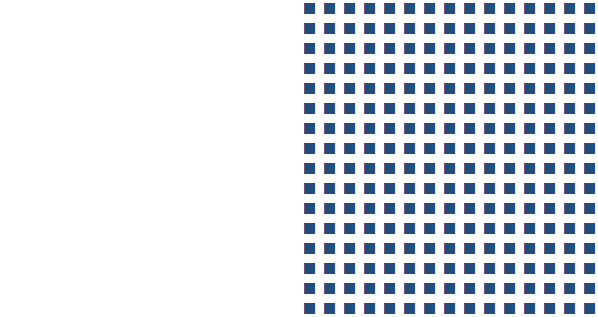
Introduction

What are Sprites?

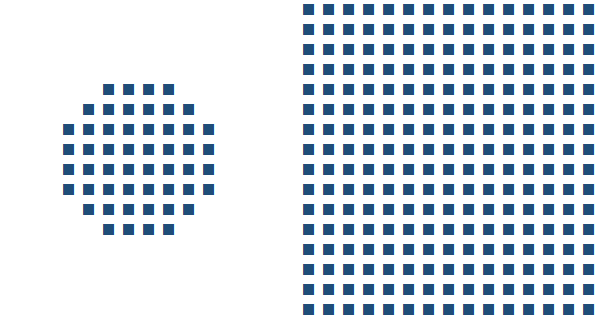
Sprites are a computer graphic which can be rendered and moved on screen as a single unit. In older systems such as the Commodore 64 and Atari, the sprite was rendered by hardware as an overlay to the normal screen image. As the sprite is an overlay, it can be moved around without it affecting the background image.

The Arduboy library has support for sprites but due to the lack of a powerful graphics processor handles them differently. When rendering a sprite, the image is mapped into a single display buffer that may already have a background image drawn on it. If you move the sprite you need to regenerate the background from its old position.

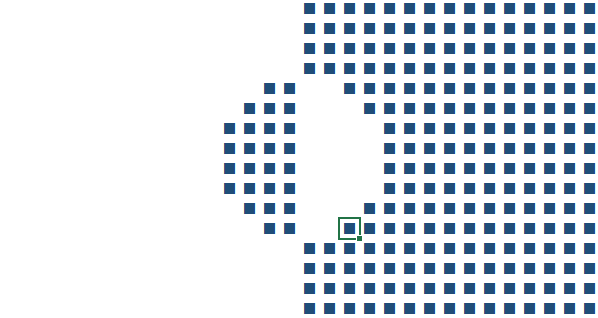
The Arduboy library also provides some nice masking utilities that allow you to render a sprite over a background and have it take that background into account. Imagine you decided to write a game of ‘Pong’ on a half white, half black background like that below:



When you render the ball on the white side of the playing field, you want the ball to be rendered in black. This can be achieved using the command aasdasdas



As the ball moves into the black portion of the screen, the same code will handle the transition.



And finally, as it is completely in the dark side.



Only 10 node types on a 9x9

# Design

Game Concepts

## Artwork

Only 14 node types

What’s an array?

*Arrays* are a data structure that can store a collection of items (bytes, integers or even other objects). Arrays can be single- or multi- dimensional and the items are retrieved using an index. Arrays in an Arduboy environment have a fixed size and cannot be resized.

For example, the following snippet of code defines a one dimensional array of 5 items and populates the 3rd item.

int newArrayA[5];

newArrayA[2] = 23;

Notice that when we want to read the 3rd item from the array we actually use the index of 2. This is because in C / C++ the first position in the array is referred to a position 0. The last item ion the array can be retrieved using the index 4 which is the size of the array, 5, less 1.

Arrays can have multiple dimensions and each dimension is declared with its own size in specified in square parenthesis. For example, the code below defines an array that is 5 columns wide and 3 columns high. It then populates the cell that is in the 4th column of the 2nd row.

int newArrayB[3][5];

newArrayB[1][3] = 666;

Again, notice that the indexes are zero-based. Also notice that the declaration of the array defined the row first, followed by the column. When referring to the array using X and Y coordinates, this produces the counter-intuitive syntax of a[y][x].

Arrays can be populated as part of their declaration. The two declarations above can be extended to include initial values, as shown below:

int newArrayA[5] = { 0, 1, 2, 3, 4 };

int newArrayB[3][5] = {

{ 1, 2, 3, 4, 5 }, // row 0

{ 6, 7, 8, 9, 10 }, // row 1

{ 11, 12, 13, 14, 15 } // row 2

};

Finally, the number of items in the array can be returned using the sizeOf() function. Be warned though that this returns the number of bytes the array uses not the number of items. The sizeOf() function will return a value of 10 for the declaration int newArrayA[5] as each integer takes up two bytes. The number of items in the array can be determined by dividing the size of the array in bytes by the size of the data type, for example:

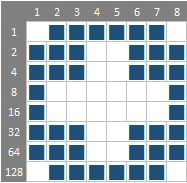
int numberOfItems = sizeof(newArrayA)/sizeof(int)

## Arduboy Sprites

For a small library, the Arduboy2 support for sprites is quit thorough. Looking at the code, it has been optimized for speed and as such imposes one restriction – sprites can be any width but they must be a multiple of 8 pixels high. This may sound like a major restriction but, as I will show later, the library also has some tricks in it to effectively get around this too. In this first installment of the article, I will restrict the sprites we use to simple 8 x 8 and 16 x 16 pixel graphics.

@Gaveno112 posted a great article in Volume 3 of the Arduboy Magazine and I encourage you to read it.

Take the simple sprite below. It is 8 pixels wide by 8 pixels high – an optimal size for a screen whose coordinates are a multiple of 8 (132 x 64 pixels).



The definition for a sprite consists of the sprites dimensions followed by an array (check out the *What’s an array?* Sidebar for more information) of numbers each of which describe a column of 8 pixels each.

The array definition for this sprite is shown below.

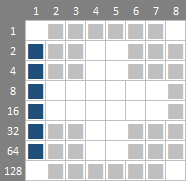
byte myFirstSprite[] = {

8, 8,

126, 231, 231, 129, 129, 231, 231, 126,

};

I have formatted the array to make it a little more readable. The first line contains the width and height of the array, in this case 8 pixels by 8 pixels. The remaining 8 bytes contain the pixel data for each column of the sprite and are calculated using a simple formula shown below.



Notice how I have labelled the side of the graphic with 1, 2, 4 and so on. To calculate that the first column’s value is 126, I simply added up all of the values adjacent to the pixels I want to be turned on (white). 2 + 4 + 8 + 16 + 32 + 64 = 126. The remaining columns are calculated in exactly the same way. If you haven’t realized it already, you have just had your first experiences with binary numbers (see the sidebar Decimal vs Hexadecimal vs Binary Numbers for more information).

Sprites are often expressed in hexadecimal rather than decimal but this purely convention and makes no difference to the execution of the code.

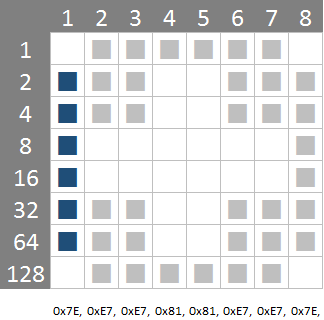
byte myFirstSprite[] = {

8, 8,

0x7E, 0xE7, 0xE7, 0x81, 0x81, 0xE7, 0xE7, 0x7E,

};

I have included my graphic design tool of choice in the repository – an Excel spreadsheet – and in it you will see a little trick I use to both design and calculate the array.



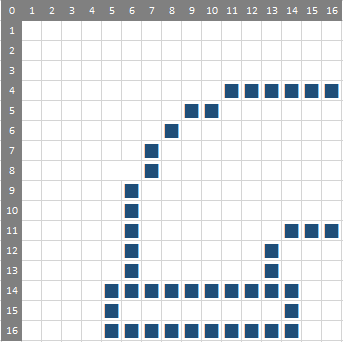
Below the graphic, my array is calculated automatically, using the formula shown below:

="0x"&DEC2HEX((SUMIF(B$2:B$9,"<>",$A$2:$A$9)),2)&","

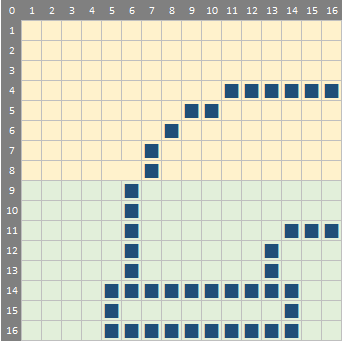
I would be guessing that if you are considering programming an Arduboy that you will be able to pull apart an Excel formula and understand how it works!

## Nodes Graphics

## Splash Screen Pipes



In the examples above, our sprites were 8 pixels wide by 8 pixels high. When defining a sprite that is 16 pixels or more in height, the array is specified from left to right then top to bottom. An example is shown in the graphic and array below:



byte logo\_elbow[] = {

16, 16,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xC0, 0x20, 0x10, 0x10, 0x08, 0x08, 0x08, 0x08, 0x08, 0x08,

0x00, 0x00, 0x00, 0x00, 0xE0, 0xBF, 0xA0, 0xA0, 0xA0, 0xA0, 0xA0, 0xA0, 0xB8, 0xE4, 0x04, 0x04,

};

The array follows a similar pattern to the 8 x8 example - the first two bytes describe the width and height of the sprite (in this case 16 pixels wide x 16 pixels high) followed by the image data. The first row of the array - shaded in yellow - corresponds to the top row of the image and the second row corresponds to the green section in the sprite.

The images for all of the pipes are shown below. The corresponding data arrays have been populated in the Images.h file in the source files of the solution.

