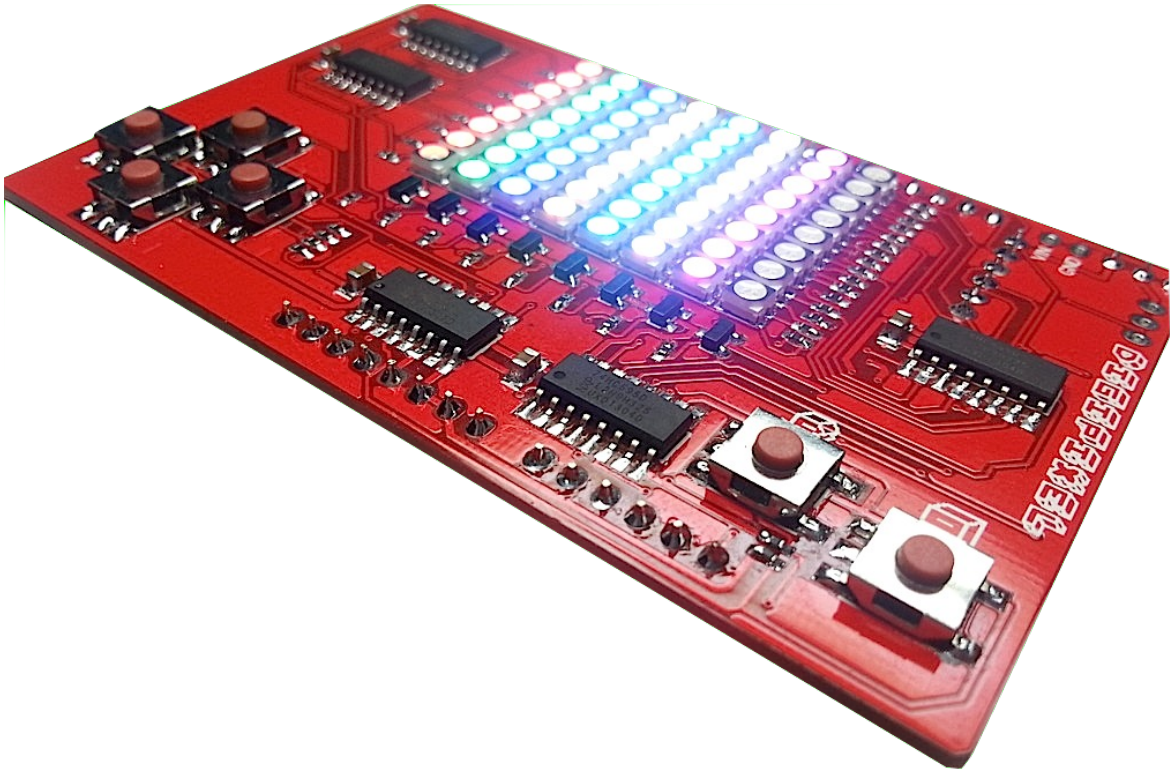
DigiPixel Library Manual



Version 1.0

List of functions:

1. drawScreen
2. clearScreen
3. drawBox
4. drawHorizontalLine
5. drawVerticalLine
6. fillScreen
7. setPixel
8. getPixel
9. checkBarrier
10. rotateScreen
11. invertColors
12. airWrite
13. saveButtonStates

1. drawScreen

Usage:
digiPixel.drawScreen();

Description:
This function will send the data stored within the bufferRed[8], bufferGreen[8] and bufferBlue[8] arrays, to the RGB LED display. Since this function takes care of drawing the screen, it will need to be called on a regular basis throughout your code. (for example, place it straight into your main loop like so:

```
void loop(){
   digiPixel.drawScreen();
}
```

Returns:
This function returns no value.

Arguments:
This function accepts no arguments.

2. clearScreen

Usage:
digiPixel.clearScreen();

Description:
This function will clear (make all 0‘s) the data stored within the bufferRed[8], bufferGreen[8], bufferBlue[8] and bufferBarriers[8] arrays. Now the next time that the digiPixel.drawScreen function is called, the buffers will be empty (contain all 0’s) and therefor the screen will be blank.

Returns:
This function returns no value.

Arguments:
This function accepts no arguments.

3. drawBox

Usage:
digiPixel.drawBox(byte fromX, byte fromY, byte toX, byte toY, byte color);

Description:
This function will draw a filled box on the screen in one of eight colors. The size and location is determined by the fromX / fromY and toX / toY variables It does this by modifying the contents of the bufferRed[8], bufferGreen[8], bufferBlue[8] and bufferBarriers[8] arrays.

Returns:
This function returns no value.

Arguments:
This function accepts five (5) arguments:

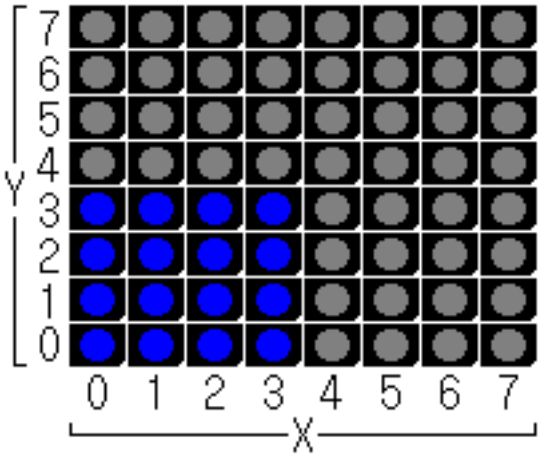
- *byte fromX - a number between 0 and 7*
- *byte fromY - a number between 0 and 7*
- *byte toX - a number between 0 and 7*
- *byte toY - a number between 0 and 7*
- *byte color - a number between 0 and 15 OR the name of a color. Refer to the color reference table in Annex A for list of valid colors.*

Example 1:

digiPixel.drawBox(0, 0, 3, 3, blue);

OR

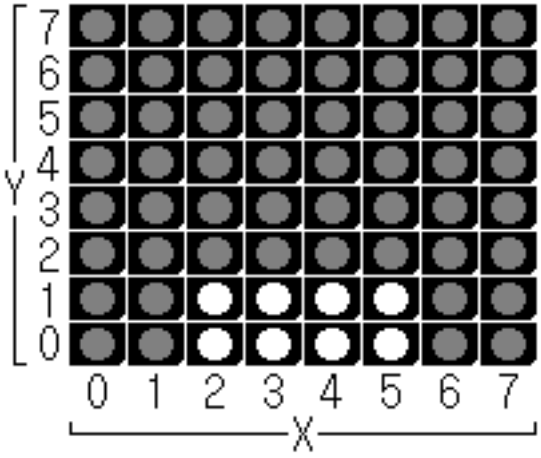
digiPixel.drawBox(0, 0, 3, 3, 4);



Example 2:
digiPixel.drawBox(2, 0, 5, 1, whitebarrier);

OR

digiPixel.drawBox(2, 0, 5, 1, 15);



4. drawHorizontalLine

Usage:
digiPixel.drawHorizontalLine (byte yLocation, byte fromX, byte toX, byte color);

Description:
This function will draw a horizontal line on the screen in one of eight colors. The length and location is determined by the *yLocation*, *fromX* and *toX* variables. It does this by modifying the contents of the bufferRed[8], bufferGreen[8], bufferBlue[8] and bufferBarriers[8] arrays.

Returns:
This function returns no value.

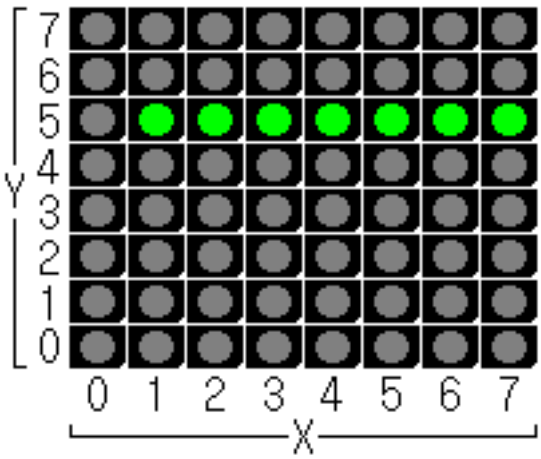
Arguments:
This function accepts four (4) arguments:

- *byte yLocation - a number between 0 and 7*
- *byte fromX - a number between 0 and 7*
- *byte toX - a number between 0 and 7*
- *byte color - a number between 0 and 15 OR the name of a color. Refer to the color reference table in Annex A for list of valid colors).*

Example 1:
digiPixel.drawHorizontalLine(5, 1, 7, green);

OR

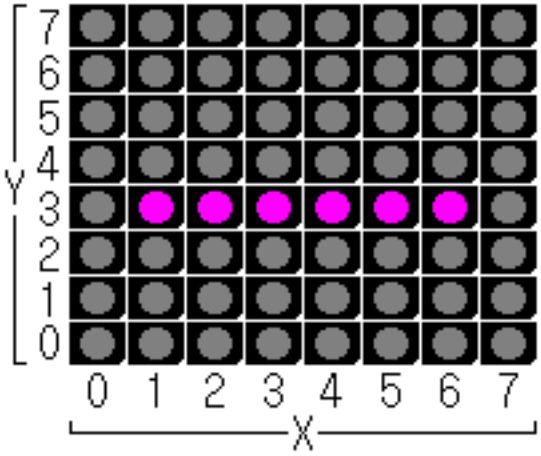
digiPixel.drawHorizontalLine(5, 1, 7, 2);



Example 2:
digiPixel.drawHorizontalLine(3, 1, 6, magenta);

OR

digiPixel.drawHorizontalLine(3, 1, 6, 5);



5. drawVerticalLine

Usage:
digiPixel.drawVerticalLine (byte xLocation, byte fromY, byte toY, byte color);

Description:
This function will draw a vertical line on the screen in one of eight colors. The length and location is determined by the *xLocation*, *fromY* and *toY* variables. It does this by modifying the contents of the bufferRed[8], bufferGreen[8], bufferBlue[8] and bufferBarriers[8] arrays.

Returns:
This function returns no value.

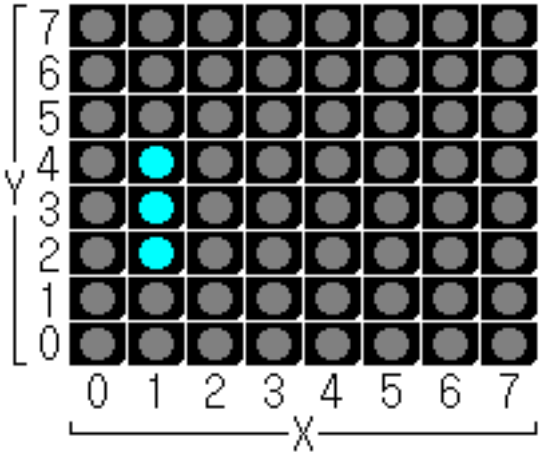
Arguments:
This function accepts four (4) arguments:

- *byte xLocation - a number between 0 and 7*
- *byte fromY - a number between 0 and 7*
- *byte toY - a number between 0 and 7*
- *byte color - a number between 0 and 15 OR the name of a color. Refer to the color reference table in Annex A for list of valid colors).*

Example 1:
digiPixel.drawVerticalLine(1, 2, 4, cyan);

OR

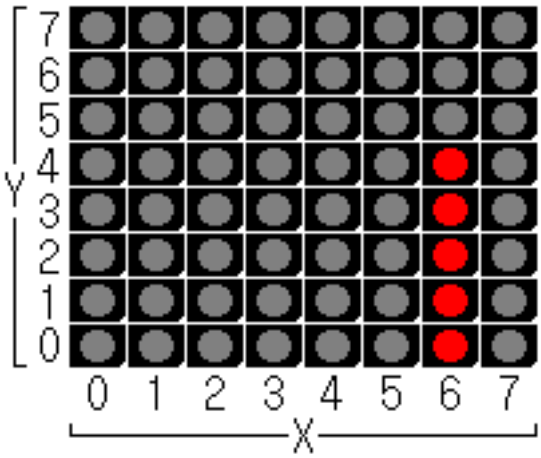
digiPixel.drawVerticalLine(1, 2, 4, 6);



Example 2:
digiPixel.drawVerticalLine(6, 0, 4, red);

OR

digiPixel.drawVerticalLine(6, 0, 4, 1);



6. fillScreen

Usage:
digiPixel.fillScreen(byte color);

Description:
This function will fill the entire screen with one single color. It does this by modifying the contents of the bufferRed[8], bufferGreen[8], bufferBlue[8] and bufferBarriers[8] arrays.

Returns:
This function returns no value.

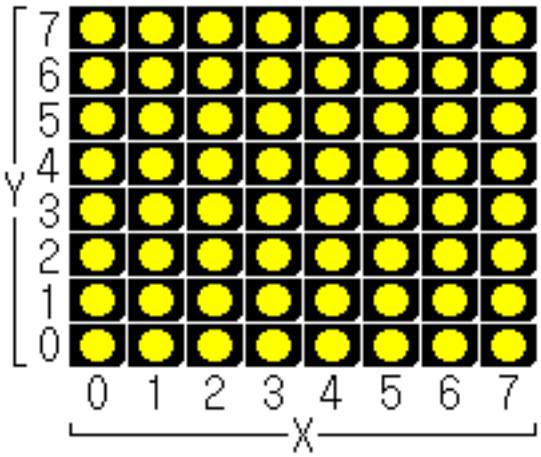
Arguments:
This function accepts one (1) argument:

- *byte color - a number between 0 and 15 OR the name of a color. Refer to the color reference table in Annex A for list of valid colors).*

Example 1:
digiPixel.fillScreen(yellow);

OR

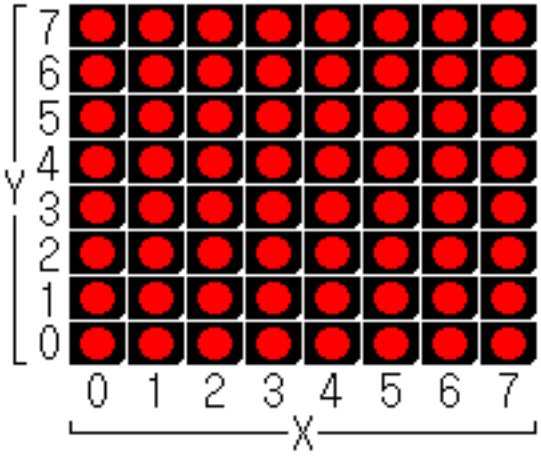
digiPixel.fillScreen(3);



Example 2:
digiPixel.fillScreen(red);

OR

digiPixel.fillScreen(1);



7. setPixel

Usage:
digiPixel.setPixel(byte pixelX, byte pixelY, byte color);

Description:
This function will make one single pixel a certain color, the location is determined by the *pixelX* and *pixelY* variables. It does this by modifying the contents of the bufferRed[8], bufferGreen[8], bufferBlue[8] and bufferBarriers[8] arrays.

Returns:
This function returns no value.

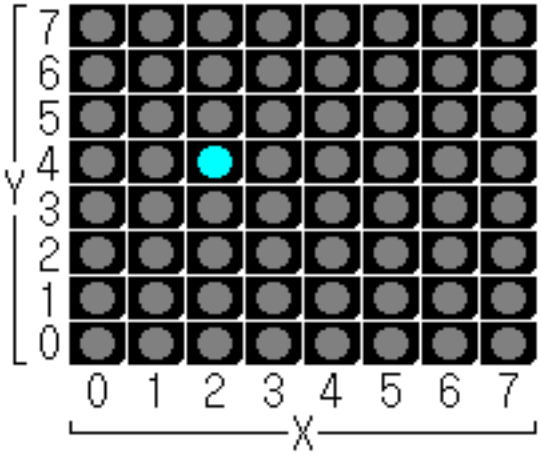
Arguments:
This function accepts three (3) arguments:

- *byte pixelX - a number between 0 and 7*
- *byte pixelY - a number between 0 and 7*
- *byte color - a number between 0 and 15 OR the name of a color. Refer to the color reference table in Annex A for list of valid colors).*

Example 1:
digiPixel.setPixel(2, 4, cyan);

OR

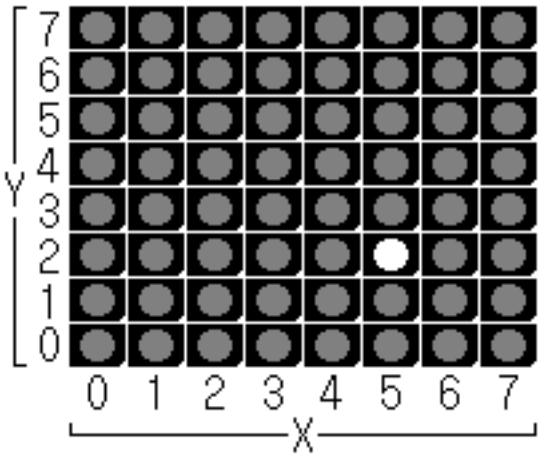
digiPixel.setPixel(2, 4, 6);



Example 2:
digiPixel.setPixel(5, 2, whitebarrier);

OR

digiPixel.setPixel(15);



8. getPixel

Usage:
digiPixel.getPixel(byte pixelX, byte pixelY);

Description:
This function will return the color of a certain pixel by reading the contents of the bufferRed[8], bufferGreen[8], bufferBlue[8] and bufferBarriers[8].

Returns:
This function returns one (1) byte:

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
not used	not used	not used	not used	barrier	blue	green	red

Arguments:
This function accepts two (2) arguments:

- *byte x - a number between 0 and 7*
- *byte y - a number between 0 and 7*

Example 1:
Assuming that the pixel at 2, 4 is cyan (E.G. from the first setPixel example)

byte myReturnedColor = digiPixel.getPixel(2, 4);

myReturnedColor will now contain the decimal number 6. Here is the binary breakdown:

bit 7 (not used)	bit 6 (not used)	bit 5 (not used)	bit 4 (not used)	bit 3 (barrier)	bit 2 (blue)	bit 1 (green)	bit 0 (red)
0	0	0	0	0	1	1	0

Example 2:
Assuming that the pixel at 5, 2 is whitebarrier (E.G. from the second setPixel example)

byte myReturnedColor = digiPixel.getPixel(5, 2);

myReturnedColor will now contain the decimal number 15. Here is the binary breakdown:

bit 7 (not used)	bit 6 (not used)	bit 5 (not used)	bit 4 (not used)	bit 3 (barrier)	bit 2 (blue)	bit 1 (green)	bit 0 (red)
0	0	0	0	1	1	1	1

9. checkBarrier

Usage:
if (digiPixel.checkBarrier(byte pixelX, byte pixelY) == true);
{
// now that we know there is a barrier at that location, run this piece of code...
}

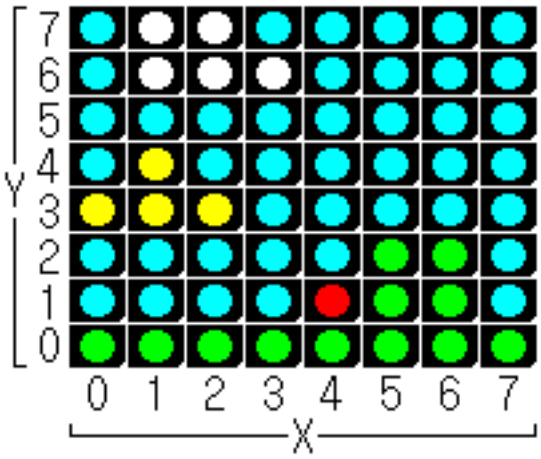
Description:
This function will return a boolean variable with a true or false condition depending on whether a certain pixel is a barrier or isn't. It does this by reading the contents of the bufferBarriers[8] array.

Returns:
This function returns a boolean variable (I.E. either TRUE or FALSE).

Arguments:
This function accepts two (2) arguments:

- *byte x - a number between 0 and 7*
- *byte y - a number between 0 and 7*

Example:



Let's say we have a Super Mario Bros type game where our character is the red pixel and is running to the right of screen. In order to prevent the character from being able to move through walls, we would ensure that the green pixels in the image above are set to 'greenbarrier'. Then we can run some code that will allow the right button to move our player only if there is NOT a barrier next to us, like so:

Example:
if (digiPixel.ButtonRightPressed & digiPixel.checkBarrier(PlayerX + 1, PlayerY) == false)
{
// put some code here to allow the player to move...
}

Notice in the code above that we are checking for a barrier just to the right of the player I.E. PlayerX + 1 is the pixel to the immediate right of the player. You could also check above, below and to the left of the player like so:

•Above - *digiPixel.checkBarrier(PlayerX , PlayerY + 1)*
•*Below - digiPixel.checkBarrier(PlayerX , PlayerY - 1)*
•Left - *digiPixel.checkBarrier(PlayerX - 1 , PlayerY)*

10. rotateScreen

Usage:
digiPixel.rotateScreen(int degrees);

Description:
This function will rotate the screen 90, 180 or 270 degrees in an anti-clockwise direction. It does this by modifying the contents of the bufferRed[8], bufferGreen[8], bufferBlue[8] and bufferBarriers[8] arrays.

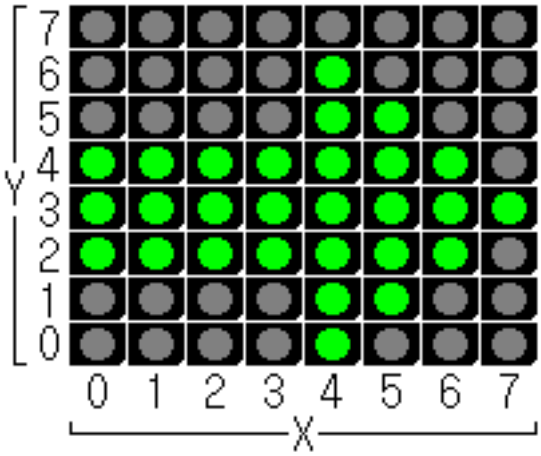
Returns:
This function returns no value.

Arguments:
This function accepts one (1) argument:

• *int degrees - valid numbers are 90, 180 or 270*

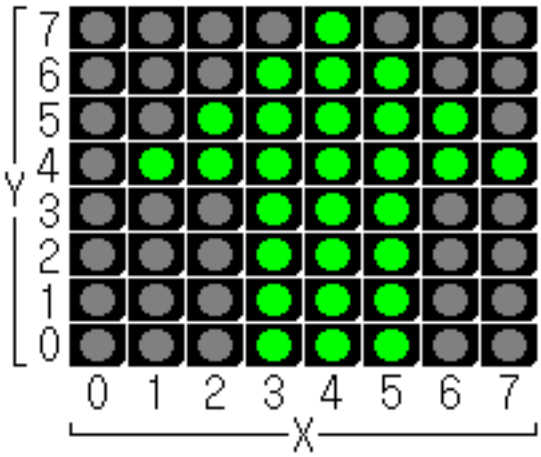
Examples.

Here is our original image:

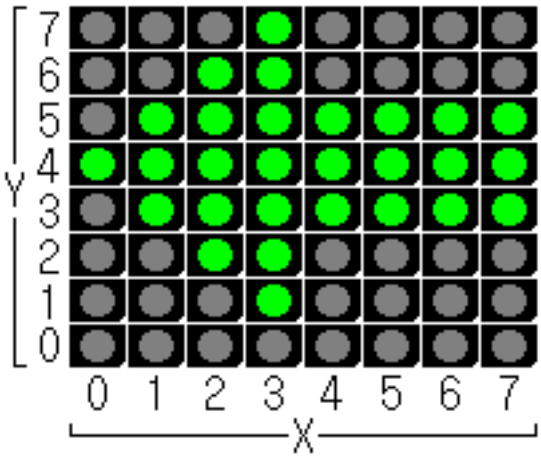


Now we will rotate this image, 90, 180 and 270 degrees.

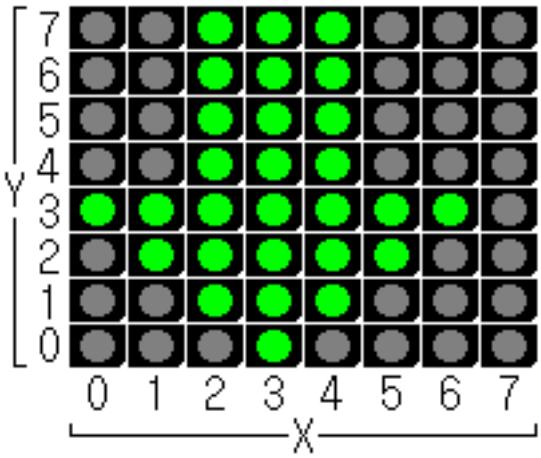
Example 1:
digiPixel.rotateScreen(90);



Example 2:
digiPixel.rotateScreen(180);



Example 3:
digiPixel.rotateScreen(270);



11. invertColors

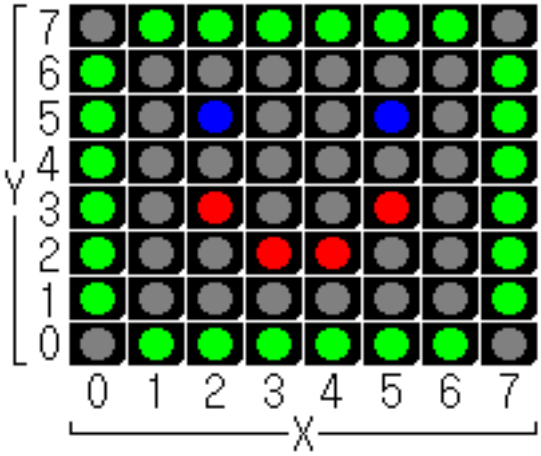
Usage:
digiPixel.invertColors();

Description:
This function will invert all colors currently displayed on the screen. It does this by modifying the contents of the bufferRed[8], bufferGreen[8] and bufferBlue[8] arrays.

Returns:
This function returns no value.

Arguments:
This function accepts no arguments.

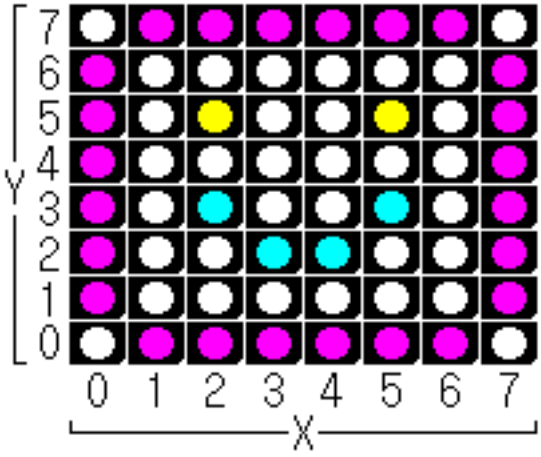
Example 1:
If we start with this image:



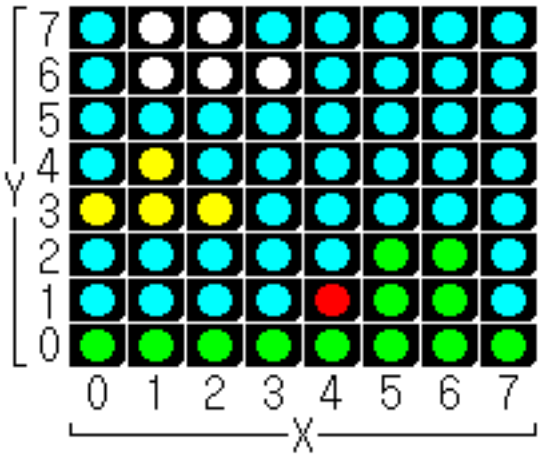
and now we add this code:

digiPixel.invertColors();

we will get this:



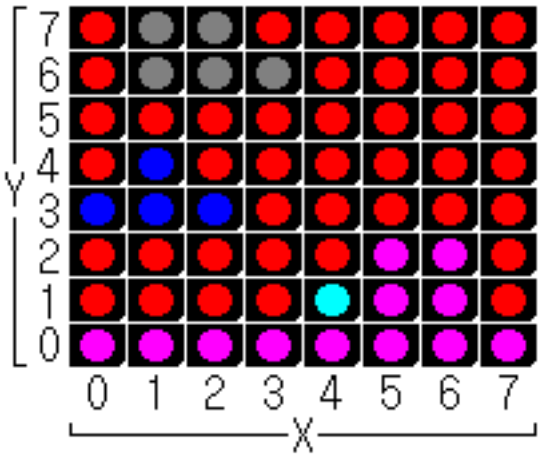
Example 2:
If we start with this image:



and now we add this code:

digiPixel.invertColors();

we will get this:



12. airWrite

Usage:
digiPixel.airWrite(int flashDelay);

Description:
This function allows you to capture images and messages in the air with your camera. (for best results, set your shutter speed to a slow setting). It does this by modifying the contents of the first byte in the bufferRed[8], bufferGreen[8] and bufferBlue[8] arrays.

Returns:
This function returns no value.

Arguments:
This function accepts one (1) argument.

- *int flashDelay - any number from 0 to 2,147,483,647. This number is used as a delay (in milliseconds) between showing each line of information on the LED's. A good starting point would be 1 which we will use in the example.*

Example:
This example will set the pixels within the first column, to random colors. The airWrite function displays data only in the first column. You then wave the DigiPixel in the air to draw in the air. (There is a delay of 1mS between generating new random pixels.)

```
digiPixel.bufferRed[0] = random(256);     // random# from 00000000 to 11111111
digiPixel.bufferGreen[0] = random(256); // random# from 00000000 to 11111111
digiPixel.bufferBlue[0] = random(256);   // random# from 00000000 to 11111111
digiPixel.airWrite(1);
```

13. saveButtonStates

Usage:
digiPixel.saveButtonStates();

Description:
This function will store the state of the six buttons in the six button boolean variables. You are then able to directly access these variables throughout your code. The six button boolean variables are:

- buttonUpPressed
- buttonDownPressed
- buttonLeftPressed
- buttonRightPressed
- buttonBPressed
- buttonAPressed

Just like the digiPixel.drawScreen(); function, this function will need to be called periodically throughout your code in order to capture the current status of each button. For example, you can place it straight into your main loop, like so:

```
void loop(){
    digiPixel.saveButtonStates();
}
```

Returns:
This function returns no value.

Arguments:
This function accepts no arguments

Example:
As long as you are checking the button states periodically, you can be sure the data stored within the button boolean variables will be the current status of those buttons. So if you wanted to move a certain pixel around the screen, you could do it like so:

```
if (digiPixel.buttonUpPressed == true){
    pixelY++;
}
if (digiPixel.buttonDownPressed == true){
    pixelY--;
}
if (digiPixel.buttonLeftPressed == true){
    pixelX--;
}
if (digiPixel.buttonRightPressed == true){
    pixelX++;
}
```

Annex A

Color Reference Table

Name of Color	Decimal Value	Looks Like
black	0	
red	1	
green	2	
yellow	3	
blue	4	
magenta	5	
cyan	6	
white	7	
blackBarrier	8	
redBarrier	9	
greenBarrier	10	
yellowBarrier	11	
blueBarrier	12	
magentaBarrier	13	
cyanBarrier	14	
whiteBarrier	15	