interpreted as transparent. In rgb565 mode `0` is interpreted as no transparency, in indexed mode anything greater than `0x0F` is interpreted as no transparency. This means that, instead of like `0xAA` as transparent color you should just use `0x0A`, else it won't work.

Frames and Frame Loop are explained below.

So, for an indexed checkered-board you'd do

```
#include <Gamebuino-Meta.h>
const uint8_t myImgBuf[] = {
  8, 8, //width, heigth
  1, 0, //frames
  0, //frame loop
  0xFF, //transparent color
  1, //color mode

  0x07, 0x07, 0x07, 0x07,
  0x70, 0x70, 0x70, 0x70,
  0x07, 0x07, 0x07, 0x07,
```

```
  0x70, 0x70, 0x70, 0x70,
  0x07, 0x07, 0x07, 0x07,
  0x70, 0x70, 0x70, 0x70,
  0x07, 0x07, 0x07, 0x07,
  0x70, 0x70, 0x70, 0x70,
};

Image myImg(myImgBuf);

void setup() {
  gb.begin();
}

void loop() {
  while(!gb.update());
  gb.display.clear(RED);

  // draw the image
  gb.display.drawImage(10, 10, myImg);
}
```

Or for a small RGB565 image you'd do

```
#include <Gamebuino-Meta.h>

const uint16_t myImgBuf[] = {
  2, 2, //width, height
  1, //frames
  0, //frame loop
  0, //transparent color
  0, //color mode

  0x1234, 0x2345,
  0x3456, 0x4567,
};

Image myImg(myImgBuf);

void setup() {
  gb.begin();
}

void loop() {
  while(!gb.update());
  gb.display.clear();

  // draw the image
  gb.display.drawImage(10, 10, myImg);
}
```

SD Card Image

Lastly, you can also load assets right from the SD card, from BMP files! Please keep in mind that every SD-card image has to keep track of an internal RAM buffer, and thus your RAM can fill up quickly if using too many at once.

Supported are BMP files with 4-bit, 24-bit and 32-bit bitdepth.

4-bit-depth BMPs are converted to indexed images, according how the current palette of the gamebuino meta is. If exact matches aren't present, closest matches are being used.

24-bit-depth BMPs are converted to rgb565 images.

32-bit-depth BMPs are also converted to rgb565 images. The alpha channel is partly ignored: If a pixel is more than 50% transparent it will appear as a transparent pixel in the final image, else as a solid pixel.

You can create a BMP image like this:

Image myImg("PATH/TO/IMAGE.BMP");

It will automatically detect what kind of BMP is present and adapt accordingly

Image Animations

Now that we discussed different image sources, let's talk about some other features these images have: Animations!

The idea is basically that you have, in your flash/RAM buffer, just multiple frames stacked. Or, with BMP image, have frames stacked vertically.

As explained above, you have different ways to specify the number of frames you have.

Frame Loop hereby is the animation speed, we'll just use 1 for now.

```
#include <Gamebuino-Meta.h>

const uint8_t myAnimBuf[] = {
  8, 8, //width, height
  0x2A, 0x00, //let's just say the animation has 42 frames
  1, //frame loop
  0xFF, //no transparency
  1, //indexed image

  /* all the frames go here */
};

Image myAnim(myAnimBuf);

void setup() {
  gb.begin();
}

void loop() {
  while(!gb.update());
  gb.display.clear();

  // let's just draw the image
  gb.display.drawImage(0, 0, myAnim);
}
```

And, there you go, it automatically progresses to the next frame and loops the animation! As simple as that!

Now, let's say the animation is a tad to fast to you, and you only want to progress one animation frame every two gamebuino frames...that is what the `fl` (Frame Loop) parameter is there for! In the buffer you just set a different Frame Loop parameter, e.g. a value of 2 would only progress the animation every two frames.

Example

Let's say you have this image (scaled up x4 for this tutorial for easier visibility):

As you can see, that will be a four-frame animation, we will be picking that pink background as transparent background. Using this tool we can convert our image to code (setting Number of frames to 4):

```
const uint16_t PlayerIdlingData[] = {16,16,4, 1, 0, 0, 0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0x1063,0xf81f,0xf81f,0xf81f,
0xf81f,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0x1063,0x1063,0xf81f,0xf81f,0x1063,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x4a49,0x1063,0x1063,0x1063,0x4a49,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x1063,0x6d45,0x1063,0x1063,0x6d45,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0x1063,0xf81f,0xf81f,
0xd229,0x1063,0x1063,0x1063,0x1063,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0x1063,0xf81f,0x1063,0xd229,0xd229,0xdeab,0xd229,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0x1063,0xf81f,
0x4a49,0x4a49,0x1063,0x1063,0x1063,0x4a49,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x4a49,0x4a49,0x1063,0x1063,0x1063,0x4a49,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x1063,0x1063,0x1063,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0x1063,0xf81f,0xf81f,0xf81f,0x1063,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
```

```
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0x1063,0xf81f,0xf81f,
0xf81f,0xf81f,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0x1063,0x1063,0xf81f,0xf81f,0x1063,0x1063,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x4a49,0x1063,0x1063,0x1063,0x4a49,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x1063,0x6d45,0x1063,0x1063,0x6d45,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0x1063,0xf81f,
0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0x1063,0xf81f,
0xd229,0x1063,0x1063,0x1063,0x1063,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0x1063,0x4a49,0x4a49,0xd229,0xdeab,
0xd229,0x4a49,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x4a49,0x4a49,0x1063,0x1063,0x1063,0x4a49,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x1063,0x1063,0x1063,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0x1063,0xf81f,0xf81f,0xf81f,0x1063,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0x1063,0xf81f,0xf81f,
0xf81f,0xf81f,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0x1063,0x1063,0xf81f,0xf81f,0x1063,0x1063,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x4a49,0x1063,0x1063,0x1063,0x4a49,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x1063,0x6d45,0x1063,0x1063,0x6d45,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0x1063,0xf81f,0xf81f,
0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0x1063,0xf81f,0xf81f,
0xd229,0x1063,0x1063,0x1063,0x1063,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0x1063,0xf81f,0x1063,0xd229,0xd229,0xdeab,0xd229,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x4a49,0x4a49,0x1063,0x1063,0x1063,0x4a49,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x4a49,0x4a49,0x1063,0x1063,0x1063,0x4a49,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0x1063,0xf81f,0xf81f,0xf81f,0x1063,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
```

```
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,0x1063,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0x1063,0x1063,0xf81f,
0xf81f,0x1063,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0x1063,0x4a49,0x1063,0x1063,0x1063,0x4a49,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x1063,0x6d45,0x1063,0x1063,0x6d45,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xd229,0x1063,0x1063,0x1063,0x1063,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0x1063,0xf81f,0xf81f,0xf81f,0x1063,0xd229,0xd229,0xdeab,0xd229,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0x1063,0xf81f,0xf81f,
0x1063,0x1063,0x1063,0x1063,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,
0x1063,0x1063,0x4a49,0x4a49,0x1063,0x1063,0x1063,0x4a49,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x4a49,0x4a49,0x1063,0x1063,0x1063,0x4a49,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0x1063,0xf81f,0xf81f,0xf81f,0x1063,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f};

Image PlayerIdling(PlayerIdlingData);
```

Now, at the point of writing this tutorial, that tool doesn't support a transparent color yet. The background color is `0xf81f` so, we need to change the first `0` in there so that it becomes:

```
const uint16_t PlayerIdlingData[] = {16,16,4, 1, 0xf81f, 0, 0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0x1063,0xf81f,0xf81f,
0xf81f,0xf81f,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0x1063,0x1063,0xf81f,0xf81f,0x1063,0x1063,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x4a49,0x1063,0x1063,0x1063,0x4a49,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x1063,0x6d45,0x1063,0x1063,0x6d45,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0x1063,0xf81f,0xf81f,
0xd229,0x1063,0x1063,0x1063,0x1063,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0x1063,0xf81f,0x1063,0xd229,0xd229,0xdeab,0xd229,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0x1063,0xf81f,
0x4a49,0x4a49,0x1063,0x1063,0x1063,0x4a49,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,
```

```
0x1063,0x4a49,0x4a49,0x1063,0x1063,0x1063,0x4a49,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x1063,0x1063,0x1063,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0x1063,0xf81f,0xf81f,0xf81f,0x1063,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0x1063,0xf81f,0xf81f,
0xf81f,0xf81f,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0x1063,0x1063,0xf81f,0xf81f,0x1063,0x1063,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x4a49,0x1063,0x1063,0x1063,0x4a49,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x1063,0x6d45,0x1063,0x1063,0x6d45,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0x1063,0xf81f,
0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0x1063,0xf81f,
0xd229,0x1063,0x1063,0x1063,0x1063,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0x1063,0x4a49,0x4a49,0xd229,0xdeab,
0xd229,0x4a49,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x4a49,0x4a49,0x1063,0x1063,0x1063,0x4a49,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x1063,0x1063,0x1063,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0x1063,0xf81f,0xf81f,0xf81f,0x1063,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0x1063,0xf81f,0xf81f,
0xf81f,0xf81f,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0x1063,0x1063,0xf81f,0xf81f,0x1063,0x1063,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x4a49,0x1063,0x1063,0x1063,0x4a49,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x1063,0x6d45,0x1063,0x1063,0x6d45,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0x1063,0xf81f,0xf81f,
0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0x1063,0xf81f,0xf81f,
0xd229,0x1063,0x1063,0x1063,0x1063,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0x1063,0xf81f,0x1063,0xd229,0xd229,0xdeab,0xd229,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x4a49,0x4a49,0x1063,0x1063,0x1063,0x4a49,0xf81f,0xf81f,0xf81f,0xf81f,
```

```
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x4a49,0x4a49,0x1063,0x1063,0x1063,0x4a49,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0x1063,0xf81f,0xf81f,0xf81f,0x1063,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,0x1063,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0x1063,0x1063,0xf81f,
0xf81f,0x1063,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0x1063,0x4a49,0x1063,0x1063,0x1063,0x4a49,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x1063,0x6d45,0x1063,0x1063,0x6d45,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xd229,0x1063,0x1063,0x1063,0x1063,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0x1063,0xf81f,0xf81f,0xf81f,0x1063,0xd229,0xd229,0xdeab,0xd229,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0x1063,0xf81f,0xf81f,
0x1063,0x1063,0x1063,0x1063,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,
0x1063,0x1063,0x4a49,0x4a49,0x1063,0x1063,0x1063,0x4a49,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x4a49,0x4a49,0x1063,0x1063,0x1063,0x4a49,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0x1063,0xf81f,0xf81f,0xf81f,0x1063,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f};

Image PlayerIdling(PlayerIdlingData);
```

And now we can just draw the image and it'll be animated! Here is the full example sketch:

```
#include <Gamebuino-Meta.h>


const uint16_t PlayerIdlingData[] = {16,16,4, 1, 0xf81f, 0, 0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0x1063,0xf81f,0xf81f,
0xf81f,0xf81f,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0x1063,0x1063,0xf81f,0xf81f,0x1063,0x1063,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x4a49,0x1063,0x1063,0x1063,0x4a49,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x1063,0x6d45,0x1063,0x1063,0x6d45,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,
```

```
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0x1063,0xf81f,0xf81f,
0xd229,0x1063,0x1063,0x1063,0x1063,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0x1063,0xf81f,0x1063,0xd229,0xd229,0xdeab,0xd229,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0x1063,0xf81f,
0x4a49,0x4a49,0x1063,0x1063,0x1063,0x4a49,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x4a49,0x4a49,0x1063,0x1063,0x1063,0x4a49,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x1063,0x1063,0x1063,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0x1063,0xf81f,0xf81f,0xf81f,0x1063,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0x1063,0xf81f,0xf81f,
0xf81f,0xf81f,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0x1063,0x1063,0xf81f,0xf81f,0x1063,0x1063,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x4a49,0x1063,0x1063,0x1063,0x4a49,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x1063,0x6d45,0x1063,0x1063,0x6d45,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0x1063,0xf81f,
0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0x1063,0xf81f,
0xd229,0x1063,0x1063,0x1063,0x1063,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0x1063,0x4a49,0x4a49,0xd229,0xdeab,
0xd229,0x4a49,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x4a49,0x4a49,0x1063,0x1063,0x1063,0x4a49,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x1063,0x1063,0x1063,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0x1063,0xf81f,0xf81f,0xf81f,0x1063,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0x1063,0xf81f,0xf81f,
0xf81f,0xf81f,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0x1063,0x1063,0xf81f,0xf81f,0x1063,0x1063,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x4a49,0x1063,0x1063,0x1063,0x4a49,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
```

```
0x1063,0x1063,0x6d45,0x1063,0x1063,0x6d45,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0x1063,0xf81f,0xf81f,
0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0x1063,0xf81f,0xf81f,
0xd229,0x1063,0x1063,0x1063,0x1063,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0x1063,0xf81f,0x1063,0xd229,0xd229,0xdeab,0xd229,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x4a49,0x4a49,0x1063,0x1063,0x1063,0x4a49,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x4a49,0x4a49,0x1063,0x1063,0x1063,0x4a49,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0x1063,0xf81f,0xf81f,0xf81f,0x1063,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,0x1063,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0x1063,0x1063,0xf81f,
0xf81f,0x1063,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0x1063,0x4a49,0x1063,0x1063,0x1063,0x4a49,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x1063,0x6d45,0x1063,0x1063,0x6d45,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xd229,0x1063,0x1063,0x1063,0x1063,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0x1063,0xf81f,0xf81f,0xf81f,0x1063,0xd229,0xd229,0xdeab,0xd229,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0x1063,0xf81f,0xf81f,
0x1063,0x1063,0x1063,0x1063,0x1063,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,
0x1063,0x1063,0x4a49,0x4a49,0x1063,0x1063,0x1063,0x4a49,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0x4a49,0x4a49,0x1063,0x1063,0x1063,0x4a49,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0x1063,0xf81f,0xf81f,0xf81f,0x1063,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,0xf81f,
0xf81f,0xf81f,0xf81f,0xf81f};
Image PlayerIdling(PlayerIdlingData);




void setup() {
  gb.begin();
}

void loop() {
```

```
  while(!gb.update());
  gb.display.clear(LIGHTBLUE);
  gb.display.drawImage(5, 5, PlayerIdling);
}
```

Please note, while that does work with BMP spritesheets, that `setFrame` is slow with BMP images, and thus it is recommended to use a flash array.

Image Cropping

You can also directly crop an image while drawing it, for that you just need to specify the `x`, `y`, `width` and `height` of the cropped image. Example, assuming that myImage is an image:

```
gb.display.drawImage(0, 0, myImage, 4, 4, 8, 8);
```

Indexed Images

The default color Index is defined by the following table:

Index	Hex	Name
0	0x00	BLACK
1	0x01	DARK BLUE
2	0x02	PURPLE
3	0x03	GREEN
4	0x04	BROWN
5	0x05	DARK GRAY
6	0x06	GRAY
7	0x07	WHITE
8	0x08	RED
9	0x09	ORANGE
10	0x0A	YELLOW
11	0x0B	LIGHT GREEN
12	0x0C	LIGHT BLUE
13	0x0D	BLUE
14	0x0E	PINK
15	0x0F	BEIGE

You can change the palette by making your own array of the `Color[]` type. To do that, you use `gb.display.setPalette`:

```
// this is obviously a pretty trashy palette, but enough to show how things are working
Color myPalette[16] = {
  (Color)0x0000,
  (Color)0x0001,
  (Color)0x0002,
  (Color)0x0003,
  (Color)0x0004,
  (Color)0x0005,
  (Color)0x0006,
  (Color)0x0007,
  (Color)0x0008,
```

```
  (Color)0x0009,
  (Color)0x000A,
  (Color)0x000B,
  (Color)0x000C,
  (Color)0x000D,
  (Color)0x000E,
  (Color)0x000F,
};

// set the palette
gb.display.setPalette(myPalette);

// if the palette is in RAM you can, after setting, just modify one of the
entries
myPalette[5] = (Color)0xFFFF;

// draw stuff

// reset back to default palette (only if you want to)
gb.display.setPalette(Gamebuino_Meta::defaultColorPalette);
```

Now, the question is, when is the color palette been used? The answer is: when drawing an indexed image to an rgb565 image, or when sending an indexed image to the screen.

This means, if your gb.display buffer is rgb565 (which is default), you could just switch the palette between drawing sprites and you will have a more-than-16-color image to draw on the screen.

This does, however, mean that if gb.display is an indexed image, while drawing sprites the color index is completely ignored, the indexes are just directly mapped onto each other. You can still swap the palette before sending to the screen, though. With cycling colors you can achieve some old-school tricks like the flowing water effects.

Graphics, general.

Graphics, general.

Graphics::drawImage

Description

```
void Graphics::drawImage(
     int16_t x ,
     int16_t y ,
     Image img [,int16_t w2, int16_t h2])
void Graphics::drawImage(
     int16_t x,
     int16_t y,
     Image img, int16_t source_x, int16_t source_y, int16_t source_width,
int16_t source_height)
```

Graphics::drawImage draws an image to the display/light/other image etc. If the image is animated it automatically progresses the animation, based on the set parameters. You can also re-scale or crop the image if you want to.

Keep in mind that you cannot draw full-color images onto indexed images!

Read the Image tutorial.

Parameters

- int16_t x: x-coordinate where to draw the image
- int16_t y: y-coordinate where to draw the image
- Image img: the image to draw
- int16_t w2 (optional): new width (for scaling)
- int16_t h2 (optional): new height (for scaling)

With source parameters:

- int16_t source_x: x-coordinate in source image where to begin drawing
- int16_t source_y: y-coordinate in source image where to begin drawing
- int16_t source_width: width in source image to what to crop to
- int16_t source_height: height in source image to what to crop to

Returns

none

Example

```
#include <Gamebuino-Meta.h>

// create red-yellow checkerboard image
const uint8_t imgBuffer[] = {
  8, 8,
  1, 0,
  0,
  0xFF,
  1,
  0x6A, 0x6A, 0x6A, 0x6A,
  0xA6, 0xA6, 0xA6, 0xA6,
  0x6A, 0x6A, 0x6A, 0x6A,
  0xA6, 0xA6, 0xA6, 0xA6,
  0x6A, 0x6A, 0x6A, 0x6A,
  0xA6, 0xA6, 0xA6, 0xA6,
  0x6A, 0x6A, 0x6A, 0x6A,
  0xA6, 0xA6, 0xA6, 0xA6,
};
Image img(imgBuffer);

void setup() {
  gb.begin();
}

void loop() {
  while(!gb.update());
  gb.display.clear();

 // draw the image onto the screen
  gb.display.drawImage(0, 0, img);

 // and now draw and upscale at the same time
  gb.display.drawImage(8, 0, img, img.width()2, img.height()2);

 // and now only draw the middle 4x4 part of the image
  gb.display.drawImage(16, 0, img, 2, 2, 4, 4);
}
```

Graphics, general.

Graphics::clear

Description

```
void gb.display.clear()
void gb.display.clear(Color c)
void gb.display.clear(ColorIndex c)
void gb.display.clear(uint8_t color)
```

gb.display.clear erases the META screen, as its name suggests. In addition, you can specify the background color of the screen to erase it (default color is BLACK, which is equivalent to 0).

Parameters

- void | Color | ColorIndex | uint8_t color: the background color to set to.

Returns

none

Example

```
#include <Gamebuino-Meta.h>

void setup() {
  // initialize the Gamebuino object
  gb.begin();
}

void loop() {
  while(!gb.update());
  gb.display.clear();
  gb.display.println("Hello world");
}
```

Graphics, general.

Graphics::drawBitmap

This is a legacy function, you should prefer the brand new gb.display.drawImage function.

Description

```
void Graphics::drawBitmap(int8_t x , int8_t y , const uint8_t* bitmap [,
uint8_t rotation , uint8_t flip ])
```

Graphics::drawBitmap draws a monochrome bitmap to the display/image/whatever. Optionally you can also rotate and flip it.

Parameters

- int8_t x: x-coordinate of the bitmap
- int8_t y: y-coordinate of the bitmap
- const uint8_t* bitmap: the bitmap itself
- uint8_t rotation (optional): rotation of the bitmap
- uint8_t flip (optional): in which direction(s) to flip the bitmap

Returns

none

Example

```
#include <Gamebuino-Meta.h>

// define a checkerbox bitmap
const uint8_t bitmap[] = {
  8, 8,
  0b01010101,
  0b10101010,
  0b01010101,
  0b10101010,
  0b01010101,
  0b10101010,
  0b01010101,
  0b10101010,
};

void setup() {
  gb.begin();
}

void loop() {
  while(!gb.update());
  gb.display.clear();

  // draw the checkered bitmap in yellow
  gb.display.setColor(YELLOW);
  gb.display.drawBitmap(0, 0, bitmap);
}
```

Graphics, general.

Graphics::drawPixel

Description

```
void Graphics::drawPixel(int16_t x , int16_t y [, Color | ColorIndex color ])
```

Graphics::drawPixel will draw a pixel to the display/light/image, at x/y coordinates and with a certain color. If no color specified whatever is set via Graphics::setColor is used.

Parameters

- int16_t x: x-coordinate of the pixel to set
- int16_t y: y-coordinate of the pixel to set
- Color | ColorIndex color (optional): color of the pixel to set

Returns

none

Example

```
#include <Gamebuino-Meta.h>

void setup() {
  gb.begin();
}

void loop() {
  while(!gb.update());
  gb.display.clear();
  gb.lights.clear();

  // turn only the top-left pixel of the lights read
  gb.lights.drawPixel(0, 0, RED);

  // set a pixel of the display green
  gb.display.setColor(GREEN);
  gb.display.drawPixel(42, 42);
}
```

Graphics, general.

Graphics::getPixelColor (*)

Description

```
Color Image::getPixelColor(int16_t x , int16_t y)
```

Image::getPixelColor fetches the pixel at (x, y) as color, no matter if the image is indexed or not.

Parameters

- int16_t x: x-coordinate of the pixel
- int16_t y: y-coordinate of the pixel

Returns

Color: color of the pixel

Example

N/A

Graphics, general.

Graphics::getPixelIndex (*)

Description

```
ColorIndex Image::getPixelIndex(int16_t x , int16_t y)
```

Image::getPixelIndex fetches the pixel at (x, y) as an indexed color. If there is no matching indexed color it'll find the closest match.

Parameters

- int16_t x: x-coordinate of the pixel
- int16_t y: y-coordinate of the pixel

Returns

ColorIndex: indexed color (closest match)

Example

N/A

Graphics, general.

Graphics::fill

Description

```
void Graphics::fill([ Color | ColorIndex color ])
```

Graphics::fill will fill the screen/image/lights with the specified color, or, if none given, use whatever was set via Graphics::setColor.

Parameters

- Color | ColorIndex color (optional): color to fill the screen with

Returns

none

Example

```
#include <Gamebuino-Meta.h>

void setup() {
  gb.begin();
}

void loop() {
  while(!gb.update());
  gb.display.clear();
  gb.lights.clear();

  // turn the display red
  gb.display.fill(RED);

  // make all lights green
  gb.lights.fill(GREEN);
}
```

Graphics, general.

Graphics::clear

Description

```
void Graphics::clear([ Color | ColorIndex bgcolor ])
```

Graphics::clear will clear the screen (by default BLACK, custom color can be specified), reset the cursor and the font.

Parameters

- Color|ColorIndex bgcolor (optional): color to fill the screen with

Returns

none

Example

```
#include <Gamebuino-Meta.h>

void setup() {
  gb.begin();
}

void loop() {
  while(!gb.update());
  // clear the screen
  gb.display.clear();

  // clear the lights
  gb.light.clear();

  gb.display.println("Hello World!");
}
```

Graphics, general.

Graphics::clearTextVars

Description

```
void Graphics::clearTextVars()
```

Graphics::clearTextVars will reset the cursor, the font etc. This function is automatically invoked when calling Graphics::clear. It makes sense to use this if you are working with a persistent screen.

Parameters

none

Returns

none

Example

N/A

Graphics, general.

Graphics::setColor

Description

```
void Graphics::setColor(Color | ColorIndex | uint8_t color [, Color | ColorIndex | uint8_t bgColor ])
void Graphics::setColor(uint8_t r , uint8_t g , uint8_t b)
```

Graphics::setColor will set the current drawing color. You can also specify a background color. If none is specified the background color will be transparent. This is used for things like drawing text, setting pixels etc. You can also specify your own RGB values you want to use.

IMPORTANT!!! Colors are *static*, meaning that if you change the color on gb.light it will also change on gb.display etc.!

Parameters

- Color | ColorIndex | uint8_t color: the color to set to
- Color | ColorIndex | uint8_t gColor (optional): the background color to set to.
- Instead of the above: uint8_t r, g, b the RGB values of the color to set to.
- You can use the following default colors. For more info see Color Palettes.

```
WHITE
GRAY
DARKGRAY
BLACK
PURPLE
PINK
RED
ORANGE
```

```
BROWN
BEIGE
YELLOW
LIGHTGREEN
GREEN
DARKBLUE
BLUE
LIGHTBLUE
```

Returns

none

Example

```
#include <Gamebuino-Meta.h>
void setup() {
  gb.begin();
}
void loop() {
  while(!gb.update());
  gb.clear();
  // set the color to draw
  gb.display.setColor(RED);
  gb.display.println("Red text");
}
```

Graphics, general.

Graphics::setTransparentColor

Description

```
void Graphics::setTransparentColor(Color | ColorIndex c)
```

Graphics::setTransparentColor will set the transparent color, so which color will act transparent when using Graphics::drawImage.

By default, Images don't have a transparent color.

The transparent color is set per Image, so you can have different transparent colors set on different images.

After you set a transparent color, you can clear is with clearTransparentColor.

It's advised to use this function in setup().

Parameters

- Color | ColorIndex c: color to be transparent

Returns

none

Example

N/A

Graphics, general.

Graphics::clearTransparentColor

Description

```
void Graphics::clearTransparentColor()
```

Graphics::clearTransparentColor will clear the currently set transparent color, so that no color is transparent.

Parameters

none

Returns

none

Example

N/A

Graphics, general.

Graphics::getBitmapPixel

Description

```
bool Graphics::getBitmapPixel(const uint8_t* bitmap , uint8_t x , uint8_t y)
```

Graphics::getBitmapPixel returns true if a pixel at (x, y) is set in the bitmap, otherwise false.

Parameters

- const uint8_t* bitmap: the bitmap to inspect
- uint8_t x: x-coordinate within the bitmap
- uint8_t y: y-coordinate within the bitmap

Returns

bool: true if pixel is set, false otherwise

Example

N/A

Graphics, general.

Graphics::height

Description

```
int16_t Graphics::height()
```

Graphics::height returns the height of the display / image / light / whatever

Parameters

none

Returns

int16_t: height

Example

```
#include <Gamebuino-Meta.h>

void setup() {
  gb.begin();
}

void loop() {
  while(!gb.update());
  gb.display.clear();

  // print the screen height
  gb.display.print("Screen height:");
  gb.display.println(gb.display.height());

  // print the light height
  gb.display.print("Light height:");
  gb.display.println(gb.lights.height());
}
```

Graphics, general.

Graphics::width

Description

```
int16_t Graphics::width()
```

Graphics::width returns the width of the display / image / light / whatever

Parameters

none

Returns

int16_t: width

Example

```
#include <Gamebuino-Meta.h>

void setup() {
  gb.begin();
}

void loop() {
  while(!gb.update());
  gb.display.clear();

  // print the screen width
  gb.display.print("Screen width:");
  gb.display.println(gb.display.width());

  // print the light width
  gb.display.print("Light width:");
  gb.display.println(gb.light.width());
}
```

Graphics, general.

Graphics::setPalette

Description

```
void Graphics::setPalette([const] Color * p)
```

Graphics::setPalette sets the current color palette in use. Please keep in mind that the palette is expected to have 16 entries.

Parameters

- [const] Color * p: pointer to the palette to use

Returns

none

Example

N/A

Graphics, general.

Graphics::getPalette

Description

```
Color* Graphics::getPalette()
```

Graphics::getPalette returns a pointer to the palette currently in use.

Parameters

none

Returns

Color* : pointer to the palette currently in use

Example

N/A

Text.

Text.

Print::print

Description

```
size_t Print::print(T text [, int base = DEC])
```

Print::print will print stuff to the screen/image/whatever. That stuff can be a string, or a number. If it is a number you can also specify in which base to print it.

Parameters

- T text: stuff to print
- int base (optional): base to print in. Possible values are DEC, HEX, BIN

Returns

size_t: number of characters printed

Example

```
#include <Gamebuino-Meta.h>

void setup() {
  gb.begin();
}

void loop() {
  while(!gb.update());
  gb.display.clear();

  // print "Hello World" to the screen
  gb.display.print("Hello World");

  // print 42 in hexadecimal
  gb.display.print(42, HEX);
}
```

Text.

Print::println

Description

```
size_t Print::println(T text [, int base = DEC])
```

Print::println will print stuff to the screen/image/whatever, followed by a line break. That stuff can be a string, or a number. If it is a number you can also specify in which base to print it.

Parameters

- T text: stuff to print
- int base (optional): base to print in. Possible values are DEC, HEX, BIN

Returns

size_t: number of characters printed

Example

```
#include <Gamebuino-Meta.h>

void setup() {
  gb.begin();
}

void loop() {
  while(!gb.update());
  gb.display.clear();

  // print "Hello World" to the screen
  gb.display.println("Hello World");

  // print 42 in hexadecimal
  gb.display.println(42, HEX);
}
```

Text.

Print::printf

Description

```
void Print::printf(const char[]text, ...)
```

Print::printf prints a formatted string to the screen / display / whatever. You can find information on how printf is working over here. Keep in mind that this version doesn't have floating point support.

Parameters

- const char[] text: text to format

Returns

none

Example

```
#include <Gamebuino-Meta.h>

void setup() {
  gb.begin();
}

void loop() {
  while(!gb.update());
  gb.display.clear();

  // print formatted text
  gb.display.printf("I have %d apples.", 42);
}
```

Text.

Graphics::drawChar

Description

```
void Graphics::drawChar(int16_t x, int16_t y, unsigned char c, uint8_t size)
```

Graphics::drawChar draws a character at (x, y) to the screen with a certain size.

Parameters

- int16_t x: x-position of the character
- int16_t y: y-position of the character
- unsigned char c: character to draw
- uint8_t size: size of the caharacter

Returns

none

Example

N/A

Text.

Graphics::setFontSize

Description

```
void Graphics::setFontSize(uint8_t size)
```

Graphics::setFontSize sets the size of the font being drawn.

Parameters

- uint8_t size: size of the font (1 being smallest, 2 being double etc.)

Returns

none

Example

```
#include <Gamebuino-Meta.h>

void setup() {
  gb.begin();
}

void loop() {
  while(!gb.update());
  gb.display.clear();

  // set the font size to double
  gb.display.setFontSize(2);

  gb.display.println("Hello World");
}
```

Text.

Graphics::setTextWrap

Description

```
void Graphics::setTextWrap(bool wrap)
```

Graphics::setTextWrap will set if the text will automatically wrap around if the end of the screen is reached.

Parameters

- bool wrap: enable wrap? true - yes, false - no

Returns

none

Example

N/A

Text.

Graphics::setFont

Description

```
void Graphics::setFont(const GFXfont* | const uint8_t* font)
```

Graphics::setFont allows you to set a different font. By default the 3x5 pixel font is used.

Parameters

- const GFXfont* | const uint8_t* font: the font to use

Returns

none

Example

N/A

Text.

Graphics::setCursor

Description

```
void Graphics::setCursor(int16_t x , int16_t y)
```

Graphics::setCursor sets both the x-position and the y-position of the text cursors at the same time.

Parameters

- int16_t x: x-position of the text cursors
- int16_t y: y-position of the text cursors

Returns

none

Example

```
#include <Gamebuino-Meta.h>

void setup() {
  gb.begin();
}

void loop() {
  while(!gb.update());
  gb.display.clear();

  // set the text cursors
  gb.display.setCursor(10, 10);

  gb.display.println("Hello World");
}
```

Text.

Graphics::setCursorX

Description

```
void Graphics::setCursorX(int16_t x)
```

Graphics::setCursorX will set the x-position of the text cursor.

Parameters

- int16_t x: x-position of the cursor

Returns

none

Example

```
#include <Gamebuino-Meta.h>

void setup() {
  gb.begin();
}

void loop() {
  while(!gb.update());
  gb.display.clear();

  // set the x-cursor to something else
  gb.display.setCursorX(10);

  gb.display.println("Hello World");
}
```

Text.

Graphics::setCursorY

Description

```
void Graphics::setCursorY(int16_t y)
```

Graphics::setCursorY will set the y-position of the text cursor.

Parameters

- int16_t y: y-position of the cursor

Returns

none

Example

```
#include <Gamebuino-Meta.h>

void setup() {
  gb.begin();
}

void loop() {
  while(!gb.update());
  gb.display.clear();

  // set the y-cursor to something else
  gb.display.setCursorY(10);

  gb.display.println("Hello World");
}
```

Text.

Graphics::getTextBounds

gets minimal rectangle needed for text to render.

Maybe major re-write if we drop the gfxFont stuff since we only use our uint8_t* fonts and thus it would be way smaller&faster?

Text.

Graphics::getCursorX

Description

```
int16_t Graphics::getCursorX()
```

Graphics::getCursorX returns the current x-position of the cursor.

Parameters

none

Returns

int16_t: x-position of the cursor

Example

N/A

Text.

Graphics::getCursorY

Description

```
int16_t Graphics::getCursorY()
```

Graphics::getCursorY returns the current y-position of the text cursor.

Parameters

none

Returns

int16_t: current y-position of the text cursor

Example

N/A

Text.

Graphics::getFontWidth

Description

```
uint8_t Graphics::getFontWidth()
```

Graphics::getFontWidth returns the width of the current font.

Parameters

none

Returns

uint8_t: font width

Example

N/A

Text.

Graphics::getFontHeight

Description

```
uint8_t Graphics::getFontHeight()
```

Graphics::getFontHeight returns the height of the currently loaded font.

Parameters

none

Returns

uint8_t: font height

Example

N/A

Shapes.

Shapes.

Graphics::drawLine

Description

```
void Graphics::drawLine(int16_t x0, int16_t y0, int16_t x1, int16_t y1)
```

Graphics::drawLine will draw a line from (x0, y0) to (x1, y1).

Parameters

- int16_t x0: x0 of the line to draw
- int16_t y0: y0 of the line to draw
- int16_t x1: x1 of the line to draw
- int16_t y1: y1 of the line to draw

Returns

none

Example

```
#include <Gamebuino-Meta.h>

void setup() {
  gb.begin();
}

void loop() {
  while(!gb.update());
  gb.display.clear();

  // let's draw a green diagonal line
  gb.display.setColor(GREEN);
   gb.display.drawLine(0, 0, gb.display.width() - 1, gb.display.height() -
1);
}
```

Shapes.

Graphics::drawFastVLine

Description

```
void Graphics::drawFastVLine(int16_t x, int16_t y, int16_t h)
```

Graphics::drawFastVLine draws a horizontal line, starting at (x, y) and with a certain height.

Parameters

- int16_t x: x-coordinate to start
- int16_t y: y-coordinate to start
- int16_t h: height of the vertical line

Returns

none

Example

```
#include <Gamebuino-Meta.h>

void setup() {
  gb.begin();
}

void loop() {
  while(!gb.update());
  gb.display.clear();
  gb.light.clear();

  // draw a border on the left of the screen in green
  gb.display.setColor(GREEN);
  gb.display.drawFastVLine(0, 0, gb.display.height());

  // make all the LEDs on the right red
  gb.light.setColor(RED);
  gb.light.drawFastVLine(1, 0, gb.light.height());
}
```

Shapes.

Graphics::drawFastHLine

Description

```
void Graphics::drawFastHLine(int16_t x, int16_t y, int16_t w)
```

Graphics::drawFastHLine draws a horizontal line with a certain width starting at (x, y)

Parameters

- int16_t x: x-coordinate to start
- int16_t y: y-coordinate to start
- int16_t w: width of the horizontal line

Returns

none

Example

```
#include <Gamebuino-Meta.h>

void setup() {
  gb.begin();
}

void loop() {
  while(!gb.update());
  gb.display.clear();

  // make the first row of pixels on the display yellow
  gb.display.setColor(YELLOW);
  gb.display.drawFastHLine(0, 0, gb.display.width());
}
```

Shapes.

Graphics::drawRect

Description

```
void Graphics::drawRect(int16_t x, int16_t y, int16_t w, int16_t h)
```

Graphics::drawRect draws a rectangle (outline only!) from (x, y) with a certain width and height.

Parameters

- int16_t x: x-coordinate of the rectangle
- int16_t y: y-coordinate of the rectangle
- int16_t w: width of the rectangle
- int16_t h: height of the rectangle

Returns

none

Example

```
#include <Gamebuino-Meta.h>

void setup() {
  gb.begin();
}

void loop() {
  while(!gb.update());
  gb.display.clear();

  // draw a yellow frame around the entire screen
  gb.display.setColor(YELLOW);
  gb.display.drawRect(0, 0, gb.display.width(), gb.display.height());
}
```

Shapes.

Graphics::fillRect

Description

```
void Graphics::fillRect(int16_t x, int16_t y, int16_t w, int16_t h)
```

Graphics::fillRect draws a filled rectangle to the screen/image/whatever, starting from (x, y) and with a certain width and height.

Parameters

- int16_t x: x-coordinate of the rectangle
- int16_t y: y-coordinate of the rectangle
- int16_t w: width of the rectangle
- int16_t h: height of the rectangle

Returns

none

Example

```
#include <Gamebuino-Meta.h>

void setup() {
  gb.begin();
}

void loop() {
  while(!gb.update());
  gb.display.clear();

  // draw a green rectangle
  gb.display.setColor(GREEN);
  gb.display.fillRect(10, 10, 20, 5);
}
```

Shapes.

Graphics::drawCircle

Description

```
void Graphics::drawCircle(int16_t x, int16_t y, int16_t r)
```

Graphics::drawCircle draws a circle (line only!) around (x, y) with a radius of r.

Parameters

- int16_t x: x-coordinate of the center
- int16_t y: y-coordinate of the center
- int16_t r: radius

Returns

none

Example

```
#include <Gamebuino-Meta.h>

void setup() {
  gb.begin();
}

void loop() {
  while(!gb.update());
  gb.display.clear();

  // draw a red circle
  gb.display.setColor(RED);
  gb.display.drawCircle(15, 15, 10);
}
```

Shapes.

Graphics::fillCircle

Description

```
void Graphics::fillCircle(int16_t x, int16_t y, int16_t r)
```

Graphics::fillCircle draws a filled circle around (x, y) with a radius of r.

Parameters

- int16_t x: x-coordinate of the center
- int16_t y: y-coordinate of the center
- int16_t r: radius

Returns

none

Example

```
#include <Gamebuino-Meta.h>

void setup() {
  gb.begin();
}

void loop() {
  while(!gb.update());
  gb.display.clear();

  // draw a red circle
  gb.display.setColor(RED);
  gb.display.fillCircle(15, 15, 10);
}
```

Shapes.

Graphics::drawTriangle

Description

```
void Graphics::drawTriangle(int16_t x0, int16_t y0, int16_t x1, int16_t y1,
int16_t x2, int16_t y2)
```

Graphics::drawTriangle draws a triangle (line only) between three points, (x0, y0), (x1, y1) and (x2, y2).

Parameters

- int16_t x0: x-coordinate of 0th point
- int16_t y0: y-coordinate of 0th point
- int16_t x1: x-coordinate of 1st point
- int16_t y1: y-coordinate of 1st point
- int16_t x2: x-coordinate of 2nd point
- int16_t y2: y-coordinate of 2nd point

Returns

none

Example

```
#include <Gamebuino-Meta.h>

void setup() {
  gb.begin();
}

void loop() {
  while(!gb.update());
  gb.display.clear();

  // draw a green triangle
  gb.display.setColor(GREEN);
  gb.display.drawTriangle(0, 10, 10, 0, 20, 10);
}
```

Shapes.

Graphics::fillTriangle

Description

```
void Graphics::fillTriangle(int16_t x0, int16_t y0, int16_t x1, int16_t y1,
int16_t x2, int16_t y2)
```

Graphics::fillTriangle draws a filled triangle between three points, (x0, y0), (x1, y1) and (x2, y2).

Parameters

- int16_t x0: x-coordinate of 0th point
- int16_t y0: y-coordinate of 0th point
- int16_t x1: x-coordinate of 1st point
- int16_t y1: y-coordinate of 1st point
- int16_t x2: x-coordinate of 2nd point
- int16_t y2: y-coordinate of 2nd point

Returns

none

Example

```
#include <Gamebuino-Meta.h>

void setup() {
  gb.begin();
}

void loop() {
  while(!gb.update());
  gb.display.clear();

  // draw a green triangle
  gb.display.setColor(GREEN);
  gb.display.fillTriangle(0, 10, 10, 0, 20, 10);
}
```

Shapes.

Graphics::drawRoundRect

Description

```
void Graphics::drawRoundRect(int16_t x, int16_t y, int16_t w, int16_t h,
int16_t r)
```

Graphics::drawRoundRect draws a rectangle at (x, y) with width w and height h whose corners are rounded with the radius r.

Parameters

- int16_t x: x-coordinate of the rectangle
- int16_t y: y-coordinate of the rectangle
- int16_t w: width of the rectangle
- int16_t h: height of the rectangle
- int16_t r: radius of the corners

Returns

none

Example

```
#include <Gamebuino-Meta.h>

void setup() {
  gb.begin();
}

void loop() {
  while(!gb.update());
  gb.display.clear();

  // draw a yellow rectangle with round corners
  gb.display.setColor(YELLOW);
  gb.display.drawRoundRect(5, 5, 42, 20, 5);
}
```

Shapes.

Graphics::fillRoundRect

Description

```
void Graphics::fillRoundRect(int16_t x, int16_t y, int16_t w, int16_t h,
int16_t r)
```

Graphics::fillRoundRect draws a filled rectangle at (x, y) with width w and height h whose corners are rounded with the radius r.

Parameters

- int16_t x: x-coordinate of the rectangle
- int16_t y: y-coordinate of the rectangle
- int16_t w: width of the rectangle
- int16_t h: height of the rectangle
- int16_t r: radius of the corners

Returns

none

Example

```
#include <Gamebuino-Meta.h>

void setup() {
  gb.begin();
}

void loop() {
  while(!gb.update());
  gb.display.clear();

  // draw a yellow rectangle with round corners
  gb.display.setColor(YELLOW);
  gb.display.fillRoundRect(5, 5, 42, 20, 5);
}
```

Frame Handling.

Frame Handling.

Image::setFrame (*)

Description

```
void Image::setFrame(uint16_t frame)
```

Image::setFrame will set the frame of the image to the specified frame, or to the last one if frame is larger than the amount of frames.

Parameters

- uint16_t frame: frame number

Returns

none

Example

N/A

Saving.

Saving.

Image::startRecording (*)

Description

```
bool Image::startRecording(char* filename)
```

Image::startRecording starts to record (animation) the image. The output file needs to be either a BMP file or a GMV file. In case of a BMP file the GMV is auto-converted to BMP upon stopping.

Parameters

- char* filename: filename to record to

Returns

bool: true on success, false on failure

Example

N/A

Saving.

Image::stopRecording (*)

Description

```
void Image::stopRecording([bool output = false])
```

Image::stopRecording stops the recording of a currently recorded image. If output is set to true progress about the BMP conversion is displayed on-screen.

Parameters

- bool output (optional): display progress about conversion?

Returns

none

Example

N/A

Saving.

Image::save (*)

Description

```
bool Image::save(char* filename)
```

Image::save saves the current state of an image to a BMP or a GMV to the sd card.

Parameters

- char* filename: filename to save to

Returns

bool: true on success, false on failure

Example

N/A

Graphics, DangerZone.

Graphics,DangerZone.

Graphics::_drawPixel

Description

```
void Graphics::_drawPixel(int16_t x , int16_t y)
```

Graphics::_drawPixel will draw a pixel with the currently set color via Graphics::setColor. Whichever class inherits from *Graphics* will have to implement this!

Do not call this in your game! If you are using this, you are probably doing something wrong ! You are probably looking for Graphics::drawPixel instead.

Parameters

- int16_t x: x-coordinate of the pixel to draw
- int16_t y: y-coordinate of the pixel to draw

Returns

none

Example

N/A

Graphics,DangerZone.

Image::getPixel (*)

Description

```
uint16_t Image::getPixel(int16_t x , int16_t y)
```

Image::getPixel returns the pixel color at (x, y), however you have to figure out yourself if it is an indexed color or an rgb565 color! It is highly adviced to use Image::getPixelColor or Image::getPixelIndex instead!

Parameters

- int16_t x: x-coordinate of the pixel
- int16_t y: y-coordinate of the pixel

Returns

uint16_t: raw pixel color

Example

N/A

Graphics,DangerZone.

Graphics::drawBufferedLine

Description

```
void Graphics::drawBufferedLine(int16_t x, int16_t y, int16_t* buffer , uint16_t w , Image& img)
```

Graphics::drawBufferedLine will render the buffered line onto the specified graphics object. The class inheriting from *Graphics* will have to implement this. Whoever is calling this has to make sure that the buffer is set correctly (rgb565 vs indexed).

If you are using this in your game you are probably doing something wrong!

Parameters

- int16_t x: x-offset of the buffer to draw
- int16_t y: y-offset of the buffer to draw
- uint16_t* buffer: pointer to the buffer to draw
- uint16_t w: width of the buffer to draw
- Image& img: calling image (can be used for transparency detection)

Returns

none

Example

N/A

Graphics,DangerZone.

Graphics::drawImageCrop

Description

```
void Graphics::drawImageCrop(int16_t x , int16_t y , int16_t w1 , int16_t
i2offset , int16_t w2cropped , int16_t j2offset , int16_t h2cropped , Image
img)
```

Graphics::drawImageCrop performs the actual drawing, with cropping stuff. This function is used internally, please use Graphics::drawImage if you want to draw an image.

Parameters

- int16_t x: x-position where to draw
- int16_t y: y-position where to draw
- int16_t w1: width of destination image
- int16_t i2offset: offset of i-counter
- int16_t w2cropped: cropped width
- int16_t j2offset: offset of j-counter
- int16_t h2cropped: cropped height
- Image img: image to draw

Returns

none

Example

N/A

Graphics,DangerZone.

Graphics::_fill

Description

```
void Graphics::_fill()
```

Graphics::_fill fills the screen with a certain color. This function is used internally. If you are using this, you are probably doing something wrong! You are probably looking for Graphics::fill instead!

Parameters

none

Returns

none

Example

N/A

Graphics,DangerZone.

Graphics::invertDisplay

inverts display, probably obsolete

Graphics,DangerZone.

Graphics::drawCircleHelper

Description

```
void Graphics::drawCircleHelper(int16_t x , int16_t y , int16_t r , uint8_t corner)
```

Graphics::drawCircleHelper is a helper function to draw partial circles. If you are using this function in your game then you are probably doing something wrong!

Parameters

- int16_t x: x-coordinate of circle center
- int16_t y: y-coordinate of circle center
- int16_t r: radius of circle
- uint8_t corner: corner to draw, bitwise (you can draw multiple at once)

Returns

none

Example

N/A

Graphics,DangerZone.

Graphics::fillCircleHelper

Description

```
void Graphics::fillCircleHelper(int16_t x , int16_t y , int16_t r , uint8_t corner , int16_t delta)
```

Graphics::fillCircleHelper is a helper function to draw partial filled circles. If you are using this function in your game then you are probably doing something wrong!

Parameters

- int16_t x: x-coordinate of circle center
- int16_t y: y-coordinate of circle center
- int16_t r: radius of circle
- uint8_t corner: corner to draw, bitwise (you can draw multiple at once)
- int16_t delta: ?

Returns

none

Example

N/A

Graphics,DangerZone.

Graphics::setTmpColor

Description

```
Color setTmpColor(Color | ColorIndex c)
```

Graphics::setTmpColor sets the color c and returns the previously set color, casted to Color no matter if index or not, for easy restore. The function is used internally! If you are using it for your game you are probably doing something wrong!

Parameters

- Color | ColorIndex c: color to set

Returns

Color: previous color, casted for easy restore

Example

N/A

Graphics,DangerZone.

Graphics::setRotation

sets rotation. probably obsolete

Graphics,DangerZone.

Graphics::cp437

cp437 stuff (offset on that one char for compability with a font we don't use anyways)

Graphics,DangerZone.

Graphics::indexTo565

Description

```
void Graphics::indexTo565(uint16_t* dest , uint8_t* src , Color* index , uint16_t length , bool skipFirst)
```

Graphics::indexTo565 converts an indexed buffer to an RGB565 buffer. This function is used internally, if you are using it in your game you are probably doing something wrong!

Parameters

- uint16_t* dest: destination buffer
- uint8_t* src: source buffer
- Color* index: color index to use (note to sorunome: why do you even need this? Don't you have it anyways in the static property?)
- uint16_t length: length in pixels
- bool skipFirst: should we skip the first pixel (and thus nibble) in the src buffer?

Returns

none

Example

N/A

Graphics,DangerZone.

Graphics::rgb565ToIndex

Description

```
ColorIndex Graphics::rgb565ToIndex(Color rgb)
```

Graphics::rgb565ToIndex converts a single rgb color into an indexed color, by closest match.

Parameters

- Color rgb: color to convert

Returns

ColorIndex: closest matching indexed color

Example

N/A

Graphics,DangerZone.

Graphics::write

Description

```
size_t Graphics::write(uint8_t c)
```

Graphics::write writes a single character and updates the text cursors accordingly. This function is used internally, if you are using it you are probably doing something wrong!

Parameters

- uint8_t c: character code of the character to write

Returns

size_t: number of characters written, 1

Example

N/A

Graphics,DangerZone.

Graphics::getRotation

gets rotation of the screen - obsolete and should be removed maybe?

Graphics,DangerZone.

Image::nextFrame (*)

Description

```
void Image::nextFrame()
```

Image::nextFrame will advance the frame of your image, so typically your animation, by one frame. This performs the checks for if the frames needs advancement etc. and is automatically invoked when calling Graphics::drawImage, so there is no need for you to call it in your game manually.

Parameters

none

Returns

none

Example

N/A

Graphics,DangerZone.

Image::getBufferSize (*)

Description

```
uint16_t Image::getBufferSize()
```

Image::getBufferSize returns the size the buffer should have, based on width/height and the image type. This function is used internally, if you are using it in your game you are probably doing something wrong!

Parameters

none

Returns

uint16_t: buffer size in bytes

Example

N/A

Color Palettes.

Color Palettes.

Official default palette

Why is it important to use the default palette?

- We spent lot of time fine tuning this palette to get the perfect blend of colors.
- You can't really go wrong with these colors.
- This will give the Gamebuino games a consistent look.
- They are not garish nor not dull and all different.
- This is a pretty versatile palette, you should be able to do any kind of games with it.
- They are the default colors used in indexed mode.

Note: You don't have to use the HEX/RGB values in your program, see gb.display.setColor

Mockups by Drakker, Adekto & Erico, updated with the new palette

Table of constants

		R8	G8	B8	HEX888	HEX565	HEX
	WHITE	255	255	255	#FFFFFF	0xFFFF	0x07
	GRAY	168	153	135	#A89987	0xACD0	0x06
	DARKGRAY	84	77	67	#544D43	0x5268	0x05
	BLACK	0	0	0	#000000	0x0000	0x00
	PURPLE	150	0	64	#960040	0x9008	0x02
	MAGENTA	207	68	133	#CF4485	0xCA30	0x0E
	RED	219	29	35	#DB1D23	0xD8E4	0x08
	ORANGE	255	168	17	#FFA811	0xFD42	0x09
	BROWN	207	142	68	#CF8E44	0xCC68	0x04
	BEIGE	255	214	144	#FFD690	0xFEB2	0x0F
	YELLOW	245	231	0	#F5E700	0xF720	0x0A
	LIGHTGREEN	133	207	68	#85CF44	0x8668	0x0B
	GREEN	0	139	80	#008B50	0x044A	0x03
	DARKBLUE	0	67	133	#004385	0x0210	0x01
	BLUE	68	133	207	#4485CF	0x4439	0x0D
	LIGHTBLUE	125	187	255	#7DBBFF	0x7DDF	0x0C

All informations about Color Palettes are on the official website.

Arduino Cheat.

Arduino Cheat

Arduino Cheat list[1]

Structure
void **setup**() void **loop**()

Control Structures
if (x<5){} **else**{}
switch (myvar){
    case 1 :
    break;
    case 2:
    break;
    default:
}
for (int i=0; i <= 255; i++){}
while (x<5){}
do{} **while** (x<5);
continue; //Go to next in do/for/while loop
return x; ll Or 'return;' for voids.
goto // considered harmful :-)

Further Syntax
// (single line comment)
/* (multi-line comment) */
#define DOZEN 12 //Not baker's!
#Include <avr/pgmspace.h>

General Operators
= (assignment operator)
+ (addition) - (subtraction)
" (multiplication) / (division)
% (modulo)
== (equal to) != (not equal to)
< (less than) > (greater than)
<= (less than or equal to)
>= (greater than or equal to)
&& (and) H (or) ! (n01)

Pointer Access
& reference operator
* dereference operator

Bitwise Operators
& (bitwise and) I (bitwise or)
^ (bitwise xor) ~ (bitwise not)
<< (bitshift left) >> (bitshift right)

Compound Operators
++ (increment) -- (decrement)
+= (compound addition)
-= (compound subtraction)
'= (compound multiplication)
/= (compound division)
&=: (compound bitwise and)
I= (compound bitwise or)

Constants
HIGH I LOW
INPUTI OUTPUT
true I false
143 // **Decimal** number
0173 // **Octal** number
0**b**11011111 //**Binary**
0**x**7B ll **Hex** number
7**U** // Force unsigned
10**L** // Force long
15**UL** // Force long unsigned
10.0 // Forces floating point
2.4**e**5 // 240000

Data Types
void
boolean (0, 1, false, true)
char (e.g. 'a' -128 to 127)
unslgned char (0 to 255)
byte (0 to 255)
int (-32,768 to 32,767)
unslgned Int (0 to 65535)
word (0 to 655word (010 65535)
long (-2,147,483,648 to 2,147,483,647)
unsigned long (010 4,294,967,295)
float (-3.4028235E+38 to 3.4028235E+38)
double (currently same as float)
sizeof(myint) ll returns 2 bytes

Strings
char S1 [15];
char S2[8]={'a','r','d','u','i','n','o');
char S3[8]={'a','r','d','U','i','n','O','\o');
//Included \0 null termination
char S4[ ] = "arduino";
char $5[8] = "arduino";
char S6[15] = "arduino";

Arrays
int myInts[6];
int myPinsl] = (2, 4, 8, 3. 6};
int mySensVals[6] = {2, 4, -8. 3. 2};

Conversion
char() **byte**()
int() **word**()
long() **float**()

[1] This chapter is based on the ARDUINO CHEAT SHEET. Content for this Cheat Sheet provided by Gavin from Robots and Dinosaurs. For more information visit: htth/arduino.cc/en/FleferencelExtended

Qualifiers
static // persists between calls
volatile // use RAM (nice for ISR)
const // make read-only
PROGMEM // use flash

Digital I/O
pinMode(pin, [INPUT,OUTPUT])
digitalWrite(pin, value)
int **digitalRead**(pin)
//Write High to inputs to use pull-up res

Analog I/O
analogReference([DEFAULT, INTERNAL,EXTERNAL])
int **analogRead**(pin) //Call twice if switching pins from high Z source.
analogWrite(pin, value) // PWM

Advanced I/O
tone(pin, freqhz)
tone(pin, freqhz ,dura1i0n_ms)
noTone(pin)
shiftOut(dataPin, clockPin, [MSBFIRST,LSBFIRST], value)
unsigned long **pulsein**(pin,[HlGH,LOW])

Time
unsigned long millisO // 50 days overflow.
unsigned lorg microso // 70 min overflow
delay(ms)
delayMicroseconds(us)

Math
min(x, y) **max**(x, y) **abs**(x)
constrain(x, minval, maxval)
map(val, fromL, fromH, toL, 10H)
pow(base, exponent) **sqrt**(x)
sin(rad) **cos**(rad) **tan**(rad)

Random Numbers
randomSeed(seed) // Long or int
long **random**(max)
long **random**(min, max)

Bits and Bytes
lowByte()
highByieO
bltRead(x,bitn)
bitWrite(x,bi1n,bit)
bitSet(x,bitn)
bltClear(x,bitn)
bit(bitn) llbitn: 0-LSB 7-MSB

External Interrupts
atlachInterrupt(interrupt, function, [LOW,CHANGE,RISING,FALLING])
detachInterrupt(interrupt)
interrupts()
noInterrupts()

Serial.
begin([300, 1200, 2400, 4800, 9600,14400, 19200, 28800, 38400, 57600,11 5200])
end()
int **available**()
int **read**()
flush()
print()
println()
write()

EEPROM (#include <EEPROM.h>)
byte **read**(intAddr)
write(intAddr,myByle)

Servo (#include <Servo.h>)
attach(pin , [min_uS, max_uS])
write(angle) // 0-180
writeMicroseconds(uS) //1000-2000,1500 is midpoint
read() // 0-180
atlached() //Returns boolean
detach()

SoftwareSerial(RxPin,TxPin)
// #include<**SoftwareSerial**.h>
begin(longSpeed) // up to 9600
char **read**() ll blocks till data
print(myData) or **println**(myData)

Wire (#include <Wire.h>) // For I2C
begin() // Join as master
begin(addr) // Join as slave @ addr
requestFrom(address, count)
beginTransmission(addr) // Step 1
send(mybyte) // Step 2
send(char ' mystring)
send(byte " data, size)
endTransmission() II Step 3
byte **available**() II Num of bytes
byte **receive**() llReturn next byte
onReceive(handler)
onRequest(handler)

Thanks.

To Sorunome for the writing of this reference,

To Maxime Junchat and Stéphane Calderoni for having create this document,

And the whole Gamebuino Academy team for their feedback.