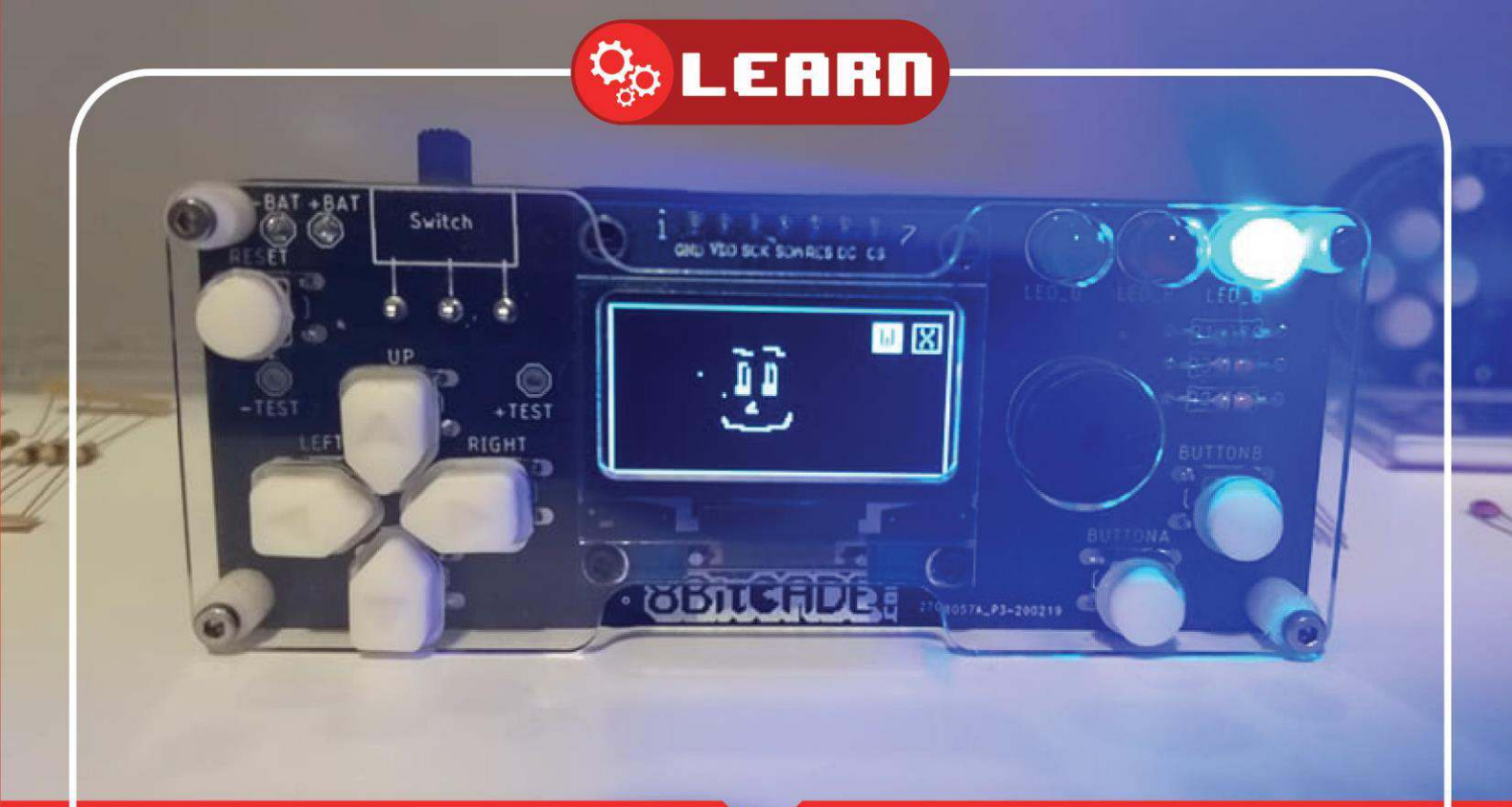
LEARN



8Bit Project: Etch-A-Sketch

With 8BitCADE

Compatible with:

8BitCADE XL

AND

8BitCADE

Contents

Written by the 8BitCADE Team

Support@8bitcade.com

Version 1

Note that:

This text showcases a task/challenge that you should attempt – all levels of coders should try and attempt these without seeing the answer.

This text showcases something that you should be doing, regardless of your coding ability. This is usually ensuring that your program is exactly like the one presented.

For beginners, we recommend ensuring your software is exactly like the one we present. Your main focus should be on understanding the coding functions, getting used to the coding syntax and understanding why we use specific functions.

For intermediate to advanced coders, we recommend that you do this tutorial first, then try writing your program with the challenges used as roadmaps/guidelines.

The way this booklet is written is:

1. The brief of what we want the code in this section to achieve.
2. Task: Can you code it by yourself? Here are the functions and what the functions mean
3. Code Explanation
4. Final Code, copy this to get a program just like the one we present!

This allows for the beginners to get a grasp on what each function and coding statement means and dip the deep end by coding some sections by themselves. For intermediate coders, it gives a challenge and for advanced coders it allows the code to be planned out ready for you to write it up using your logic – then all levels can check it with the presented code and adjust it accordingly.

Have fun, if you have any errors with the code. Check your code with the final code, in the final pages of the booklet.

Bring out the learner in you – with 8BitCADE!

-8BitCADE Team

8BitCADE Project: Etch-A-Sketch

In this tutorial, we are going to be learning how to use Arduino to program an Etch-A-Sketch game for our 8BitCADE/XL.

To fully understand this tutorial, you need to be able to understand basic Arduino syntax and Arduino classes. We advise that the following tutorials are completed before starting this project:

- Arduino Basic's: Classes
- Arduino Basic's: Library & Board Setup

File Breakdown

8Bit-Etch-A-Sketch: The main Arduino file that contains the setup/loop

Controller.h: Defines the Class Controller, that will be used to read and process the controls

Controller: Creates all of the class methods of the class Controller

Controller.h

The controller class deals with the buttons controls of the 8BitCADE and has methods inside the class to deal with button debounce. In this header file, we define all attributes and methods for the class. We use header and .ino files to help organize our code. Header files are for defining attributes of a class or library and .ino files are for running code or in our case, writing what does inside the actual methods we define.

Your task is to create a new header file called "Controller.h"

A snippet from the "Classes 101 Booklet", be sure to look at that before attempting this tutorial.

To create a new file in Arduino, you need to press the toggle menu on the right of the files bar (under serial monitor) and click "NewTab"

Here we write the file name and then the file extension: ino is for Arduino based programs and will be used for the file that will contain all of the class

When you create this file, it's important to add the file extension (.h) so the program knows it isn't a default .ino file but rather a header file.

Upon file creation, you'll notice that Arduino has not produced any void loops or setups. This is as there can only be one of each in the program. Also, as this is a header file, we will only be defining variables and classes.

Note that all attributes and methods need to be defined before we can use them, if you reference a variable that hasn't been created, you will get an error. Below are some helpful definitions before you copy the final code:

point [Variable name] To create a structure with int X and int Y. We defined the structure in the main section of the code.

bool data type is for true and false (or 1/0)

long data types are like integers but have a larger range.

To create a constructor method, simple write **[nameOfClass]();**

```
5  typedef struct point {
6    int x;
7    int y;
8  };
```

Point, will be defined in the main file do not worry about writing this yet, we will declare the variables in the next section of the tutorial – meaning you will get errors if you run your code at this moment in time, do not worry. Point is a structure. A structure is a defined datatype that holds groups of data, aka int x and int y in this case. We can access these through using point.x or point.y and it will return the integer value of x or y

Controller.h Code

```
1  class Controller
2  {
3    public:
4      //CONSTRUCTOR
5      Controller();
6      //Create from the point structure a position varaible. This will be used to move the cursor
7      point position;
8      //Define the method UPDATE used to update the controls every loop
9      void update();
10
11     //BOOLEAN VARIABLES THAT STORES WHETHER BUTTON PRESSED IS A/B
12     bool AButtonPressed;
13     bool BButtonPressed;
14
15     //BOOLEAN VARIABLE THAT STORES WHETHER THE PREVIOUS BUTTON WAS PRESSED
16     bool previousUPButtonPressed;
17     bool previousDOWNButtonPressed;
18     bool previousLEFTButtonPressed;
19     bool previousRIGHTButtonPressed;
20     bool previousAButtonPressed;
21     bool previousBButtonPressed;
22
23     //BOOLEAN VARIABLES THAT CHECKS IF THE BUTTON PRESS IS UPDATED (A/B PRESSED)
24     bool AButtonWasPressed;
25     bool BButtonWasPressed;
26
27
28     //BOOLEAN VARIABLES THAT CHECKS IF THE BUTTON CAN BE PRESSED. USED FOR COLLISION/RESTRICTION
29     bool UPButtonCanBePress = 1;
30     bool DOWNButtonCanBePress = 1;
31     bool LEFTButtonCanBePress = 1;
32     bool RIGHTButtonCanBePress = 1;
33
34     //HOLDS POSITION OF 'CURSOR' OR CHARACTER
35     int pos;
36
37     long debounceDelay = 75; //TIME WAITED UNTILL NEXT BUTTON IS INPUTTED
38     long currenttime = 0; //CURRENT TIME
39     long lastHoldTime = 100; //TIME, IN MILLIS, OF LAST BUTTON PRESS
40
41 };
```

Here we can see, on line 5, we need a constructor, we can leave this blank if we don't have any values to pass through upon creating an object. To do this, use the [classname]() and leave the parameters blank. See more about constructors in the "**Classes 101**" booklet.

Line 7: **Point position**; defines an attribute from the structure point which we formed in the main Calculator file.

Also note that when defining any variable holding a millis value, ensure you use a data type that can deal with large amounts of data – as recording the time can amount to a lot of data quickly if not updated. Here we used long as it has a large plus and minus range. (see lines 37 to 39).

Please copy this code to ensure you have the correct program – this is important as it defines all the variables we will use. A typo here is usually a major culprit for errors!

Controller

The controller file deals with the methods for the controller class. It's important when writing in this file to call files using the "outside class" calling method, as discussed in the "classes 101" booklet:

"void [Classname] :: [Method Name]()"

This file mainly focuses on dealing with button debounce. *Your task is to create a new tab file, name it "Controller" – it will default to .ino*

Before we start implementing debounce algorithms, we have to first create the constructor method, we can utilize this to set the controller.position.x and y position values to 0 (or if we needed the player to start in the middle, we could, therefore, set these values to the centre x and centre y position)

Note that when referencing class attributes in a class, we do not need to write "classname. attribute" we can just write "attribute". It's only outside of classes where we need to specify the classname.

The way the controls will work is that we will take in button inputs, and alter the position.x and position.y values accordingly. Remember that "position" is a structure that contains two integer variables "x" and "y"

We also use a Serial.print to check that the object was created correctly.

Your task is to write the constructor method for the class Controller() and create an empty function called "update". Be sure to use "outside class" method declaration. The below functions might help:

***position.x** will access the x position of the controller and replacing the x with a y will access they position for the controller.*

***Serial.print("Hello World");** Will print to the console the string "Hello World"*

***Void [Class name] :: [function name]() { [code] }** will create a function for the class specified.*

Controller Method Creation Code

On the right, is the code you should type in.

What is debounce in a button?

Debounce, in buttons, is when the button input is read multiple times in a short amount of time. Without debouncing, the input could be up to 4 times and therefore would alter the position value too much!

```
1  Controller::Controller()
2  {
3    Serial.print("BUTTON TEST BUTTON TEST");
4    position.x = 0;
5    position.y = 0;
6  }
7
8  void Controller::update()
9  {
10
11 }
```

Your task is to write a simple algorithm that takes one button (UP_BUTTON) and ensures that the button is registered as pressed with a debouncing algorithm. The below functions will come in handy:

Aboy.pressed(UP_BUTTON) will return either true or false depending on if the up button is pressed or not (can put any button in LEFT_BUTTON etc)

Millis() will return the time in milliseconds since the program started running

currentTime = Millis(); Will capture that time

Once the button is registered as press, we update the position, use position.y to access the global y position of the class controller. For up we would add 1, for down we would -1.

Button Debounce Code

To stop this from happening, we use the below algorithm (repeated for each button) to combat this:

```
27  // A and B Buttons
28  if (aboy.pressed(A_BUTTON) and previousAButtonPressed == 1 and (millis() - lastHoldTime) > 250) {
29    lastHoldTime = millis();
30    AButtonPressed = true;
31  }
```

Here, if the button is currently being pressed (**aboy.pressed(UP_BUTTON)**) and it was previously pressed (**previousUPButtonPressed** would be 1 if it is turned on and 0 if It is off) and the time from the last recorded button press is greater than 150 mili seconds, and the button can be pressed, then change the position accordingly, in this case, add one to the position structure y. (structures are accessed using [DOT] StructureName.variableName).

aboy.pressed(UP_BUTTON) = gets the current button status of the "UP_BUTTON".

PreviousUPButtonPressed = Is the status of the button UP, in the last cycle of code (aka the last reading)

```
27  // A and B Buttons
28  if (aboy.pressed(A_BUTTON) and previousAButtonPressed == 1 and (millis() - lastHoldTime) > 250) {
29    lastHoldTime = millis();
30    AButtonPressed = true;
31  }
32  else {
33    AButtonPressed = false;
34  }
35  if ((aboy.pressed(B_BUTTON) == 1 and previousBButtonPressed == 1 and (millis() - lastHoldTime) > 250) ) {
36    lastHoldTime = millis();
37    BButtonPressed = true;
38  }
39  else {
40    BButtonPressed = false;
41  }
```

(millis() – A_lastHoldTime) > 150 = This allows us to calculate how much time has passed since we last took a reading. We can see that as **A_lastHoldTime,** in the if statement is equal to **milis();** therefore we record the time it was pressed. If it is less than 150 then we know it is the button bouncing and we know not to record the button.

Note that we have to use separate lastholdtime variables to allow both buttons to be read at the same time, if we used the same variable, when one is read, the other button cannot be read until the time limit is over. Hence using A_lastholdtime and B_lastholdtime.

While we do not use the A and B buttons, it's important to note the above code as this "Controller" class is used for a lot of 8BitCADE projects.

aboy.pressed(A_BUTTON) = Checks if the button is currently pressed

previousAButtonPressed = Checks the status of the button, in the last cycle of code (aka the last reading)

(millis() – A_lastHoldTime) > 250) = only takes the reading if there has been more then 250 milliseconds from now to the last time we took a button reading.

UPButtonCanBePress is our variable used to deal with collisions and boundaries. We can turn on and off each button allowing us to have full control over where the sprite is allowed to go. I'll go over the function that controls this aspect after this, but for now know that when the player reaches a boundary, we can turn these on and off to stop movement in one direction and to allow movement in the opposite direction.

```
  //Set all previous button pressed variables
  previousUPButtonPressed = aboy.pressed(UP_BUTTON);
  previousDOWNButtonPressed = aboy.pressed(DOWN_BUTTON);
  previousLEFTButtonPressed = aboy.pressed(LEFT_BUTTON);
  previousRIGHTButtonPressed = aboy.pressed(RIGHT_BUTTON);
  previousAButtonPressed = aboy.pressed(A_BUTTON);
  previousBButtonPressed = aboy.pressed(B_BUTTON);
```

Here we set the previous button pressed variables to the current button press status for the next loop. This is part of the above algorithm. See the "full code" below to check if your file is correct.

The full code for the methods of the class in file "Controller" is below:

```
Controller::Controller()
{
  Serial.print("BUTTON TEST BUTTON TEST");
  position.x = 0;
  position.y = 0;
}

void Controller::update()
{
  if (aboy.pressed(UP_BUTTON) and previousUPButtonPressed == 1 and (millis() - lastHoldTime) > 150 and UPButtonCanBePress) {
    lastHoldTime = millis();
    position.y += 1;
  }
  if ((aboy.pressed(DOWN_BUTTON) == 1 and previousDOWNButtonPressed == 1 and (millis() - lastHoldTime) > 150) and DOWNButtonCanBePress) {
    lastHoldTime = millis();
    position.y -= 1;
  }
  if (aboy.pressed(LEFT_BUTTON) and previousLEFTButtonPressed == 1 and (millis() - lastHoldTime) > 150 and LEFTButtonCanBePress) {
    lastHoldTime = millis();
    position.x -= 1;
  }
  if ((aboy.pressed(RIGHT_BUTTON) == 1 and previousRIGHTButtonPressed == 1 and (millis() - lastHoldTime) > 150) and RIGHTButtonCanBePress) {
    lastHoldTime = millis();
    position.x += 1;
  }
  if (aboy.pressed(A_BUTTON) and previousAButtonPressed == 1 and (millis() - A_lastHoldTime) > 250) {
    A_lastHoldTime = millis();
    AButtonPressed = true;
  }
  else {
    AButtonPressed = false;
  }
  if ((aboy.pressed(B_BUTTON) == 1 and previousBButtonPressed == 1 and (millis() - B_lastHoldTime) > 250) ) {
    B_lastHoldTime = millis();
    BButtonPressed = true;
  }
  else {
    BButtonPressed = false;
  }
  previousUPButtonPressed = aboy.pressed(UP_BUTTON);
  previousDOWNButtonPressed = aboy.pressed(DOWN_BUTTON);
  previousLEFTButtonPressed = aboy.pressed(LEFT_BUTTON);
  previousRIGHTButtonPressed = aboy.pressed(RIGHT_BUTTON);
  previousAButtonPressed = aboy.pressed(A_BUTTON);
  previousBButtonPressed = aboy.pressed(B_BUTTON);
}
```

Please copy this code to ensure you have the correct program.

Drawing Mechanism

To create the drawing mechanism, we use a new function called drawPixel, within this class we have a method that calculates the equation depending on the sign. This will be written in your "8Bit-Etch-A-Sketch.ino" file.

```
1 #include <Arduboy2.h>
2 Arduboy2 aboy;
```

Before we begin, we must use the #include (library name) to include a library. Here we can also see how we redefine the library name from Arduboy2 to aboy, as when we call library function, we have to call them using the library name: libraryname.libraryfunction();

Next, we have the structure we define earlier called point.

```
5 typedef struct point {
6   int x;
7   int y;
8 };
```

```
9 #include "controller.h"
10 Controller Controller;
```

Currently, our main file is not connected to the controller header file, this is because we need to reference the header file before the setup (we don't need to reference .ino files as they will be run automatically). To do this we use #include again, this time we use speech marks as the header file is in the directory of the current sketch (meaning both files are in the same folder). We use <> when it is a library file located in the Arduino library folder. Here we also create an object of the class Controller called Controller, allowing us to use the methods and attributes of the class Controller – if that doesn't make sense to you, check out the classes 101 guides to learn more about classes and object-oriented programming.

Finally, before the void setup, we need to define some variables we will be suing to get the drawing mechanism working:

```
12 bool colour = true;
13 bool drawstate = true;
14
15 point previouspoint;
16
```

Colour = As we are working in Back (0 or false) and white (1 or true) we can make the colour a Boolean value, as true to Arduino is simply 1, therefore we can control and change the colour that is drawn

Drawstate = allows us to turn on and off drawing, as we want the user to be able to draw, then "lift up" their pen and move to draw on another part of the screen.

Previouspoint = save the X and Y coordinates of the previous point – allowing us to fill it in with the desired colour. We can treat point like a data type because it is a structure made up of many different data types, in this case, int x and int y.

```
1  #include <Arduboy2.h>
2  Arduboy2 aboy;
3
4  typedef struct point {
5    int x;
6    int y;
7  };
8
9  #include "controller.h"
10 Controller Controller;
11 bool colour = true;
12 bool drawstate = true;
13
14 point previouspoint;
15
```

The first part of the main file should look like this (on the left).

Please copy this code to ensure you have the correct program.

Void Setup

The void setup is a function where any code inside runs once and as soon as the program runs. We can use this to clear the screen, setup values and set up pin modes.

Your task is to see if you can write the void setup for this program. The things we need to do is boot the Arduboy (to initialize the library), set the pin modes as OUTPUTs of the 3 LEDs (10 = Yellow, 3 = Green, 9 = Red), clear the Arduboy screen and define the controller.x and controller.y values so the cursor begins in the middle of the screen. The below functions might come in handy:

***Void setup() { [CODE] }** = code that runs once as soon as the program starts running*

***aboy.boot()** = will boot the Arduboy without the Arduboy splash screen, it will also initialize the library*

***pinMode([PinNumber],[INPUT/OUTPUT);** = Will define if a pin is an input or output if it will be receiving data or sending out data*

***aboy.clear();** = In simple terms, this function (clear) will clear the screen. It will clear the screen buffer and display it. This means it will clear the screen and whatever is on it. Every time we print something, aboy.print("Hello World"); it gets stored in the screen buffer. When we use the aboy.display() it will display the screen buffer, aka the "Hello World".*

***aboy.height()** = will return the integer value of the screen height*

***aboy.width()** = will return the integer value of the screen width*

Please copy this code to ensure you have the correct program.

```
16 void setup() {
17   aboy.boot();
18   aboy.clear();
19   Serial.begin(9600);
20
21   //LED PINMODES
22   pinMode(10, OUTPUT); //Yellow
23   pinMode(3, OUTPUT); //Green
24   pinMode(9, OUTPUT); //Red
25
26   Controller.position.y = aboy.height() / 2;
27   Controller.position.x = aboy.width() / 2;
28 }
29
```

Void Loop

Here, after finally setting everything up, we can begin writing the core mechanism of our game.

Your task is to write a simple program that allows the user to draw. The below functions might come in handy:

Aboy.drawPixel(Xpos, Ypos, Colour); *= Will draw a single pixel on the X and Y position specified. It will turn it either black or white (specified by the colour attribute.*

Controller.update(); *= Runs the update method of the controller which read and updates the position based on the button readings.*

```
30 void loop() {
31   Controller.update();
32   aboy.drawPixel(Controller.position.x, Controller.position.y, colour);
33   aboy.display();
```

As we are not clearing the screen after every pixel drawn, we can see that the previous pixel drawn remains and allows us to draw.

Note that we use the Controller.position.x and Controller.position.y variables as these are constantly updated by the controller – this is the position the user can alter by using the buttons. Therefore, when the user clicks the up arrow and the position updates, this function will turn that pixel to the desired colour.

Adding Draw state

At the moment we have a very basic drawing app, that allows us to draw continuously. Next, we want to add a feature that allows the user to "lift" the pen and "move it" and then "put down" the pen and continue drawing. To do this we can use the drawstate variable we declared beforehand.

Your task is to allow the user to toggle if they want to draw or not using the B Button, using the draw state. To demonstrate to the player that they are not drawing, turn on the blue LED. The below functions could be useful:

aboy.getPixel(Xpos,Ypox); *= Will return the value of that pixel, in our case either 1 for white or 0 for black.*

To toggle the drawstate, we will be using the B button. To access the Bbutton, we use ***Controller.BButoonPressed*** *This returns true if it is pressed. Using ClassName.ClassAttribute allows you to access all of the classes attributes, if this confuses you, check out the Classes 101 guide.*

!variable *= the exclamation mark (!) is logical Not, for those acquainted with logic gates, it is a NOT gate. For those that are not, it simply inverts a Bool value. Turning true to false and false to true. Lets say BoolVariable = 1 (true) then !BoolVariable becomes 0 (false).*

digitalWrite(Pin, State) *= NOTE that the LEDs have been "pulled up" using internal resistors, however, we do not need to know about this. We just need to remember that: digitalWrite 1 or high will turn the LED OFF and 0 or LOW will turn the LED ON.*

Normally, HIGH would turn on the Blue LED and LOW turn off the LED. However, now we have to write: digitalWrite(9, LOW). Note that 9 is the pin that the blue LED is connected to. Did you know? That you can replace the HIGH value with a bool variable, in our case we could use the drawstate, so the LED displays the value of the drawstate. This would look like digitalWrite(9,drawstate);

Did you know? You can write if(BoolVariable) and if the bool variable is true, then the if statement will run, if it is false, then the if statement won't run. This means we don't have to write if(BoolVariable == true)

```
void loop() {
  Controller.update();

  if (drawstate) {
    aboy.drawPixel(Controller.position.x, Controller.position.y, colour);
  }
  else {
    aboy.drawPixel(previouspoint.x, previouspoint.y, !(aboy.getPixel(previouspoint.x, previouspoint.y)));
    // Before we set the pixel to the invert of what it was before, here we are resetting it back.
    // Meaning there is no overal change in the pixel and therefore we are not drawing/removing.
    aboy.drawPixel(Controller.position.x, Controller.position.y, !(aboy.getPixel(Controller.position.x, Controller.position.y)));
    // Draw cursor in the invert of whatever the current colour is
  }

  if (Controller.BButtonPressed == true) {
    drawstate = !drawstate;
    //As draw state is a bool, we can use ! to inverse the current value - 1 turns into 0 and  0 turns into 1
    digitalWrite(9, drawstate);
    // Turn on/off LED depending on state of draw.
  }
  aboy.display();
  // set the current x and y position of the point to the previous variables to use on the next loop run.
  previouspoint.x = Controller.position.x;
  previouspoint.y = Controller.position.y;
}
```

Please copy this code to ensure you have the correct program.

Before we begin drawing, we need to check if we should be drawing or not. We check with the drawstate to decide what we should do – if drawstate is true then we run the below code:

```
  if (drawstate) {
    aboy.drawPixel(Controller.position.x, Controller.position.y, colour);
  }
```

Using the function drawPixel, we simply replace the current pixel with the desired colour – this should look familiar as it is the same line we wrote in the task beforehand!

However, if the draw state is false, then we cannot draw and should only show a cursor. To do this

```
  else {
    aboy.drawPixel(previouspoint.x, previouspoint.y, !(aboy.getPixel(previouspoint.x, previouspoint.y)));
    // Before we set the pixel to the invert of what it was before, here we are resetting it back.
    // Meaning there is no overal change in the pixel and therefore we are not drawing/removing.
    aboy.drawPixel(Controller.position.x, Controller.position.y, !(aboy.getPixel(Controller.position.x, Controller.position.y)));
    // Draw cursor in the invert of whatever the current colour is
  }
```

Here we can see we have two draw pixel functions, both of these means create a cursor that does not draw. It also turns the cursor the inverse colour of what pixel it is "hovering" over. AKA if you were to move the cursor over a white pixel, the cursor would turn black so you can see it better. The same if it was black, the cursor would be seen as white.

```
aboy.drawPixel(previouspoint.x, previouspoint.y, !(aboy.getPixel(previouspoint.x, previouspoint.y)));
```

The first draw function sets the last pixel we were on back to its original colour, it achieves this through the getPixel command. Using the previouspoint.x and previouspoint.y variables, we can locate the last pixel and then use the logical NOT (!) to inverse the result. We do this to cancel out the change we did.

```
aboy.drawPixel(Controller.position.x, Controller.position.y, !(aboy.getPixel(Controller.position.x, Controller.position.y)));
```

The change was when we drew the cursor, as we drew the cursor, we overwrote the pixel by inverting its colour – this is achieved by also using the getPixel command, to get the colour of the current pixel and to use the logical NOT (!) to inverse the result and then overwrite that

value. You can see we conduct this change on the current pixel as we use the Controller.position.x and Controller.position.y variables.

To ensure that we don't change the actual pixels/picture, whatever changes we make, we need to change them back. This creates a cursor, as we can move around a pixel until the drawstate is true again.

To control the drawstate, we need to utilize one of the buttons to let the user toggle between drawing and not drawing.

```
  if (Controller.BButtonPressed == true) {
    drawstate = !drawstate;
    //As draw state is a bool, we can use ! to inverse the current value - 1 turns into 0 and  0 turns into 1
    digitalWrite(9, drawstate);
    // Turn on/off LED depending on state of draw.
  }
```

Here we utilize the B button, if the B button was pressed, then invert the drawstate (we set drawstate to true initially, therefore the user begins drawing and when they press the button, the drawstate is inverted using logical NOT (!). To indicate if the user is drawing or not, we display the drawstate status through the Blue LED on the Arduino. As drawstate is either 0 or 1, this is the same as writing either HIGH or LOW here.

```
  aboy.display();
  // set the current x and y position of the point to the previous variables to use on the next loop run.
  previouspoint.x = Controller.position.x;
  previouspoint.y = Controller.position.y;
}
```

We then update the display and update the previous point positions, ready for the next loop.

```
#include <Arduboy2.h>
Arduboy2 aboy;

typedef struct point {
  int x;
  int y;
};

#include "controller.h"
Controller Controller;
bool colour = true;
bool drawstate = true;
point previouspoint;

void setup() {
  aboy.boot();
  aboy.clear();
  Serial.begin(9600);

  pinMode(10, OUTPUT); //Yellow
  pinMode(3, OUTPUT); //Green
  pinMode(9, OUTPUT); //Red

  Controller.position.y = aboy.height() / 2;
  Controller.position.x = aboy.width() / 2;
}

void loop() {
  Controller.update();

  if (drawstate) {
    aboy.drawPixel(Controller.position.x, Controller.position.y, colour);
  }
  else {
    aboy.drawPixel(previouspoint.x, previouspoint.y, !(aboy.getPixel(previouspoint.x, previouspoint.y)));
    aboy.drawPixel(Controller.position.x, Controller.position.y, !(aboy.getPixel(Controller.position.x, Controller.position.y)));
  }

  if (Controller.BButtonPressed == true) {
    drawstate = !drawstate;
    digitalWrite(9, drawstate);
  }
  aboy.display();
  previouspoint.x = Controller.position.x;
  previouspoint.y = Controller.position.y;
}
```

Please ensure your current file looks like the one above.

Colour

Next, we are going to add the ability to change the colour. While this is a black and white screen, we still need to allow for the user to switch colours – this also acts as a rubber if the player draws the wrong pixel and wants to remove it. Toggle this, we will use button A.

Your task is to allow the user to toggle the colour of the pen (using the bool variable colour) with button A. Display when the user is painting with black colour by turning on the RED LED (Pin 10). The below functions could be useful:

*To access button A, simply type **Controller.AButtonPressed** and this will return the state of the button.*

***digitalWrite(Pin, State)** = See above – the Red LED is on Pin 10.*

Please copy the code below to ensure you have the correct program – place this in the void loop.

```
39  //A BUTTON: Changes Colour
40  if (Controller.AButtonPressed == true) {
41    colour = !colour;
42    digitalWrite(10, colour);
43  }
```

Here we check if the button has been pressed, if it has then we simply inverse the colour by using the logical NOT (!). We then display the colour state by writing the boolean value to the LED. As the LEDs have been internally pulled up, we can simply pass through the colour boolean variable to the LED – meaning when it is 1 or white, the LED will be off. Therefore when the value is 0 or black, the LED is ON, meaning the user is alerted that they are painting in black.

Adding a Reset Option

The final mechanic of the game is the option to clear the whole screen and reset. While we can achieve this by using the RESET button on the microcontroller, we can simply use aboy.clear() to clear the screen instead, meaning the user doesn't have to load the game again.

Your task is to write a simple function that writes to the screen "RESET" and then clears the screen to black then sets the users cursor back to the centre. Only reset when the user presses BOTH A and B buttons. The below functions could be useful:

To create a function with no return, use the term void FunctionName(). If the function needs to return a value, then begin the function declaration with the data type of the value being returned, aka if we wanted to return the value 55, we would use int FunctionName().

***Aboy.clear();** = clears everything on the screen.*

***Aboy.setCursor(Xpos, Ypos);** Sets the position of the cursor to print text. E.g setCursor(5,0); and then aboy.print("Hello World"); would print out the words Hello World at x position 5 and y position 0 on the screen.*

***Aboy.print("This text will print");** This is just like Serial.print(); and will print to the screen – remember to put an aboy.display() after printing to show it on the screen!*

***delay(TimeInMiliSeconds);** stops the program and waits for the specified amount of time. E.g. delay(1000); would stop and wait for 1 second or 1000 milliseconds*

Please copy the code below to ensure you have the correct program – place this in the void loop() and the function before the void setup()

```
62  if (aboy.pressed(A_BUTTON) == true and aboy.pressed(B_BUTTON) == true) {
63    reset();
64  }
```

Here, if both buttons are pressed, then we run the function below

```
15 void reset() {
16   aboy.clear();
17   aboy.setCursor(49, 27);
18   aboy.print("RESET");
19   aboy.display();
20   delay(1000);
21   aboy.clear();
22   Controller.position.y = aboy.height() / 2;
23   Controller.position.x = aboy.width() / 2;
24 }
```

Here, we clear the screen and set the cursor to the correct position to print the word RESET. We display the word and then wait 1 second. We then clear the screen again and set the cursor to the centre by overwriting the Controller.position.y and x values to the centre.

If you are ever not sure about where to put a section of code, check the final code section at the back of the book.

Adding a Basic GUI

Now that we have the core mechanisms of the game down, we can start creating a basic GUI. Currently, we tell the user what colour they are using through an LED, let's also display on the screen, the first letter of the colour they are using in a box coloured the same colour. Aka, we would draw a filled box of white with the letter W inside of it and an empty box with the letter B inside – to indicate Black and White. Let's also add a white frame around the outskirts of the screen, so the user knows the boundary (we will add the boundary afterwards).

Your task is to create a GUI function called frame() that we can call to draw the GUI on currently we need to draw a white frame around the outskirts of the screen. The below functions could be useful:

aboy.drawRect(StartXpos, StartYpos, EndXpos, EndYpos, Colour); *= when drawing an empty rectangle, no fill, we must specify the coordinates of the first point (top left of the rectangle) and the second point (bottom right). We also need to specify the border colour, either "WHITE" or "BLACK".*

```
43 void loop() {
44   Controller.update();
45   frame();
46
47   if (drawstate) {
```

Firstly we call the function in the loop, after the Controller.update();

```
26 void frame() {
27   aboy.drawRect(0, 0, 128, 64, WHITE);
28 }
```

This then links to the below function, here we will fill in the rest of the GUI, but for now you should have your rectangle!

Next, we are going to add the Colour indicator. To do this, we write inside of the frame(); function

---

Your task is to write the Colour indicator when the user is drawing with a white pencil display a 'W' inside of a white box. When the user is drawing with a black pen, display a 'B' inside of a black box. The below functions might come in handy:

Aboy.fillRect(StartXpos, StartYpos, EndXpos, EndYpos, Colour); = *Is the same format as the drawRect but will fill the rectangle with the same colour as the border.*

Aboy.drawChar(Xpos, Ypos, Char, TextColour, BackgroundColour, TextSize); *= Will draw the "char", e.g. the letter W, at the Xpos and Ypos (meaning you don't need to set the cursor position beforehand) at the TextSize of the Text Colour, (meaning you don't need to set the text size or colour beforehand).*

Your new frame function should look like this:

```
26 void frame() {
27   aboy.drawRect(0, 0, 128, 64, WHITE);
28
29   if (!colour) {
30     aboy.fillRect(98, 5, 11, 11, BLACK);// fill
31     aboy.drawRect(98, 5, 11, 11, WHITE); // outline
32     aboy.drawChar(101, 7, 'B', WHITE, BLACK, 1);
33   }
34   else {
35     aboy.fillRect(98, 5, 11, 11, WHITE);
36     aboy.drawChar(101, 7, 'W', BLACK, WHITE, 1);
37   }
38 }
```

Be sure to use the values above if you want the program to look exactly like the one we have (and to save you the math of positioning the letter inside of the box). Remember that one character is 6 pixels wide and 8 pixels in high. Here, if the colour is black, we invert the result meaning the value turns into 1, as 1 is the same as true the if statement runs and therefore runs the code to fill the rectangle black, draw a white border around the rectangle, and then draw the character in the middle. If the colour is white, the value is inverted and the if statement does not run, but rather the else statement runs, filling the rectangle white and displaying the character with black front and white background – to contrast.

Your task is to write the Draw indicator. When the user is drawing, display a D in a white box. However, when the user toggles this, switch to an X in a black box, to demonstrate that they are not drawing. The functions from beforehand will come in handy

---

```
26 void frame() {
27   aboy.drawRect(0, 0, 128, 64, WHITE);
28
29   if (!colour) {
30     aboy.fillRect(98, 5, 11, 11, BLACK);// fill
31     aboy.drawRect(98, 5, 11, 11, WHITE); // outline
32     aboy.drawChar(101, 7, 'B', WHITE, BLACK, 1);
33   }
34   else {
35     aboy.fillRect(98, 5, 11, 11, WHITE);
36     aboy.drawChar(101, 7, 'W', BLACK, WHITE, 1);
37   }
38
39   if (!drawstate) {
40     aboy.fillRect(113, 5, 11, 11, BLACK);// fill
41     aboy.drawRect(113, 5, 11, 11, WHITE); // outline
42     aboy.drawChar(116, 7, 'X', WHITE, BLACK, 1);
43   }
44   else {
45     aboy.fillRect(113, 5, 11, 11, WHITE);
46     aboy.drawChar(116, 7, 'D', BLACK, WHITE, 1);
47   }
48 }
```

Here you can see a very similar code however the C coordinate has changed – be sure to type in the values to get the two boxes equally spaced in the top right corner.

Note that the reason we have to draw a black rectangle and then a border is that if we just drew a border, you would still see the white box from beforehand as we do not utilize aboy.clear() in this code – so we overwrite the white box with a black box, then draw a border.

Adding Boundaries

However, the current code means that we can move the controller.position. x value of the screen, causing the player to draw off of the screen.

Your task is to write code that will restrict the player from exceeding controller.positon.x between 1 and 127. The functions below might be useful:

Void [FunctionName] (int ArgOne, int ArgTwo){ [Code] } = Allows us to create a function with parameters, in this case, we created a function that has no returns (as void means there is no return, therefore, no need to specify a data type). Here we have two parameters that must be met upon using this function.

Controller.LEFTButtonCanBePress; = is a Boolean variable we defined earlier that switches the movement in a certain direction on/off. Set this to false to stop the program from reading and updating any inputs from this button

---

Adding an X Boundary/Collision

For this program, we have two boundaries, the X and the Y – to ensure the user can't draw off of the screen.

```
232 void Check_Range_X(int lowrange, int highrange)
233 {
234   if (Controller.position.x <= lowrange) {
235     Controller.LEFTButtonCanBePress = 0;
236     Controller.RIGHTButtonCanBePress = 1;
237   }
238   else if (Controller.position.x >= highrange) {
239     Controller.LEFTButtonCanBePress = 1;
240     Controller.RIGHTButtonCanBePress = 0;
241   }
242   else {
243     Controller.LEFTButtonCanBePress = 1;
244     Controller.RIGHTButtonCanBePress = 1;
245   }
246 }
```

Hopefully, your code looks similar to this format, if not then type in the above code. It should be placed after the void loop.

Here you can see, when we define the function we require two numbers, this help makes the function dynamic and be able to be used in many different situations. Here we require that we get the range parameters – aka lowest range and highest range. This is then used in the if statement.

If the current position is lower than the range, then we turn off movement going even lower (left button) as we want the player to stop moving, stopping the movement also stops the player if the player is holding down the right button and moving into a boundary, this function checks it and therefore stops ALL movement. We only allow movement on the X-axis to be done via the right button, aka to get back in range. The next statement uses a similar format, however, it's the higher range, therefore, we turn off the movement for the right button, and only allow the position to be updated when the user moves back into range (aka only allow the user to press the left button).

```
97  void loop() {
98    Check_Range_X(1, 127);
99    Controller.update();
100   frame();
```

Be sure to write this section of code inside the void loop!

Here you can see we run it in the main loop before we update the position values (aka before we check the button controls). This is so we can check if the player is in range, and therefore only allow the correct buttons to be read (I say read because the user can still press the button, however as we set the "**ButtonCanBePress**" value to false/0, we choose to ignore it regardless.

This ensures that the user can never exceed the values between 1 and 127.

Adding a Y Boundary/Collision

Now, we can use the above code to help us write collision detection for the Y-axis.

Your task is to write code that will restrict the player from exceeding controller.positon.y between 1 and 127. The functions from before might be useful.

```
43 // BOUNDARY CHECK
44 void Check_Range_Y( int lowrange, int highrange)
45 {
46   if (Controller.position.y <= lowrange) {
47     Controller.UPButtonCanBePress = 0;
48     Controller.DOWNButtonCanBePress = 1;
49   }
97  void loop() {                    .y >= highrange) {
98    Check_Range_X(1, 127);         ess = 1;
99    Controller.update();           Press = 0;
100   frame();
55     Controller.UPButtonCanBePress = 1;
56     Controller.DOWNButtonCanBePress = 1;
57   }
58 }
```

Here we have a similar algorithm but using the position.y variables to alter the correct axis. If we go off-screen, it will simply stop us as its ignoring the inputs from that button.

Be sure to copy this section of code and the values used.

```
97 void loop() {
98   Check_Range_X(1, 127);
99   Controller.update();
100  frame();
```

Be sure to write this section of code inside the void loop!

Adding a SplashScreen

Now, let's add the finishing touches, a splash screen!

Your task is to write a function that will display a splash screen in the void setup – it should say "8Bit-Etch" but you can always customize it with your name! The below functions might be useful:

Aboy.setTextSize(1); = whatever the value you put in, the standard 6x8 will be multiplied. E.g. a value of 2 would double the size meaning it would be 12x16 pixels.

```
84 void splashscreen() {
85
86   aboy.setCursor(10, 8);
87   aboy.setTextSize(2);
88   aboy.print("8Bit-Etch");
89   aboy.display();
90   delay(4000);
91   aboy.clear();
92   frame();
93 }
```

Here we have some simple code that displays some text for the splash screen – feel free to experiment with the splash screen by adding your name or even drawing something!

Be sure to copy this section of code in before the void setup.

We then call the function splash screen() in the void setup

Final Code

Congratulations on completing this course! Now you have an Etch-A-Sketch on your 8BitCADE. Now is your chance to play around with it and see what you can add. Why not customize the splash screen, add your name or see what else you can do! The world is yours!

8Bit-Etch-A-Sketch.ino

```
1.  //
2.  //  BSD 3-Clause License
3.  //
4.  //  Copyright (c) 2020, Jack Daly (@8bitcade)
5.  //  All rights reserved.
6.  //
7.  //  Redistribution and use in source and binary forms, with or without
8.  //  modification, are permitted provided that the following conditions are met:
9.  //
10. //  1. Redistributions of source code must retain the above copyright notice, this
11. //     list of conditions and the following disclaimer.
12. //
13. //  2. Redistributions in binary form must reproduce the above copyright notice,
14. //     this list of conditions and the following disclaimer in the documentation
15. //     and/or other materials provided with the distribution.
```

```
16. //
17. //  3. Neither the name of the copyright holder nor the names of its
18. //     contributors may be used to endorse or promote products derived from
19. //     this software without specific prior written permission.
20. //
21. //  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
22. //  AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
23. //  IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
24. //  DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE
25. //  FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
26. //  DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
27. //  SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
28. //  CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
29. //  OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
30. //  OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
31. //
32. #include <Arduboy2.h>
33. Arduboy2 aboy;
34.
35. typedef struct point {
36.   int x;
37.   int y;
38. };
39.
40. #include "controller.h"
41. Controller Controller;
42. bool colour = true;
43. bool drawstate = true;
44. point previouspoint;
45.
46. void reset() {
47.   aboy.clear();
48.   aboy.setCursor(49, 27);
49.   aboy.print("RESET");
50.   aboy.display();
51.   delay(1000);
52.   aboy.clear();
53.   Controller.position.y = aboy.height() / 2;
54.   Controller.position.x = aboy.width() / 2;
55. }
56.
57. // BOUNDARY CHECK
58. void Check_Range_Y( int lowrange, int highrange)
59. {
60.   if (Controller.position.y <= lowrange) {
61.     Controller.UPButtonCanBePress = 0;
62.     Controller.DOWNButtonCanBePress = 1;
63.   }
64.   else if (Controller.position.y >= highrange) {
65.     Controller.UPButtonCanBePress = 1;
66.     Controller.DOWNButtonCanBePress = 0;
67.   }
68.   else {
69.     Controller.UPButtonCanBePress = 1;
70.     Controller.DOWNButtonCanBePress = 1;
71.   }
72. }
73.
74. void Check_Range_X(int lowrange, int highrange)
75. {
76.   if (Controller.position.x <= lowrange) {
77.     Controller.LEFTButtonCanBePress = 0;
78.     Controller.RIGHTButtonCanBePress = 1;
79.   }
80.   else if (Controller.position.x >= highrange) {
81.     Controller.LEFTButtonCanBePress = 1;
```

```
    Controller.RIGHTButtonCanBePress = 0;
  }
  else {
    Controller.LEFTButtonCanBePress = 1;
    Controller.RIGHTButtonCanBePress = 1;
  }
}


void frame() {
  aboy.drawRect(0, 0, 128, 64, WHITE);

  if (!colour) {
    aboy.fillRect(98, 5, 11, 11, BLACK);// fill
    aboy.drawRect(98, 5, 11, 11, WHITE); // outline
    aboy.drawChar(101, 7, 'B', WHITE, BLACK, 1);
  }
  else {
          aboy.fillRect(98, 5, 11, 11, WHITE);
          aboy.drawChar(101, 7, 'W', BLACK, WHITE, 1);
        }

        if (!drawstate) {
          aboy.fillRect(113, 5, 11, 11, BLACK);// fill
          aboy.drawRect(113, 5, 11, 11, WHITE); // outline
          aboy.drawChar(116, 7, 'X', WHITE, BLACK, 1);
        }
        else {
          aboy.fillRect(113, 5, 11, 11, WHITE);
          aboy.drawChar(116, 7, 'D', BLACK, WHITE, 1);
        }
      }

      void splashscreen() {

        aboy.setCursor(10, 8);
        aboy.setTextSize(2);
        aboy.print("8Bit-Etch");
        aboy.display();
        delay(4000);
        aboy.clear();
        frame();
      }

      void setup() {
        aboy.boot();
        aboy.clear();
        Serial.begin(9600);

        pinMode(10, OUTPUT); //Yellow
        pinMode(3, OUTPUT); //Green
        pinMode(9, OUTPUT); //Red

        Controller.position.y = aboy.height() / 2;
        Controller.position.x = aboy.width() / 2;

        splashscreen();
      }

      void loop() {
        Check_Range_X(1, 127);
        Check_Range_Y(2, 62);
        Controller.update();

        frame();

```

```
148.          if (drawstate) {
149.            aboy.drawPixel(Controller.position.x, Controller.position.y, colour);
150.          }
151.          else {
152.            aboy.drawPixel(previouspoint.x, previouspoint.y, !(aboy.getPixel(previou
    spoint.x, previouspoint.y)));
153.            aboy.drawPixel(Controller.position.x, Controller.position.y, !(aboy.getP
    ixel(Controller.position.x, Controller.position.y)));
154.          }
155.
156.          //A BUTTON: Changes Colour
157.          if (Controller.AButtonPressed == true) {
158.            colour = !colour;
159.            digitalWrite(10, colour);
160.          }
161.
162.          if (Controller.BButtonPressed == true) {
163.            drawstate = !drawstate;
164.            digitalWrite(9, drawstate);
165.          }
166.
167.          if (Controller.AButtonPressed == true and Controller.BButtonPressed == tru
    e) {
168.            reset();
169.          }
170.
171.          aboy.display();
172.          previouspoint.x = Controller.position.x;
173.          previouspoint.y = Controller.position.y;
174.        }
```

Controller.h

```
1.  class Controller
2.  {
3.    public:
4.      Controller();
5.      point position;
6.      void update();
7.      bool AButtonPressed; //BOOLEAN VARIABLE THAT STORES WHETHER BUTTON PRESSED IS A

8.      bool BButtonPressed; //BOOLEAN VARIABLE THAT STORES WHETHER BUTTON PRESSED IS B

9.
10.     bool previousUPButtonPressed;
11.     bool previousDOWNButtonPressed;
12.     bool previousLEFTButtonPressed;
13.     bool previousRIGHTButtonPressed;
14.     bool previousAButtonPressed;
15.     bool previousBButtonPressed;
16.
17.     bool UPButtonCanBePress = 1;
18.     bool DOWNButtonCanBePress = 1;
19.     bool LEFTButtonCanBePress = 1;
20.     bool RIGHTButtonCanBePress = 1;
21.
22.
23.     bool AButtonWasPressed; //BOOLEAN VARIABLES THAT CHECKS IF THE BUTTON PRESS IS
    UPDATED (A PRESSED)
24.     bool BButtonWasPressed; //BOOLEAN VARIABLES THAT CHECKS IF THE BUTTON PRESS IS
    UPDATED (B PRESSED)
25.     boolean BButtonCanBePress = 1; //HOLDS CURRENT BUTTON STATE FOR B
26.     boolean AButtonCanBePress = 1; //HOLDS CURRENT BUTTON STATE FOR A
```

```
    void Check_Range(); //CHECKS POSITION IS IN RANGE COMPARED TO TWO ARGUEMENTS
    int pos; //HOLDS POSITION OF 'CURSOR' OR CHARACTER

    long lastDebounceTime = 0;  // the last time the output pin was toggled
    long debounceDelay = 75;
    long currenttime = 0;
    long A_lastHoldTime = 100;
    long B_lastHoldTime = 100;
    long lastHoldTime = 100;

};
```

Controller.ino

```
Controller::Controller()
{
  Serial.print("BUTTON TEST BUTTON TEST");
  position.x = 0;
  position.y = 0;
}

void Controller::update()
{
  if (aboy.pressed(UP_BUTTON) and previousUPButtonPressed == 1 and (millis() - lastHoldTime) > 150 and UPButtonCanBePress) {
    lastHoldTime = millis();
    position.y -= 1;
  }
  if ((aboy.pressed(DOWN_BUTTON) == 1 and previousDOWNButtonPressed == 1 and (millis() - lastHoldTime) > 150) and DOWNButtonCanBePress) {
    lastHoldTime = millis();
    position.y += 1;
  }
  if (aboy.pressed(LEFT_BUTTON) and previousLEFTButtonPressed == 1 and (millis() - lastHoldTime) > 150 and LEFTButtonCanBePress) {
    lastHoldTime = millis();
    position.x -= 1;
  }
  if ((aboy.pressed(RIGHT_BUTTON) == 1 and previousRIGHTButtonPressed == 1 and (millis() - lastHoldTime) > 150) and RIGHTButtonCanBePress) {
    lastHoldTime = millis();
    position.x += 1;
  }
  if (aboy.pressed(A_BUTTON) and previousAButtonPressed == 1 and (millis() - A_lastHoldTime) > 250) {
    A_lastHoldTime = millis();
    AButtonPressed = true;
  }
  else {
    AButtonPressed = false;
  }
  if ((aboy.pressed(B_BUTTON) == 1 and previousBButtonPressed == 1 and (millis() - B_lastHoldTime) > 250) ) {
    B_lastHoldTime = millis();
    BButtonPressed = true;
  }
  else {
    BButtonPressed = false;
  }
  previousUPButtonPressed = aboy.pressed(UP_BUTTON);
  previousDOWNButtonPressed = aboy.pressed(DOWN_BUTTON);
  previousLEFTButtonPressed = aboy.pressed(LEFT_BUTTON);
  previousRIGHTButtonPressed = aboy.pressed(RIGHT_BUTTON);
```

```
44.   previousAButtonPressed = aboy.pressed(A_BUTTON);
45.   previousBButtonPressed = aboy.pressed(B_BUTTON);
46. }
```

Thank you for following along with this tutorial. If you have any programming questions, we strongly advise that you check out the Arduboy community. If you have any 8BitCADE related issues, please email us at 8BitCADE@support.com.

Check out our other tutorials at 8bitcade.com/learn