



# Make Your Own Arduboy Game

Learn to Program in C++ &  
Create a Game in 9 Easy Lessons

by Jonathan Holmes

[twitter.com/crait](https://twitter.com/crait)

1. <a href="#">Setting up your Computer</a> .....	1
2. <a href="#">Printing Text</a> .....	9
3. <a href="#">Storing Variables</a> .....	13
4. <a href="#">Questions &amp; Button Input</a> .....	18
5. <a href="#">Your First Game</a> .....	24
6. <a href="#">Graphics</a> .....	44
7. <a href="#">Make Pong from Scratch</a> .....	58
8. <a href="#">Starting Dinomasher</a> .....	112
9. <a href="#">Mapping Dinomasher</a> .....	139

# Part 1: Setting up your Computer

## The Arduino IDE & Arduboy Library

So, in order to program a game for the Arduboy, you need to write your code in a program called the Arduino IDE. An IDE is an Integrated Development Environment. This means that it includes most of the tools you'll need to program for your Arduboy. Arduino is the type of computer chip that the Arduboy is made from. All Arduino chips can be programmed the same way that we're going to program your Arduboy, so you'll learn how to program for several devices!

## Downloading The IDE

Okay, to download the latest version of the Arduino IDE, you should follow [this link](#) .

Alternatively, you can directly download it here:

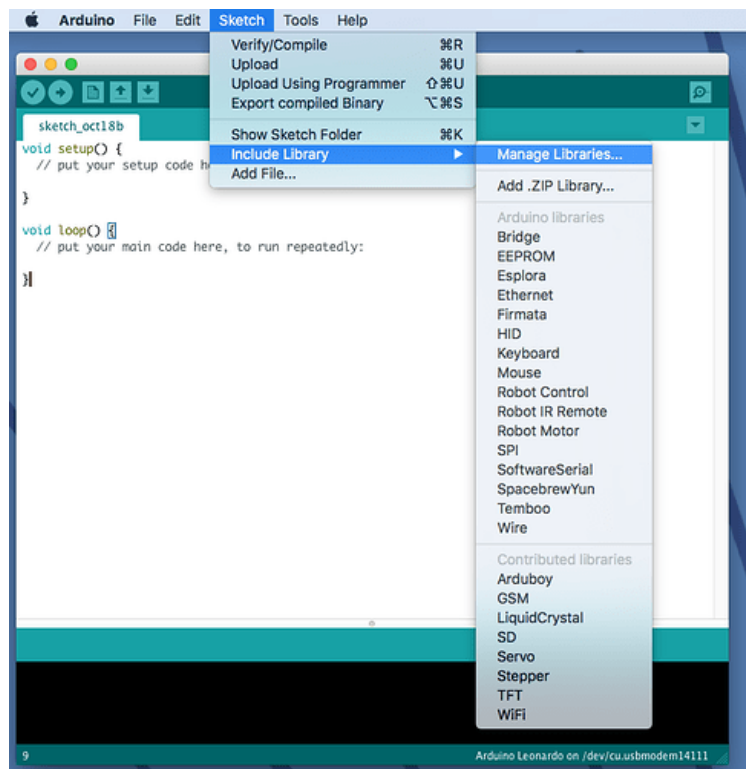
- [Windows](#)
- [macOS](#)
- [Linux \(32-Bit\)](#)
- [Linux \(64-Bit\)](#)

## Installing The IDE

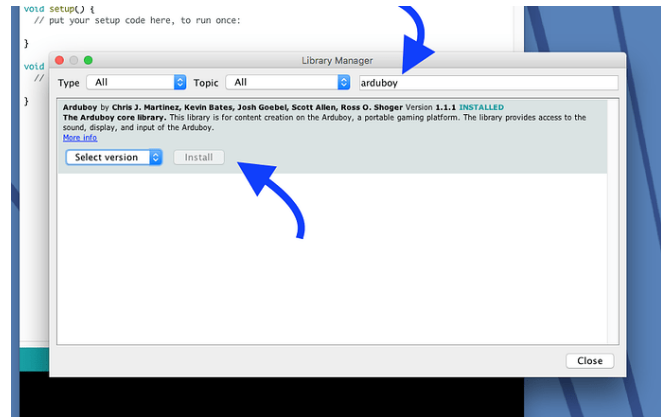
Install the Arduino IDE like you would any other program.

## Installing the Arduboy Library

Not only do you need the Arduino IDE, but you'll also need the Arduboy Library. You need that in order to tell your IDE how to interact with the Arduboy. To get it, go to Sketch > Include Library > Manage Libraries...

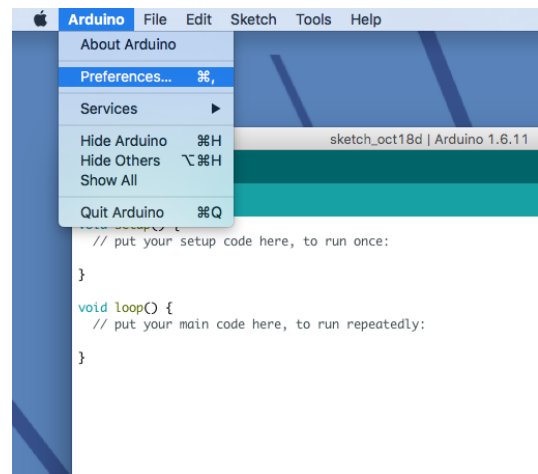


In the Manage Libraries window, search for “Arduboy”. One search result should appear. Select the most recent version and hit **Install**.



## Installing the Board

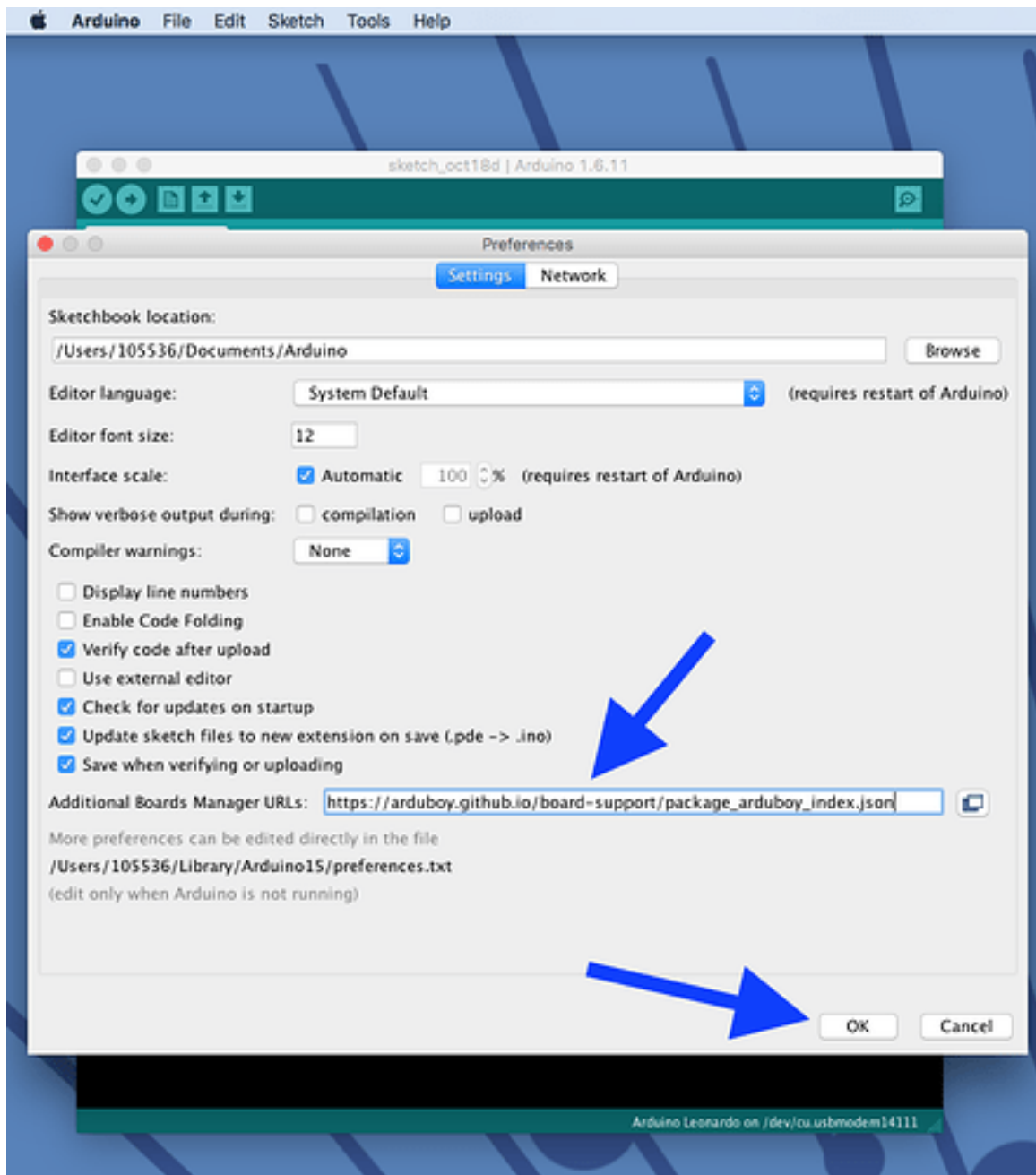
Go to **File > Preferences** on Windows or **Arduino > Preferences** on macOS. In the Settings window, find the **Additional Boards Manger URLs** box.



Insert the following:

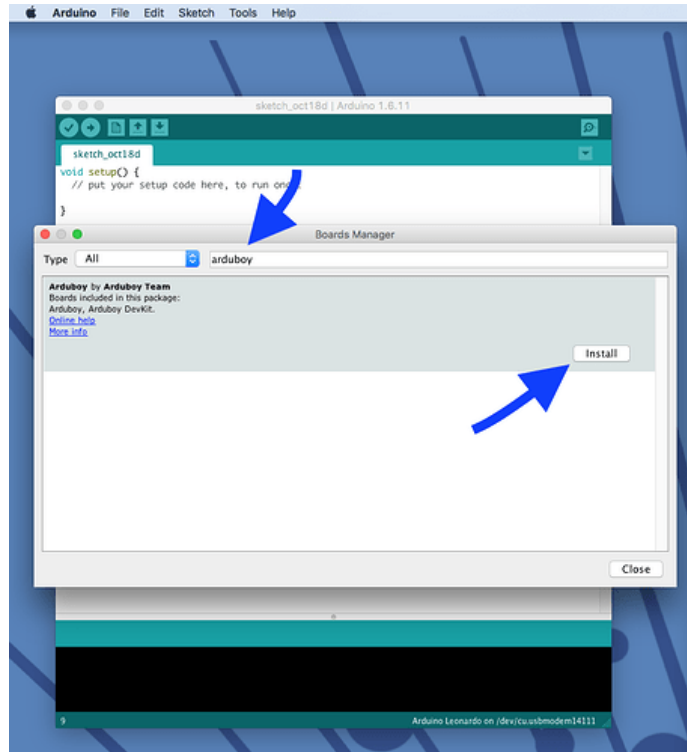
[https://arduboy.github.io/board-support/package\\_arduboy\\_index.json](https://arduboy.github.io/board-support/package_arduboy_index.json)

and click **OK**.



Next, go to **Tools > Board > Board Manager**. In the window that appears, search for “Arduboy”. The list should be narrowed down to the Arduboy board. Click it, then click the **Install** button.

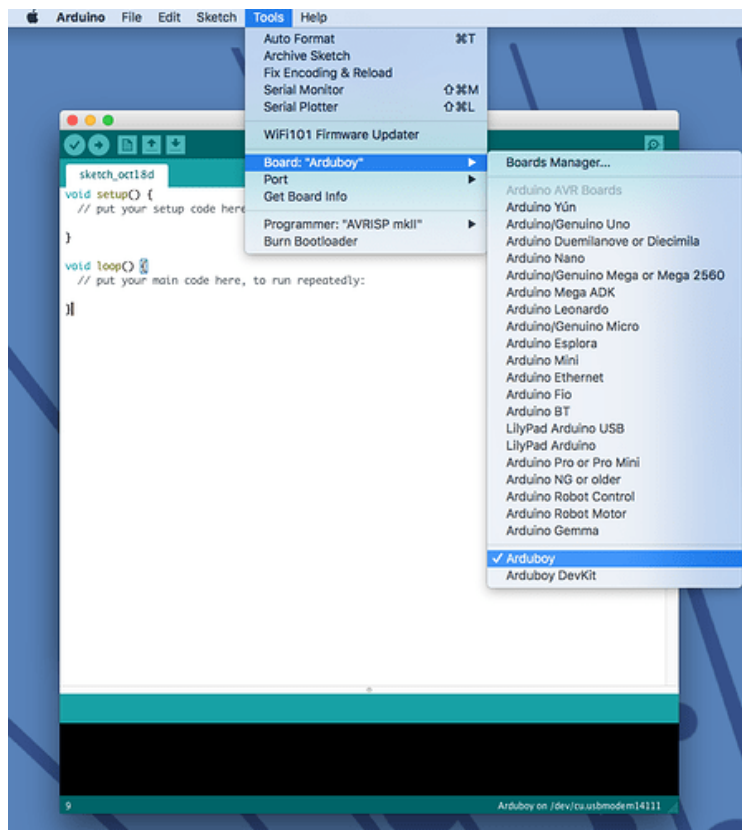
Once it’s done installing, click the **Close** button.



## Selecting the Board

In order to tell the IDE that you want to program for the Arduboy, you need to tell it that you’re going to work with the Arduboy, go to **Tools > Board > Arduboy**.

Please note that if you’re using a discontinued Arduboy model, you may want to check the Install Support For The Arduboy & DevKit section of [this guide](#). Most people will not have to do this.

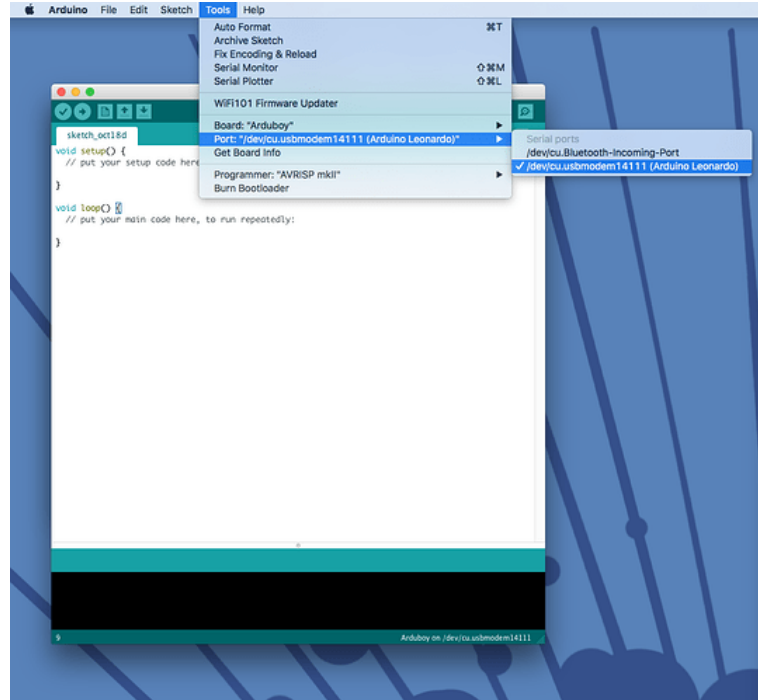


## Selecting the Port

Turn on your Arduboy and connect it with a USB data cable. Then, go to **Tools > Port**, then select the Arduino Leonardo.

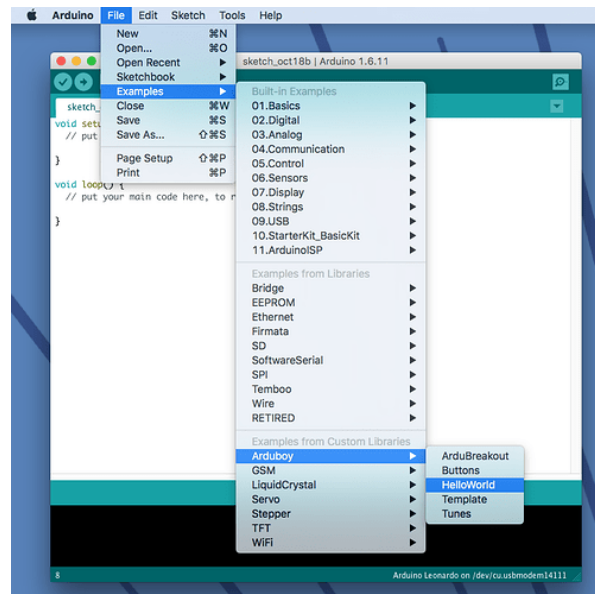
If you are running Windows, the port may be something like COM5.

If you're on macOS or Linux, it may be something like `/dev/cu.usbmodem1411`.



## Loading an Example

Alrighty! You should be ready to test this out! Let's load an example program that we got in the Arduboy Library. Goto **File > Examples > Arduboy > HelloWorld**. This will load a test application that prints out a single line of text to the Arduboy's screen.



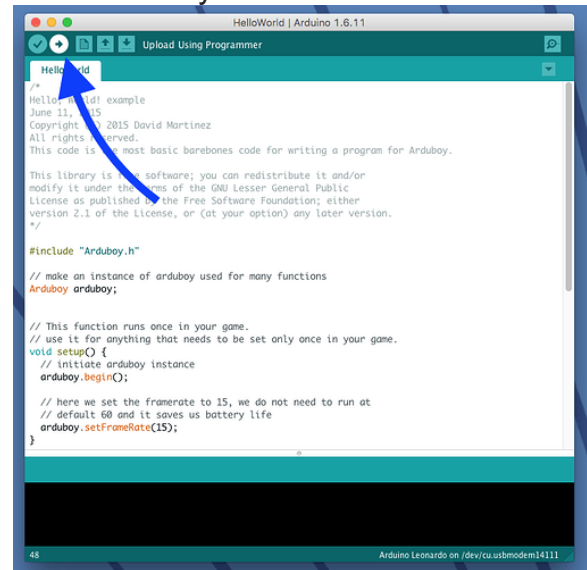
# Uploading the Code

Whenever you're ready to install the code, click the arrow key that looks like this:

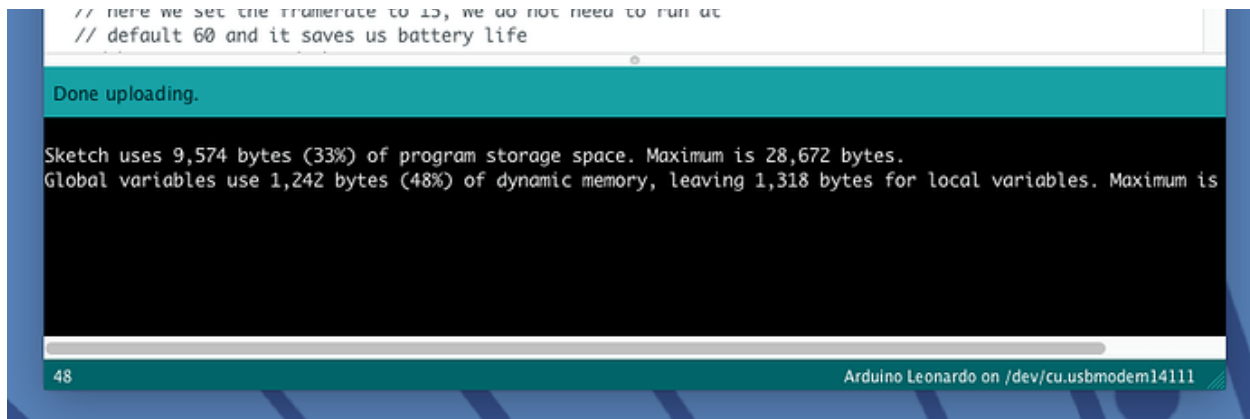


This will do 3 tasks:

1. Check your code for common problems or typos.
2. Compile your code. This is when the IDE turns your programming language into computer language that the Arduboy can understand.
3. Transfer your code your code to the Arduboy.



If the IDE sees any problems with your code or has an issue when compiling your game, it will alert you in the console window, which is the black area at the bottom. If you don't run into any problems, then it will tell you how big your game is.



# Look at Your Arduboy!

If you've followed these steps correctly, you should see your Arduboy displaying "Hello World!"





# Part 2: Printing Text

## What Is Programming?

Because of how hacking and programming are portrayed in movies and stuff, many people have no idea how programming works. Basically, programs are a big list of instructions for the computer to follow. Computers do exactly what they are told. They have no opinions nor intuition, so computers can't just decide or guess what to do next. You must be very specific.

Different programming languages have different ways that they're written and different kinds of instructions. These tutorials will go over C++ basics and you'll learn about those kinds of instructions.

## Creating A New Sketch

Using the [Arduino IDE](#), go to **File > New**. This will create a new window with a new **sketch**. A *sketch* is a file with your code in it. Sometimes you'll hear it called "source code".

## Reviewing Our Code

In that new window, you should see the following code:

```
void setup() {  
    // put your setup code here, to run once:  
}  
void loop() {  
    // put your main code here, to run repeatedly:  
}
```

Whenever you write code, you're going to put a lot of it inside of braces, which look like { and }. That is to split up different lines of code that occur at different times. You'll learn more specifics about what those different times are later on. All you need to worry about is that you have some code that will be put into the **setup()** section and the **loop()** section. These are called **functions**, but we don't need to know much about them right now. All we need to know is that whatever code that's inside of the **setup()** function will be run once whenever the Arduboy turns on and whatever code is inside the **loop()** function will be run later on.

## Comments

The next thing I want to bring to your attention are the **comments**. These are lines of code that are completely ignored. They're great for adding in notes so that you can read your code easier. I'm going to use them a lot, so I thought we should start with them.

It's very typical to use comments to explain the purpose of functions, variables, and at the beginning of your code to explain the purpose of your program/game.

Let's go ahead and add some comments to the top of the program! At the top, add a few new lines of text that starts with two slashes, then your name, date, and the title of this program.

Your code should now look like this:

```
//Jonathan Holmes (crait)
//October 18th, 2016
//Printing Text
void setup() {
    // put your setup code here, to run once:
}
void loop() {
    // put your main code here, to run repeatedly:
}
```

Those lines of text should turn grey.

## Getting Arduboy-Specific Instructions

Okay, that was simple enough, but we need to start adding in our instructions, now. To add in a lot of Arduboy-specific instructions, we need to import the Arduboy Library into our code. Make a new line of code after your comments and add `#include <Arduboy.h>`. This line of code says that you want to grab a lot of instructions from the *Arduboy.h* file. There's nothing more about that that you need to know.

Make a new line after that, and add `Arduboy arduboy;`. Don't worry about this too much. Like I said, this is part of getting Arduboy-Specific instructions.

Our code should look like this:

```
//Jonathan Holmes (crait)
//October 18th, 2016
//Printing Text

#include <Arduboy.h>
Arduboy arduboy;

void setup() {
    // put your setup code here, to run once:
}

void loop() {
    // put your main code here, to run repeatedly:
}
```

## The setup() Function

Sweet! I'm glad you got this far. Add the following lines of code inside of the braces of the **setup()** function:

```
arduboy.begin();
arduboy.clear();
arduboy.print("Hello");
arduboy.display();
```

Notice that each line of code ends in a semi-colon ; . This is because all instructions in C++ need to end in one. Otherwise, the computer will think it's something else that I'll explain later.

Next, notice that all of these lines of code include `arduboy.` at the beginning. This is because our instructions are stored inside of the **arduboy** library.

Last thing I want you to look at is the parenthesis, ( and ) . All functions have these. Some include things inside of them and some do not.

The `arduboy.begin();` function pretty much tells the Arduboy to turn on properly. Once that is done, the next instruction is followed.

`arduboy.clear();` tells the Arduboy to erase everything on the screen.

`arduboy.print("Hello");` tells the Arduboy to write some text to the screen. Notice that we actually have something inside of the ( and ) this time. To tell the Arduboy *what* we want to print, we need to put it inside of the parenthesis. Any time you're working with text, you have to

put quotation marks around the text, so we gotta do that, too. You can actually put whatever you want inside of those quotations. Maybe play around with it and put your own name into it.

Last, we have to tell the Arduboy to actually refresh the screen and show you what you just printed to it. That's what the `arduboy.display();` does.

## You're done!

That's it! You've programmed your first program for the Arduboy! This is what the finished code should look like:

```
//Jonathan Holmes (crait)  
//October 18th, 2016  
//Printing Text  
  
#include <Arduboy.h>  
Arduboy arduboy;  
  
void setup() {  
  // put your setup code here, to run once:  
  arduboy.begin();  
  arduboy.clear();  
  arduboy.print("Holmes is cool!");  
  arduboy.display();  
}  
  
void loop() {  
  // put your main code here, to run repeatedly:  
}
```

## Installing

Like we did in [the previous tutorial](#), connect your Arduboy with a USB data cable, select the correct port and board, then click the button to transfer it to your Arduboy!



# Part 3: Storing Variables

## Variables

Computers work a lot with calculations and data. To make most video games, you're going to need to be able to store data, like high scores, or player location, or lives remaining. In order to remember data, a computer must set aside some memory to put that data into. Then, the computer can be told to change the data stored there.

## Loops

Remember how I said that computers have to be given specific instructions? Ever notice how the back of shampoo bottles say to **1. Lather, 2. Rinse, 3. Repeat?** If a computer were given those instructions, they would be stuck in an infinite loop of lathering, rinsing, and repeating. In programming, having instructions repeat in a loop can be very useful. I'll show you how to use the **loop()** function to do this.



## Getting Started

In this program, we're going to make the Arduboy keep track of a number and display it to you as it is increased.

Let's do it! Grab the code from the previous tutorial so that we can build off of it:

```
//Jonathan Holmes (crait)
//October 18th, 2016
//Printing Text

#include <Arduboy.h>
Arduboy arduboy;

void setup() {
  // put your setup code here, to run once:
  arduboy.begin();
  arduboy.clear();
  arduboy.print("Holmes is cool!");
  arduboy.display();
}
```

```
void loop() {  
  // put your main code here, to run repeatedly:  
}
```

## Initialization

Whenever you create a variable, you have to **initialize** it, which is setting aside memory for the data and giving it a name. You've already done this but didn't realize it. Check out the `Arduboy arduboy;` line. You initialized an object called `arduboy`. Objects are a lil' more complex than I want to get into during this tutorial, but there are different kinds of variables that you can initialize. Here are some: Integers, booleans, characters, objects, doubles, and many more.

We'll initialize our number that's going to increase as an integer. This basically means that it's a whole number, without fractions. Integers will appear as `int` inside of C++ code.

To initialize a variable, you must put 3 things: The type of variable, the name of the variable, then a semi-colon. Let's call our variable `counter`. Here's the code: `int counter;` Put it under the `Arduboy arduboy;` line.

## Assignment

Whenever you create a variable, you can give it a value. This is called **assignment**. We don't know what `counter`'s value is because we never gave it one.

Let's clean up the code by removing the `arduboy.print();` and `arduboy.display();` functions from `setup()`. Instead, let's put the assignment there:

```
counter = 0;
```

This line of code is saying that `counter`'s value is now equal to 0. Instead of 0, you could put another number, a mathematical formula, or some other things that I'll explain below.

Here's what your code should look like:

```
//Jonathan Holmes (crait)
//October 18th, 2016
//Counter

#include <Arduboy.h>
Arduboy arduboy;

int counter;

void setup() {
  // put your setup code here, to run once:
  arduboy.begin();
  arduboy.clear();
  counter = 0;
}

void loop() {
  // put your main code here, to run repeatedly:
}
```

## Incrementing

Okay, our program will be repeating a few simple instructions over and over again. Basically, we'll change the value of counter, then display the value of counter, then repeat. To do this, we should add some code into the **loop()** function.

```
counter = counter + 1;
```

This line of code means that you are assigning the value of counter to itself plus 1. Or, in other words, counter's value is now equal to the value of counter + 1.

## Displaying The Variable's Value

Now that we have the variable increasing, let's display it! Remember how we used arduboy.print() to print some text to the screen? Well, we can use that same function to display numbers to the screen, too. Add arduboy.print(counter); .

If you were to run this code as it is, now, then the Arduboy's screen would fill up with numbers. If we're going to print something new, we need to be sure to erase what was previously on the screen. We need to add in arduboy.clear(); at the beginning of **loop()** and arduboy.display(); at the end to display the updated screen.

Have you ever used a type writer? Whenever you type letters, the cursor moves over. The `arduboy.print()` function works similarly. Every time you use the `arduboy.print()` function, it moves the **cursor** over. So we need to reset the cursor to the top of the screen with `arduboy.setCursor(0, 0);`. I will explain this more in a later tutorial, but just throw that at the top of the **loop()**.

## The Completed Code

I've added in some comments, but your code should look like the following:

```
//Jonathan Holmes (crait)
//October 18th, 2016
//Printing Text

//Include the Arduboy Library
#include <Arduboy.h>
//Initialize the arduboy object
Arduboy arduboy;
//Initialize our counter variable
int counter;
//The setup() function runs once when you turn your Arduboy on
void setup() {
    //Start the Arduboy properly and display the Arduboy Logo
    arduboy.begin();
    //Get rid of the Arduboy Logo and clear the screen
    arduboy.clear();
    //Assign our counter variable to be equal to 0
    counter = 0;
}
//The loop() function repeats forever after setup() is done
void loop() {
    //Clear whatever is printed on the screen
    arduboy.clear();
    //Move the cursor back to the top-left of the screen
    arduboy.setCursor(0, 0);
    //Increase counter's value by 1
    counter = counter + 1;
    //Print out the value of counter
    arduboy.print(counter);
    //Refresh the screen to show whatever's printed to it
    arduboy.display();
}
```



## Running the Code

Connect your Arduboy to your computer and upload this code to it. Your Arduboy should start counting up!



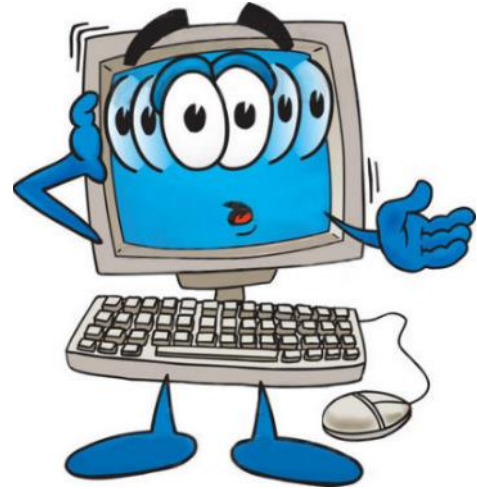
# Part 4: Questions & Button Input

## An Important Question

I used to teach programming to little kids and would always ask them a very important question. Think about this for a second: **What is a computer's favorite color?**

I've heard so many different answers to this. One kid told me, "*Green! Because hackers use green text on their screen!*" That was wrong. Of course, the correct answer is that computers don't have a favorite color! That's because computers don't have opinions. You can't just *ask* a computer what its favorite color is.

The only questions that you can ask a computer is to evaluate numbers, check the value of a variable or to count something.



## Stopping Robots from Washing Their Hair

Remember in the previous part of this tutorial, I told you about the back of the shampoo bottles? The instructions say:

1. Lather
2. Rinse
3. Repeat

The problem with this is that if you gave these instructions to a robot, it will repeat these steps forever. We need to implement some way to stop the robot from doing that. In computer programming, we can use **conditional statements**. Conditional statements are basically questions that we ask the computer and if the answer is true, then follow different instructions

1. Lather
2. Rinse
3. If you need to wash your hair again, then Repeat

The first kind of conditional statements you'll encounter are **if** statements. Like the example above, you can basically ask the computer if something is true, then do something. A robot following the above instructions won't wash their hair forever since it will eventually be washed.

# Buttons

We can also use `if` statements to test buttons, too! That's how video games know *if* you're pushing buttons. Moving a character may look like this:

1. If the Right Button **is** pressed, **move** character **right**
2. If the Left Button **is** pressed, **move** character **left**
3. **Print** character picture **to** screen
4. Repeat

## Format

The format for `if` statements in C/C++ is:

```
if(something is true) {  
    DoSomething();  
}
```

Only if the `something is true`, then `DoSomething();` command will run.

To check if the A Button is pressed, we need to use the `arduboy.pressed(A_BUTTON)` function. Putting it in the correct format, it will look like this:

```
if( arduboy.pressed(A_BUTTON) == true ) {  
    DoSomething();  
}
```

Notice that we use a double equal sign `==` instead of just a single equal sign `=`. You'll mostly see this inside of conditional statements. Don't let it scare you! A single equal sign `=` means that you are assigning a value. A double equal sign `==` is checking the value. I'll talk more about this later!

Above, we checked to see if the A Button was being pressed, but we can also check if the A Button is not being pressed!

```
if( arduboy.pressed(A_BUTTON) == false ) {  
    DoSomething();  
}
```

Cool, right?!

# Modifying The Code

Let's go grab that code from Part 3 and modify it:

```
//Jonathan Holmes (crait)
//October 18th, 2016
//Printing Text
//Include the Arduboy Library
#include <Arduboy.h>
//Initialize the arduboy object
Arduboy arduboy;
//Initialize our counter variable
int counter;
//The setup() function runs once when you turn your Arduboy on
void setup() {
    //Start the Arduboy properly and display the Arduboy Logo
    arduboy.begin();
    //Get rid of the Arduboy Logo and clear the screen
    arduboy.clear();
    //Assign our counter variable to be equal to 0
    counter = 0;
}
//The loop() function repeats forever after setup() is done
void loop() {
    //Clear whatever is printed on the screen
    arduboy.clear();
    //Move the cursor back to the top-left of the screen
    arduboy.setCursor(0, 0);
    //Increase counter's value by 1
    counter = counter + 1;
    //Print out the value of counter
    arduboy.print(counter);
    //Refresh the screen to show whatever's printed to it
    arduboy.display();
}
```

Remember that counter is our variable that increases forever in our previous code. What we're going to do is change this code to increase or decrease counter depending on if you're holding the Up Button or Down Button.

We need to find the line that increases counter and remove it. After we clear the screen with `arduboy.clear()`; , let's add the following code:

```
if( arduboy.pressed(UP_BUTTON) == true ) {  
    //Increase counter  
}  
if( arduboy.pressed(DOWN_BUTTON) == true ) {  
    //Decrease counter  
}
```

Our code to increase counter was `counter = counter + 1`; . Using that code, we can decrease it by switching the plus sign for minus sign. To decrease counter, our code will be `counter = counter - 1`; . Try to plug those in and when you're done, check to see if it is the same as the completed code, below.

## Testing Numbers & Variables

I was going to end the tutorial now, but let's extend it a little more and tell you about testing numbers!

I told you before that inside of an `if` statement, you can check to see if something is true or false. But, you can also check the values of variables or numbers.

```
if( counter == 36 ) {  
    arduboy.setCursor(30, 30);  
    arduboy.print("Yay!");  
}
```

What do you think would happen if you ran the above code? `if` counter is equal to 36, then the Arduboy will set the cursor and print out "Yay!" to the screen. Notice that you can put multiple lines of code inside of the `if` statements.

Add that right after the other `if` statements that check for the Up and Down Buttons.

## The Completed Code

```
//Jonathan Holmes (crait)
//October 21st, 2016
//Button Test
//Include the Arduboy Library
#include <Arduboy.h>
//Initialize the arduboy object
Arduboy arduboy;
//Initialize our counter variable
int counter;
//The setup() function runs once when you turn your Arduboy on
void setup() {
    //Start the Arduboy properly and display the Arduboy Logo
    arduboy.begin();
    //Get rid of the Arduboy Logo and clear the screen
    arduboy.clear();
    //Assign our counter variable to be equal to 0
    counter = 0;
}
//The loop() function repeats forever after setup() is done
void loop() {
    //Clear whatever is printed on the screen
    arduboy.clear();
    //Check if the UP_BUTTON is being pressed
    if( arduboy.pressed(UP_BUTTON) == true ) {
        //Increase counter by 1
        counter = counter + 1;
    }
    //Check if the DOWN_BUTTON is being pressed
    if( arduboy.pressed(DOWN_BUTTON) == true ) {
        //Decrease counter
        counter = counter - 1;
    }
    //Check if counter is equal to 36
    if( counter == 36 ) {
        arduboy.setCursor(30, 30);
        arduboy.print("Yay!");
    }
    //Move the cursor back to the top-left of the screen
    arduboy.setCursor(0, 0);
    //Print out the value of counter
    arduboy.print(counter);
    //Refresh the screen to show whatever's printed to it
    arduboy.display();
}
```

## Running The Code

Run the code and see what happens when you push the Up and Down Button!



# Part 5: Your First Game

I think you've got all the skills you need to make your first game! This game will be a number guessing game where the Arduboy picks a random number and the player has 7 attempts to guess the number. After each guess, the Arduboy will tell the player a hint. If the player guesses right, the player wins!

## Starting Code

Okay, let's start a new project. Open the Arduino IDE and go to **File > New**. In the new window, add some comments at the top that include your name and the title of this game.

Whenever I make a new game, the first things I do are: 1) Include the Arduboy library, 2) Create a new Arduboy object, 3) Prepare the `setup()` and `loop()` sections.

Here's what the code looks like when we're ready to begin:

```
//Jonathan Holmes (crait)  
//October 24th, 2016  
//Number Guessing Game  
  
#include <Arduboy.h>  
Arduboy arduboy;  
  
void setup() {  
    arduboy.begin();  
    arduboy.clear();  
  
}  
  
void loop() {  
    arduboy.clear();  
  
    arduboy.display();  
}
```

## Setting Up Variables

For our game, we need to set up some variables. We need one to keep track of how many attempts the players has left, what the number the player is guessing, what number the Arduboy picked, and one to know if the player has won or not.

At this point, you only know how to use `int`'s, so we'll continue using them. In the next part, I'm going to teach you about more of them.



Anyway, let's initialize the variables by adding this code before `setup()`:

```
int playerwin;  
int attempts;  
int guessednumber;  
int randomnumber;
```

Inside of `setup()`, let's assign these variables to 0.

```
playerwin = 0;  
attempts = 0;  
guessednumber = 0;
```

`randomnumber` isn't supposed to be 0. Instead, let's use the following code to assign a random number to it between 1 and 100;

```
srand(7/8);  
randomnumber = 1 + rand() % 100;
```

This is some wizardry if you don't know much about computer programming. But keep following these tutorials and we'll explain them in a later part.

Next, we're going to put most of our game's code between the `arduboy.clear()` and `arduboy.display()` functions. So, put a comment in between them so we can keep track of where we are. Your code should now look like this:

```
//Jonathan Holmes (crait)  
//October 24th, 2016  
//Number Guessing Game  
  
#include <Arduboy.h>  
Arduboy arduboy;  
int playerwin;  
int attempts;  
int guessednumber;  
int randomnumber;  
  
void setup() {  
  arduboy.begin();  
  arduboy.clear();  
  playerwin = 0;  
  attempts = 0;  
  guessednumber = 0;  
  randomnumber = 0;  
}
```

```
void loop() {  
  arduboy.clear();  
  //Gameplay code goes here  
  arduboy.display();  
}
```

## Else

We want our player to know if they have won or not. So, our `playerwin` variable will help with that. If `playerwin` is equal to 0, the player has not won. If `playerwin` is not equal to 0, then the player has won. In our code, we can put 2 `if` statements in to handle this.

```
if( playerwin == 0 ) {  
  //Ask the player for a number and play the game  
}  
if( playerwin == 1 ) {  
  //Tell the player that they won!  
}
```

There can be some problems from doing our code this way, so I am going to teach you a better method using the `if-else` statement.

What we want is for our game to run some code if the player didn't win, yet, and run other code if they did. At the end of an `if` statement, we can add `else` and more braces for a new section of code. Confused? Take a look below:

```
if( playerwin == 0 ) {  
  //Ask the player for a number and play the game  
} else {  
  //Tell the player that they won!  
}
```

*Else* is just another way to say *otherwise*. The above code says that if `playerwin` is equal to 0, then we need to run some code, otherwise we need to run some other code.

# Coding The Game

Let's stick the above `if-else` statement into the game and see where we're at:

```
//Jonathan Holmes (crait)
//October 24th, 2016
//Number Guessing Game

#include <Arduboy.h>
Arduboy arduboy;
int playerwin;
int attempts;
int guessednumber;
int randomnumber;

void setup() {
  arduboy.begin();
  arduboy.clear();
  playerwin = 0;
  attempts = 0;
  guessednumber = 0;
  randomnumber = 0;
}

void loop() {
  arduboy.clear();
  if( playerwin == 0 ) {
    //Ask the player for a number and play the game
  } else {
    //Tell the player that they won!
  }
  arduboy.display();
}
```

Now, we have two sections of code we need to finish: 1) Asking the player for numbers and 2) Telling the player that they won.

## Guessing A Number

The player will have 7 attempts to guess a number. So, when their attempts is at 7, we'll show them a game over screen instead of letting them guess, again.

We'll use another `if-else` statement.

```
if( attempts == 7 ) {  
    //Game Over screen  
} else {  
    //Player has more attempts  
}
```

We'll worry about the Game Over screen in a minute. For now, let's modify the part with players having more attempts remaining.

The first thing we want to do is handle the buttons. If a player uses the Up or Down button, the guessednumber will increase or decrease. When the player presses the A Button, the player will actually attempt to guess that number.

```
if( arduboy.pressed(UP_BUTTON) == true ) {  
    guessednumber = guessednumber + 1;  
}  
if( arduboy.pressed(DOWN_BUTTON) == true ) {  
    guessednumber = guessednumber - 1;  
}  
if( arduboy.pressed(A_BUTTON) == true ) {  
    //Guess number  
}
```

This is something we've seen before in the previous tutorial. After we handle the button input, we want to display the number being guessed as well as how many attempts the player has left. Add this code in:

```
arduboy.setCursor(0, 0);  
arduboy.print("Attempt: ");  
arduboy.print(attempts);  
arduboy.print("\n");  
arduboy.print("Number to guess: ");  
arduboy.print(guessednumber);
```

You should know what this code does. However, there is something new: `arduboy.print("\n");` This line of code creates a linebreak, which is like hitting the *Enter* button on your keyboard to insert a new line of text.

Let's go back and add one more `if-else` statement in this to handle the player making a guess! Remember, if the player guesses the random number, the player will win, otherwise (else) the player use up an attempt.

Remember, when a player wins, we already decided that `playerwin` would be changed from 0 to another number! Change the A Button code to this:

```
if( arduboy.pressed(A_BUTTON) == true ) {  
    if( guessednumber == randomnumber ) {  
        playerwin = 1;  
    } else {  
        attempts = attempts + 1;  
    }  
}
```

Makes sense, right? Here's what the code should currently look like! We're almost done!

```
//Jonathan Holmes (crait)  
//October 24th, 2016  
//Number Guessing Game  
  
#include <Arduboy.h>  
Arduboy arduboy;  
int playerwin;  
int attempts;  
int guessednumber;  
int randomnumber;  
  
void setup() {  
    arduboy.begin();  
    arduboy.clear();  
    playerwin = 0;  
    attempts = 0;  
    guessednumber = 0;  
    randomnumber = 0;  
    srand(7/8);  
    randomnumber = 1 + rand() % 100;  
}
```

```

void loop() {
  arduboy.clear();
  if( playerwin == 0 ) {
    //Ask the player for a number and play the game
    if( attempts == 7 ) {
      //Game Over screen
    } else {
      //Player has more attempts
      if( arduboy.pressed(UP_BUTTON) == true ) {
        guessednumber = guessednumber + 1;
      }
      if( arduboy.pressed(DOWN_BUTTON) == true ) {
        guessednumber = guessednumber - 1;
      }
      if( arduboy.pressed(A_BUTTON) == true ) {
        if( guessednumber == randomnumber ) {
          playerwin = 1;
        } else {
          attempts = attempts + 1;
        }
      }
      arduboy.setCursor(0, 0);
      arduboy.print("Attempt: ");
      arduboy.print(attempts);
      arduboy.print("\n");
      arduboy.print("Number to guess: ");
      arduboy.print(guessednumber);
    }
  } else {
    //Tell the player that they won!
  }
  arduboy.display();
}

```

## Game Over Screen

The Game Over screen is pretty easy. We need to put 2 things into the Game Over section: 1) Tell the player they lost and what the correct number *actually* was, and 2) Let them push A to restart the game. We already know how to tell the player that they lost:

```

arduboy.setCursor(0, 0);
arduboy.print("You lost!");
arduboy.print("\n");
arduboy.print("Correct Number: ");
arduboy.print(randomnumber);

```

Easy, right? When the player presses the A Button, we want the variables to reset and find a new random number. Here's what that code looks like.

```
if( arduboy.pressed(A_BUTTON) == true ) {  
    randomnumber = 1 + rand() % 100;  
    attempts = 0;  
    playerwin = 0;  
}
```

## Win Screen

The Win screen will look almost identical to the Game Over screen. Let's add this code to the Win screen:

```
arduboy.setCursor(0, 0);  
arduboy.print("You won!");  
arduboy.print("\n");  
arduboy.print("Correct Number: ");  
arduboy.print(randomnumber);  
if( arduboy.pressed(A_BUTTON) == true ) {  
    randomnumber = 1 + rand() % 100;  
    attempts = 0;  
    playerwin = 0;  
}
```

## Uh, Oh!! There's A Problem!

Uh, oh!!! There's a problem... We're almost done, but there's a problem that I specifically avoided telling you about. Go ahead and run this code on your Arduboy and see if you can figure out what it is! If you've done the previous tutorials in this series, you may have already figured it out!

Here's what the code should look like so far:

```
//Jonathan Holmes (crait)  
//October 24th, 2016  
//Number Guessing Game  
  
#include <Arduboy.h>  
Arduboy arduboy;  
int playerwin;  
int attempts;  
int guessednumber;  
int randomnumber;
```

```

void setup() {
  arduboy.begin();
  arduboy.clear();
  playerwin = 0;
  attempts = 0;
  guessednumber = 0;
  randomnumber = 0;
  srand(7/8);
  randomnumber = 1 + rand() % 100;
}

void loop() {
  arduboy.clear();
  if( playerwin == 0 ) {
    //Ask the player for a number and play the game
    if( attempts == 7 ) {
      //Game Over screen
      arduboy.setCursor(0, 0);
      arduboy.print("You lost!");
      arduboy.print("\n");
      arduboy.print("Correct Number: ");
      arduboy.print(randomnumber);
      if( arduboy.pressed(A_BUTTON) == true ) {
        randomnumber = 1 + rand() % 100;
        attempts = 0;
        playerwin = 0;
      }
    } else {
      //Player has more attempts
      if( arduboy.pressed(UP_BUTTON) == true ) {
        guessednumber = guessednumber + 1;
      }
      if( arduboy.pressed(DOWN_BUTTON) == true ) {
        guessednumber = guessednumber - 1;
      }
      if( arduboy.pressed(A_BUTTON) == true ) {
        if( guessednumber == randomnumber ) {
          playerwin = 1;
        } else {
          attempts = attempts + 1;
        }
      }
    }
  }
}

```



```

        arduboy.setCursor(0, 0);
        arduboy.print("Attempt: ");
        arduboy.print(attempts);
        arduboy.print("\n");
        arduboy.print("Number to guess: ");
        arduboy.print(guessednumber);
    }
} else {
    //Tell the player that they won!
    arduboy.setCursor(0, 0);
    arduboy.print("You won!");
    arduboy.print("\n");
    arduboy.print("Correct Number: ");
    arduboy.print(randomnumber);
    if( arduboy.pressed(A_BUTTON) == true ) {
        randomnumber = 1 + rand() % 100;
        attempts = 0;
        playerwin = 0;
    }
}
arduboy.display();
}

```

Did you figure out the problem? Yeah, it's kinda unexpected. The Arduboy is *too* fast when playing a game like this! Playing the game and pressing the A button will blast through the entire menu super fast. Same with pressing Up or Down... The numbers will scroll *too* fast.

## Fixing This Issue

So, we need to do 1 of 2 things to fix this speed issue. We *could* intentionally slow the Arduboy down, however, when making some games, they will move too slow to be enjoyable. That's not really an option for us. The second option is to have some kind of variable that keeps track if the player is holding down the button or not. The Arduboy is so fast that if you press the button for a fraction of a second, it will already run through the code several times, thinking you've pressed the button several times.

Let's start by going all the way to the top of our file to initialize some new variables.

```

int upbuffer;
int downbuffer;
int abuffer;

```

Next, let's assign them to be 0 in our `setup()` section.

```
upbuffer = 0;
downbuffer = 0;
abuffer = 0;
```

To keep track of our button presses, we'll have these variables hold either a 0 or a 1. If the buffer variable is 0, that means that the player did not press that button. If the buffer variable is equal to 1, that means the player *did* press that button.

Whenever the player lets go of a button, we want to set these variables back to 0.

This means that any time we press a button, we can check to see if the button was being held down or if it was pressed for the first time. If it is being held down, we want to pretend like the button was not even pressed and ignore it.

This may be a little confusing for some people to understand, but please bear with me.

At the end of `loop()`, let's add some code that can do that.

```
if( arduboy.notPressed(A_BUTTON) == true ) {
    abuffer = 0;
}
if( arduboy.notPressed(DOWN_BUTTON) == true ) {
    downbuffer = 0;
}
if( arduboy.notPressed(UP_BUTTON) == true ) {
    upbuffer = 0;
}
```

Now, every place that we check to see if a button is pressed or not, we need to also check to see if the buffer variable is 0 or 1.

If the button is being pressed and the buffer variable is 0, then we will run the code as well and change the buffer variable to 1.

Replace all the button checks that look like the following...

```
if( arduboy.pressed(A_BUTTON) == true ) {
```

to...

```
if( arduboy.pressed(A_BUTTON) == true and abuffer == 0 ) {  
  abuffer = 1;
```

Notice that the word `and` allows us to check for multiple things inside of an `if` statement. Do this with Up and Down Button checks, too!

From...

```
if( arduboy.pressed(DOWN_BUTTON) == true ) {
```

to...

```
if( arduboy.pressed(DOWN_BUTTON) == true and downbuffer == 0 ) {  
  downbuffer = 1;
```

From...

```
if( arduboy.pressed(UP_BUTTON) == true ) {
```

to...

```
if( arduboy.pressed(UP_BUTTON) == true and upbuffer == 0 ) {  
  upbuffer = 1;
```

After you do that, test your code, which should look just like this:

```
//Jonathan Holmes (crait)
//October 24th, 2016
//Number Guessing Game

#include <Arduboy.h>
Arduboy arduboy;
int playerwin;
int attempts;
int guessednumber;
int randomnumber;
int upbuffer;
int downbuffer;
int abuffer;

void setup() {
  arduboy.begin();
  arduboy.clear();
  playerwin = 0;
  attempts = 0;
  guessednumber = 0;
  randomnumber = 0;
  srand(7/8);
  randomnumber = 1 + rand() % 100;
  upbuffer = 0;
  downbuffer = 0;
  abuffer = 0;
}

void loop() {
  arduboy.clear();
  if( playerwin == 0 ) {
    //Ask the player for a number and play the game
    if( attempts == 7 ) {
      //Game Over screen
      arduboy.setCursor(0, 0);
      arduboy.print("You lost!");
      arduboy.print("\n");
      arduboy.print("Correct Number: ");
      arduboy.print(randomnumber);
      if( arduboy.pressed(A_BUTTON) == true and abuffer == 0 ) {
        abuffer = 1;
        randomnumber = 1 + rand() % 100;
        attempts = 0;
        playerwin = 0;
      }
    }
  }
}
```

```

} else {
    //Player has more attempts
    if( arduboy.pressed(UP_BUTTON) == true and upbuffer == 0 ) {
        upbuffer = 1;
        guessednumber = guessednumber + 1;
    }
    if( arduboy.pressed(DOWN_BUTTON) == true and downbuffer == 0 ) {
        downbuffer = 1;
        guessednumber = guessednumber - 1;
    }
    if( arduboy.pressed(A_BUTTON) == true and abuffer == 0 ) {
        abuffer = 1;
        if( guessednumber == randomnumber ) {
            playerwin = 1;
        } else {
            attempts = attempts + 1;
        }
    }
    arduboy.setCursor(0, 0);
    arduboy.print("Attempt: ");
    arduboy.print(attempts);
    arduboy.print("\n");
    arduboy.print("Number to guess: ");
    arduboy.print(guessednumber);
}
} else {
    //Tell the player that they won!
    arduboy.setCursor(0, 0);
    arduboy.print("You won!");
    arduboy.print("\n");
    arduboy.print("Correct Number: ");
    arduboy.print(randomnumber);
    if( arduboy.pressed(A_BUTTON) == true and abuffer == 0 ) {
        abuffer = 1;
        randomnumber = 1 + rand() % 100;
        attempts = 0;
        playerwin = 0;
    }
}
}
if( arduboy.notPressed(A_BUTTON) == true) {
    abuffer = 0;
}
if( arduboy.notPressed(DOWN_BUTTON) == true ) {
    downbuffer = 0;
}
}

```

```
if( arduboy.notPressed(UP_BUTTON) == true ) {
    upbuffer = 0;
}
arduboy.display();
}
```

## Hints

After testing your game, you may have realized that it's just *too hard* for someone to actually beat. I tried a many times and won only once. It's almost impossible!

Let's make the game a little easier and a little more fun by adding in hints to the player. To start this, let's create a new variable called `lastguess` and give it a value of 0.

Every time the player makes a guess, let's set `lastguess` to have the same value of their `guessednumber`.

Look for the line of code where the player uses one of their attempts:

```
attempts = attempts + 1;
```

After that line, let's add the following:

```
lastguess = guessednumber;
```

Add a new linebreak after with `arduboy.print("\n");` and add a new `if-else` statement. We're going to check the player guessed anything, yet.

```
arduboy.print("\n");
if( attempts == 0 ) {
    //No Last guess
} else {
    //Last guess was wrong
}
```

If there have been 0 attempts so far, then let's print out "Good luck!" Otherwise, let's display the last guess. That code should now look like:

```
arduboy.print("\n");
if( attempts == 0 ) {
    arduboy.print("Good luck!");
} else {
    arduboy.print(lastguess);
}
```

Our hint will actually tell the user if lastguess is less than < or greater than > the randomnumber and print that out. That code should now look like:

```
arduboy.print("\n");
if( attempts == 0 ) {
  arduboy.print("Good luck!");
} else {
  arduboy.print(lastguess);
  if( lastguess > randomnumber ) {
    arduboy.print(" is too high!");
  }
  if( lastguess < randomnumber ) {
    arduboy.print(" is too low!");
  }
}
```

You've seen all this code before, except for checking if a number is less than < or greater than > another number.

## Full Code

Yes! You're finally done!! Test out this code and have fun!

```
//Jonathan Holmes (crait)
//October 24th, 2016
//Number Guessing Game

#include <Arduboy.h>
Arduboy arduboy;
int playerwin;
int attempts;
int guessednumber;
int randomnumber;
int upbuffer;
int downbuffer;
int abuffer;
int lastguess;
```

```

void setup() {
  arduboy.begin();
  arduboy.clear();
  playerwin = 0;
  attempts = 0;
  guessednumber = 0;
  randomnumber = 0;
  srand(7/8);
  randomnumber = 1 + rand() % 100;
  upbuffer = 0;
  downbuffer = 0;
  abuffer = 0;
  lastguess = 0;
}

void loop() {
  arduboy.clear();
  if( playerwin == 0 ) {
    //Ask the player for a number and play the game
    if( attempts == 7 ) {
      //Game Over screen
      arduboy.setCursor(0, 0);
      arduboy.print("You lost!");
      arduboy.print("\n");
      arduboy.print("Correct Number: ");
      arduboy.print(randomnumber);
      if( arduboy.pressed(A_BUTTON) == true and abuffer == 0 ) {
        abuffer = 1;
        randomnumber = 1 + rand() % 100;
        attempts = 0;
        playerwin = 0;
      }
    } else {
      //Player has more attempts
      if( arduboy.pressed(UP_BUTTON) == true and upbuffer == 0 ) {
        upbuffer = 1;
        guessednumber = guessednumber + 1;
      }
      if( arduboy.pressed(DOWN_BUTTON) == true and downbuffer == 0 ) {
        downbuffer = 1;
        guessednumber = guessednumber - 1;
      }
    }
  }
}

```



```

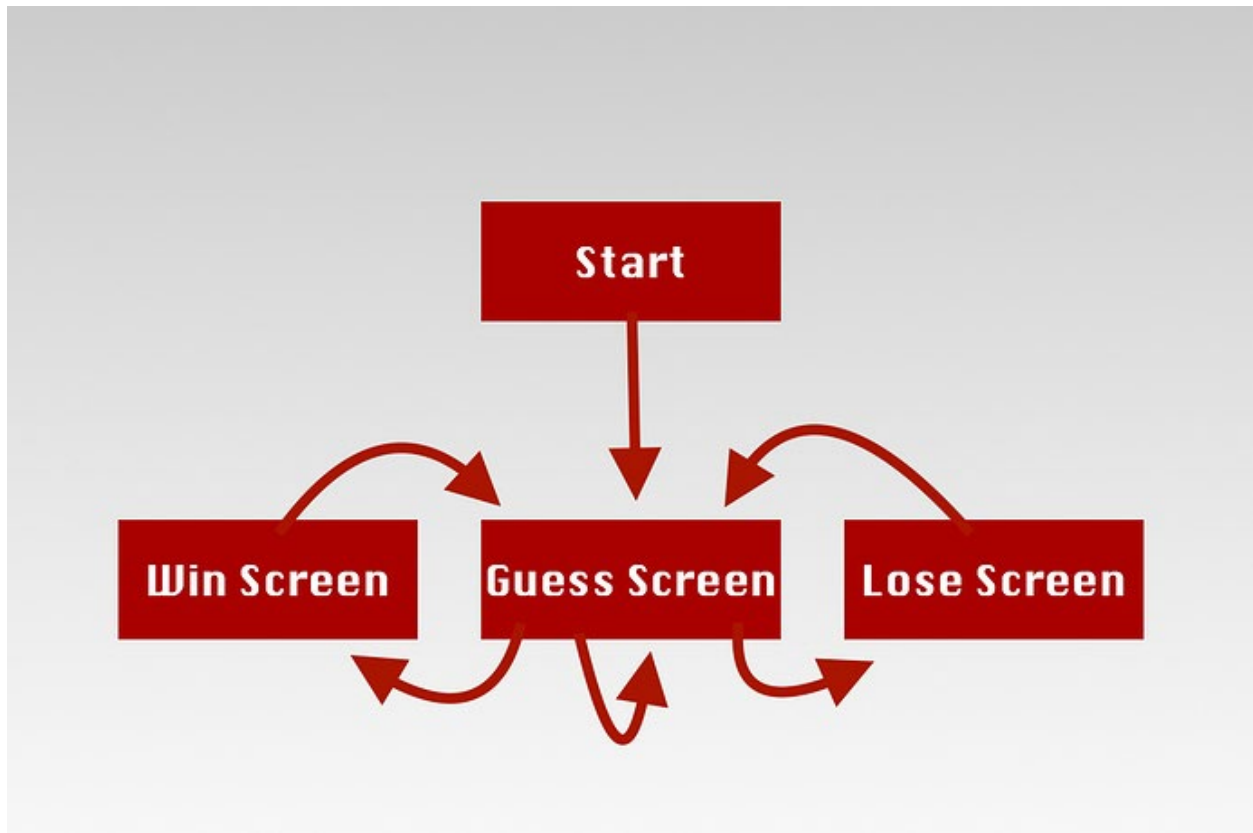
    if( arduboy.pressed(A_BUTTON) == true and abuffer == 0 ) {
        abuffer = 1;
        if( guessednumber == randomnumber ) {
            playerwin = 1;
        } else {
            attempts = attempts + 1;
            lastguess = guessednumber;
        }
    }
    arduboy.setCursor(0, 0);
    arduboy.print("Attempt: ");
    arduboy.print(attempts);
    arduboy.print("\n");
    arduboy.print("Number to guess: ");
    arduboy.print(guessednumber);
    arduboy.print("\n");
    if( attempts == 0 ) {
        arduboy.print("Good luck!");
    } else {
        arduboy.print(lastguess);
        if( lastguess > randomnumber ) {
            arduboy.print(" is too high!");
        }
        if( lastguess < randomnumber ) {
            arduboy.print(" is too low!");
        }
    }
}
} else {
    //Tell the player that they won!
    arduboy.setCursor(0, 0);
    arduboy.print("You won!");
    arduboy.print("\n");
    arduboy.print("Correct Number: ");
    arduboy.print(randomnumber);
    if( arduboy.pressed(A_BUTTON) == true and abuffer == 0 ) {
        abuffer = 1;
        randomnumber = 1 + rand() % 100;
        attempts = 0;
        playerwin = 0;
    }
}
if( arduboy.notPressed(A_BUTTON) == true) {
    abuffer = 0;
}
if( arduboy.notPressed(DOWN_BUTTON) == true ) {
    downbuffer = 0;
}
}

```

```
if( arduboy.notPressed(UP_BUTTON) == true ) {  
    upbuffer = 0;  
}  
arduboy.display();  
}
```



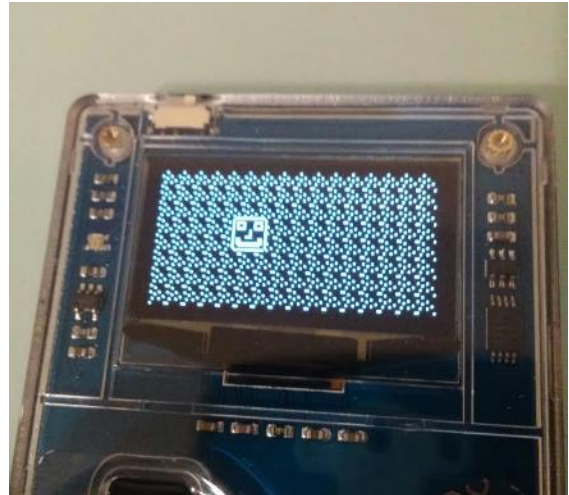
For anyone wanting a graphical representation of this game, here's a flow chart.



# Part 6: Graphics

## This Tutorial

In this tutorial, I'm going to show you how to make a game demo where you can move a sprite around a screen that has a tiled background. I'll also teach you some more basic programming fundamentals.



## Starting Code

Let's use this code to start our program. Open the Arduino IDE, start a new project, and make sure the code looks like this:

```
//Jonathan Holmes (crait)
//November 1st, 2016
//Moving Character Demo
#include <Arduboy.h>
Arduboy arduboy;
void setup() {
  arduboy.begin();
  arduboy.clear();
}
void loop() {
  arduboy.clear();

  arduboy.display();
}
```

## Starting Images

Right-click and download these two images to your computer.



This image will be your background image. We're going to tile it across the entire Arduboy screen.



This image will be your player *sprite*, which is the image that represents your character. You'll move this face around your screen on the Arduboy.

## Convert Images

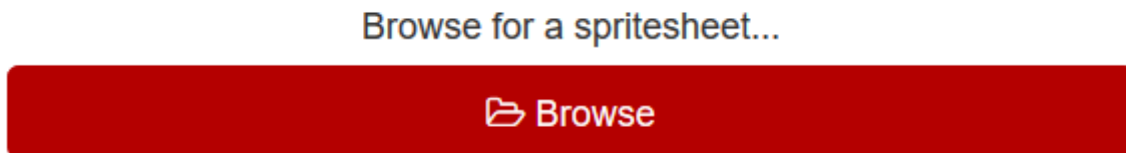
Remember how I said that you can store data and variables in several formats? Remember, one format that we used before for numbers was `int`. Well, to store images, we actually want to convert them into raw data that we can store them in *byte arrays*. Don't worry too much about what that means, just know it's different than an `int` and can hold a different amount of data.

I've seen some people on this forum say that they convert images to raw data by hand, but luckily, I created an online tool that will do it for us called **ToChars**. Open up <http://www.crait.net/tochars/> in a new tab and follow these instructions for each image.

1) *Click Start*



2) *Click Browse*



3) *Set Size*

For the background image, set the height and width to both be 8 pixels. For the player sprite, set them to be 16 pixels.

Set parameters...


Sprite Width:	<input type="text" value="8"/>
Sprite Height:	<input type="text" value="8"/>

#### 4) Click Convert

**Data Type**    ☒ Hexadecimal  
                  ☐ Binary

 Convert

#### 5) Grab Data



Sprite Count: 1

Output:

0xfe, 0x1, 0x3d, 0x25, 0x25, 0x3d, 0x1, 0x1, 0xc1, 0x1, 0x3d, 0x25, 0x25, 0x3d, 0x1, 0xfe, 0x7f, 0x80, 0x9c, 0xbc, 0xb0, 0xb0, 0xb2, 0xb2, 0xb3, 0xb0, 0xb0, 0xb0, 0xbc, 0x9c, 0x80, 0x7f,

Start Over

Here's the raw data for background image:

```
0x84, 0x20, 0x9, 0x00, 0x24, 0x00, 0x10, 0x80,
```

Here's the raw data for the player sprite:

```
0xfe, 0x1, 0x3d, 0x25, 0x25, 0x3d, 0x1, 0x1, 0xc1, 0x1, 0x3d, 0x25, 0x25, 0x3d, 0x1, 0xfe, 0x7f, 0x80, 0x9c, 0xbc, 0xb0, 0xb0, 0xb2, 0xb2, 0xb3, 0xb0, 0xb0, 0xb0, 0xbc, 0x9c, 0x80, 0x7f,
```

Hold onto these for a second and I'll teach you where to put them.

## Talking About Initializing Variables

Alright, I kinda left out some information about initializing variables. So far, I've had you initialize variables at the top of the sketch, right under the `Arduboy arduboy;` line. But, it's important to know that you can actually initialize them pretty much anywhere. We'll do that later on, so I don't want it to surprise you.

I want to teach you a shortcut, though. When we've been initializing variables, we have been doing them in the following format:

```
int counter;  
counter = 5;
```

This is valid, but there's a shortcut that you can take. You can actually set a value to a variable at the same time you initialize it.

```
int counter = 5;
```

Cool, huh? You can also do this with constants.

## Constants

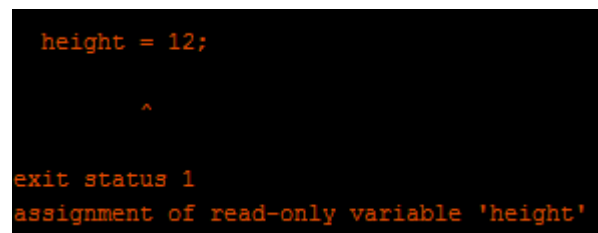
A **constant** is like a variable in the sense that you can have it store data that you can also initialize. The only difference is that the data can't be changed. It saves memory to store data in constants instead of variables if you aren't going to change the data that it holds. We'll do that with our image data since we won't change the images at all.

To create a constant, you use the `const` keyword in front of the initialization.

```
int counter = 5;  
const int height = 5;
```

Since you can't change the data inside of a constant, `height` cannot be changed or given a new value. If you were to use the following code, you would get an error and the code will not work.

```
height = 12;
```

A screenshot of a terminal window with a black background and orange text. The first line shows the code 'height = 12;'. Below it, there is a small orange caret '^' pointing to the space before the semicolon. The next line shows 'exit status 1', and the final line shows the error message 'assignment of read-only variable 'height''.

```
height = 12;  
      ^  
exit status 1  
assignment of read-only variable 'height'
```

## Storing Images

Whew! That was a lot to tell you real quick, but let's move on to storing the raw image data into byte arrays. Where you normally initialize variables, put the following code:

```
const unsigned char background[] PROGMEM = {  
    0x84, 0x20, 0x9, 0x00, 0x24, 0x00, 0x10, 0x80,  
};
```

```
const unsigned char player[] PROGMEM = {
    0xfe, 0x1, 0x3d, 0x25, 0x25, 0x3d, 0x1, 0x1, 0xc1, 0x1, 0x3d, 0x25,
    0x25, 0x3d, 0x1, 0xfe, 0x7f, 0x80, 0x9c, 0xbc, 0xb0, 0xb0, 0xb2, 0xb2,
    0xb3, 0xb0, 0xb0, 0xb0, 0xbc, 0x9c, 0x80, 0x7f,
};
```

To summarize this code, we converted 2 images to 2 groups of *characters*. One is named `background` and the other is named `player`. Let's break this code down a little and go word-by-word.

- `const`: We already know that we are creating a variable that cannot be changed.
- `unsigned`: This means that we cannot use negative values. Just ignore this for a while.
- `char`: The way we store byte data is by putting it into a `char`, which stands for *character*. This is the kind of variable that we are creating instead of using an `int` like the last tutorial.
- `background`: Like all variables, we need to give these two character arrays names. The top one is called `background` and the bottom one is called `player`.
- `[]`: The brackets that you see here means that we're creating an *array*. An array is a group of variables. We're creating groups of `char` variables. We can call them *character arrays*.
- `=`: Like in the shortcut that I explained above, when you initialize a variable, you can assign a value to it at the same time. We can do this with arrays. We'll actually assign the characters directly into the array.
- `{ }`: Anything inside of the braces is what we're actually putting inside of the array.
- `0x84, 0x20, ...`: This is the data that we got from the **ToChars** site. It's an image stored in multiple bytes/characters. Each character is separated by a comma. **ToChars** converted the images *to characters*. Get it?

What about `PGGMEM` ?? No need to worry too much about what that means. Leave that in, though! It's a special word called a macro that will tell the compiler *where* to store the array in memory. It makes storing images more efficient for us.

Your game's code should look like this:

```
//Jonathan Holmes (crait)
//November 1st, 2016
//Moving Character Demo
#include <ArduBoy.h>
ArduBoy arduBoy;
const unsigned char background[] PROGMEM = {
    0x84, 0x20, 0x9, 0x00, 0x24, 0x00, 0x10, 0x80,
};
const unsigned char player[] PROGMEM = {
    0xfe, 0x1, 0x3d, 0x25, 0x25, 0x3d, 0x1, 0x1, 0xc1, 0x1, 0x3d, 0x25,
    0x25, 0x3d, 0x1, 0xfe, 0x7f, 0x80, 0x9c, 0xbc, 0xb0, 0xb0, 0xb2, 0xb2,
    0xb3, 0xb0, 0xb0, 0xb0, 0xbc, 0x9c, 0x80, 0x7f,
};
```



```
void setup() {  
    arduboy.begin();  
    arduboy.clear();  
}  
void loop() {  
    arduboy.clear();  
    arduboy.display();  
}
```

## Parameters

I can't wait to show you how to draw the above images to the Arduboy screen! But I can't just yet! I need to tell you about a function's parameters! Remember that a function is an instruction for the computer. Sometimes, they look like this:

```
arduboy.clear();
```

That clears the Arduboy's screen. It's a function that is easy to understand. Here's another:

```
arduboy.print(randomnumber);
```

This is from the last tutorial. The `randomnumber` is a variable that we put into the function. We told the Arduboy *what* to print. That's called a parameter. And in fact, functions can have multiple parameters. We already saw one earlier that did this in the last tutorial, too!

```
arduboy.setCursor(0, 0);
```

The `arduboy.setCursor()` function requires *two* parameters that are separated by a comma.

The function to draw an image to the Arduboy's screen has a lot more parameters that you need to use. This is actually pretty common.

## Drawing Images

So, we want to draw an image to the Arduboy screen, finally! Let's start by drawing the player sprite that we called `player`. Inside of the `loop()` area, after we clear the screen, let's add this line of code:

```
arduboy.drawBitmap(5, 10, player, 16, 16, WHITE);
```

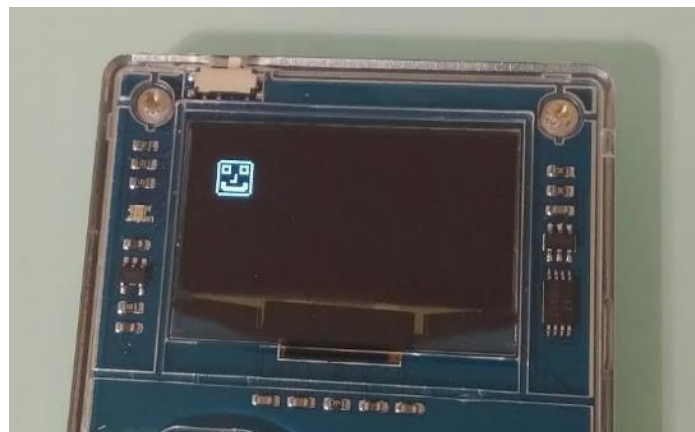
- The first two parameters are 5 and 10. These numbers represent the X and Y location that the image will be drawn to. Changing these numbers will change where the image appears.
- The next parameter that's given is the image that we want to draw, which is `player`.

- The next two parameters are the width and height of the image that is being drawn. For the `player` image, we'll have 16, 16, but for the background image, we'll have 8, 8 since that's how big those images are.
- the last parameter is what color we want to draw to the screen. `WHITE` means that we're drawing all the white pixels in the image as white. `BLACK` means that we're drawing all the white pixels in the image as black. It seems counter-intuitive, but it becomes useful later on.

Okay! If your code looks like the following, then go ahead and put it on your Arduboy!

```
//Jonathan Holmes (crait)
//November 1st, 2016
//Moving Character Demo
#include <Arduboy.h>
Arduboy arduboy;
const unsigned char background[] PROGMEM = {
    0x84, 0x20, 0x9, 0x00, 0x24, 0x00, 0x10, 0x80,
};
const unsigned char player[] PROGMEM = {
    0xfe, 0x1, 0x3d, 0x25, 0x25, 0x3d, 0x1, 0x1, 0xc1, 0x1, 0x3d, 0x25,
    0x25, 0x3d, 0x1, 0xfe, 0x7f, 0x80, 0x9c, 0xbc, 0xb0, 0xb0, 0xb2, 0xb2,
    0xb3, 0xb0, 0xb0, 0xb0, 0xbc, 0x9c, 0x80, 0x7f,
};
void setup() {
    arduboy.begin();
    arduboy.clear();
}
void loop() {
    arduboy.clear();
    arduboy.drawBitmap(5, 10, player, 16, 16, WHITE);
    arduboy.display();
}
```

Your Arduboy should properly display as:



# Control The Sprite

Okay, remember how we controlled the value of a variable with the Up and Down buttons? To move the sprite around, we'll do something very similar.

To start, let's initialize 2 variables at the top of the sketch:

```
int playerx = 5;
int playery = 10;
```

The `playerx` and `playery` variables will hold the coordinates for our image as it's moving around. In the `arduboy.drawBitmap()` function, let's replace the X and Y parameter with these variables.

```
arduboy.drawBitmap(playerx, playery, player, 16, 16, WHITE);
```

Let's increase/decrease those variables' values when the Up/Down and Left/Right buttons are pressed.

```
if(arduboy.pressed(LEFT_BUTTON)) {
    playerx = playerx - 1;
}
if(arduboy.pressed(RIGHT_BUTTON)) {
    playerx = playerx + 1;
}
if(arduboy.pressed(UP_BUTTON)) {
    playery = playery - 1;
}
if(arduboy.pressed(DOWN_BUTTON)) {
    playery = playery + 1;
}
```

That's all it takes to move an image around the screen! Test out the code on your Arduboy and have fun with it!

```
//Jonathan Holmes (crait)
//November 1st, 2016
//Moving Character Demo
#include <Arduboy.h>
Arduboy arduboy;
int playerx = 5;
int playery = 10;
const unsigned char background[] PROGMEM = {
    0x84, 0x20, 0x9, 0x00, 0x24, 0x00, 0x10, 0x80,
};
```

```

const unsigned char player[] PROGMEM = {
    0xfe, 0x1, 0x3d, 0x25, 0x25, 0x3d, 0x1, 0x1, 0xc1, 0x1, 0x3d, 0x25,
    0x25, 0x3d, 0x1, 0xfe, 0x7f, 0x80, 0x9c, 0xbc, 0xb0, 0xb0, 0xb2, 0xb2,
    0xb3, 0xb0, 0xb0, 0xb0, 0xbc, 0x9c, 0x80, 0x7f,
};
void setup() {
    arduboy.begin();
    arduboy.clear();
}
void loop() {
    arduboy.clear();
    if(arduboy.pressed(LEFT_BUTTON)) {
        playerx = playerx - 1;
    }
    if(arduboy.pressed(RIGHT_BUTTON)) {
        playerx = playerx + 1;
    }
    if(arduboy.pressed(UP_BUTTON)) {
        playery = playery - 1;
    }
    if(arduboy.pressed(DOWN_BUTTON)) {
        playery = playery + 1;
    }
    arduboy.drawBitmap(playerx, playery, player, 16, 16, WHITE);
    arduboy.display();
}

```

## For Loop

Okay, remember how I said that *loops* tell the computer to repeat a set of actions over and over again? I'm going to teach you about one called the **for** loop. This loop allows you to repeat a set of instructions for a specified amount of times.

Why is this important? We want to make a background for this game, but the background image that we have is too small to fill the entire screen, so we'll print it many times in different places until it fills up the entire screen.

Let's say we want to use the `arduboy.print()` function to print your name 10 times in a row. We'd have to use a loop keyword and we'll need a variable to keep track of how many times we've printed so far, and a line of code to increase that variable every time we loop through. Luckily, the **for** loop has a lot of that built-in. Take a look at it, below:

```

for( int counter = 0; counter < 10; counter = counter + 1 ) {
    arduboy.print("crait");
}

```

Let's break this down again:

- `for`: This is kinda like a function. It lets the Arduboy know you want to use a **for** loop.
- `int counter = 0;`: Whenever you run a for loop, you'll need to initialize/set a variable. We create a counter variable that is equal to 0. This code gets executed before the loop is entered.
- `counter < 10;`: This is the check to see if the loop should run or not. If counter is less than 10, then it should run.
- `counter = counter + 1`: After all of the instructions inside of the loop are followed, this code is ran. It increases counter by 1. Eventually, counter will grow big enough so that the loop does not execute anymore.
- `{ }`: Anything inside of these braces will be considered part of the loop and only be executed when the Arduboy is running this for loop.

Instead of this code printing my name 10 times, we can have it count!

```
for( int counter = 0; counter < 10; counter = counter + 1 ) {  
    arduboy.print( counter );  
}
```

If we run the above loop, the Arduboy would print out the numbers 0 to 9! You can change the number of times it will loop, you can change what number the counter starts with, and you can even change how many numbers counter is increased by every time the loop is run.

## Tile Background

Alrighty! Let's get the background implemented! We're almost done!

The Arduboy's screen resolution is 128 pixels wide by 64 pixels tall, which means if we use a background image of 8 pixels by 8 pixels, we would have 16 columns wide by 8 rows high.

Let's tile the background image 8 times across the top of the screen.

```
for( int backgroundx = 0; backgroundx < 128; backgroundx = backgroundx + 8  
) {  
    arduboy.drawBitmap( backgroundx, 0, background, 8, 8, WHITE );  
}
```

Do you understand what's going on, here? Notice that the backgroundx variable is used to count the loop, but also used when drawing the background image. Since we want to span the background across the width of the screen, we want to count up to 128. Because we want to put a new image every 8 pixels, we'll increase it by 8 every time a tile is drawn.

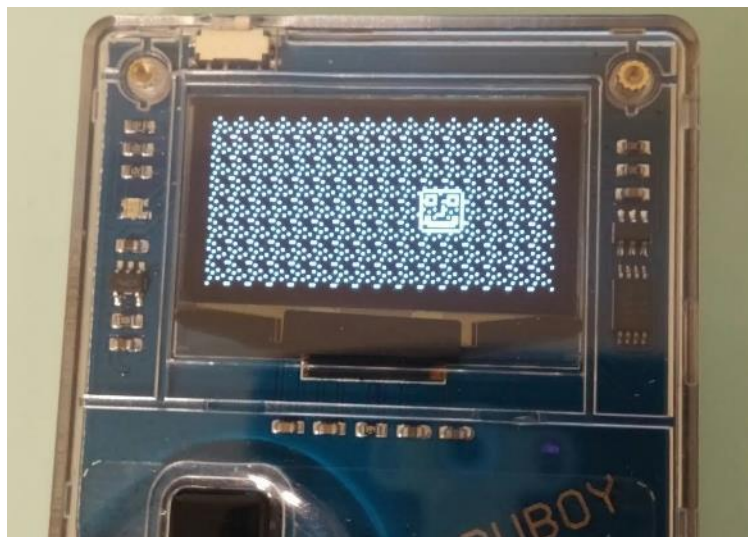
Add this code before you draw your sprite and run it on your Arduboy. It should appear like this:



Next, let's add another for loop. Instead of doing this after the previous loop, we'll have this loop **inside** of the other!

```
for( int backgroundx = 0; backgroundx < 128; backgroundx = backgroundx + 8
) {
    for( int backgroundy = 0; backgroundy < 64; backgroundy = backgroundy
+ 8 ) {
        arduboy.drawBitmap( backgroundx, backgroundy, background, 8, 8,
WHITE );
    }
}
```

To summarize the above code, for each column on the screen, we will draw several rows of background images until the screen is full. This is the result:



Here's the full code:

```
//Jonathan Holmes (crait)
//November 1st, 2016
//Moving Character Demo
#include <Arduboy.h>
Arduboy arduboy;
int playerx = 5;
int playery = 10;
const unsigned char background[] PROGMEM = {
    0x84, 0x20, 0x9, 0x00, 0x24, 0x00, 0x10, 0x80,
};
const unsigned char player[] PROGMEM = {
    0xfe, 0x1, 0x3d, 0x25, 0x25, 0x3d, 0x1, 0x1, 0xc1, 0x1, 0x3d, 0x25,
    0x25, 0x3d, 0x1, 0xfe, 0x7f, 0x80, 0x9c, 0xbc, 0xb0, 0xb0, 0xb2, 0xb2,
    0xb3, 0xb0, 0xb0, 0xb0, 0xbc, 0x9c, 0x80, 0x7f,
};
void setup() {
    arduboy.begin();
    arduboy.clear();
}
void loop() {
    arduboy.clear();
    if(arduboy.pressed(LEFT_BUTTON)) {
        playerx = playerx - 1;
    }
    if(arduboy.pressed(RIGHT_BUTTON)) {
        playerx = playerx + 1;
    }
    if(arduboy.pressed(UP_BUTTON)) {
        playery = playery - 1;
    }
    if(arduboy.pressed(DOWN_BUTTON)) {
        playery = playery + 1;
    }
    //For each column on the screen
    for( int backgroundx = 0; backgroundx < 128; backgroundx = backgroundx
+ 8 ) {
        //For each row in the column
        for( int backgroundy = 0; backgroundy < 64; backgroundy =
backgroundy + 8 ) {
            //Draw a background tile
            arduboy.drawBitmap( backgroundx, backgroundy, background, 8,
8, WHITE );
        }
    }
}
```

```
//Draw player sprite
arduboy.drawBitmap(playerx, playery, player, 16, 16, WHITE);
arduboy.display();
}
```

## Clean Up

Uh, oh! You notice the problem? Our character's face is clear and you can see the background though it. How can we fix that? Let's just simply draw a black square behind it to block the background images. Put this before we draw the player's sprite.

```
arduboy.fillRect(playerx, playery, 16, 16, BLACK);
```

## Finished Code

Here it is! Run this code on your Arduboy to make sure it works properly! If it does, then feel free to modify this code to start making your own game!

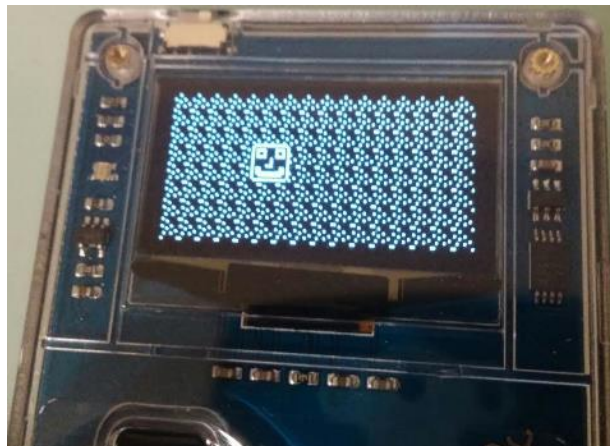
```
//Jonathan Holmes (crait)
//November 1st, 2016
//Moving Character Demo
#include <Arduboy.h>
Arduboy arduboy;
int playerx = 5;
int playery = 10;
const unsigned char background[] PROGMEM = {
    0x84, 0x20, 0x9, 0x00, 0x24, 0x00, 0x10, 0x80,
};
const unsigned char player[] PROGMEM = {
    0xfe, 0x1, 0x3d, 0x25, 0x25, 0x3d, 0x1, 0x1, 0xc1, 0x1, 0x3d, 0x25,
    0x25, 0x3d, 0x1, 0xfe, 0x7f, 0x80, 0x9c, 0xbc, 0xb0, 0xb0, 0xb2, 0xb2,
    0xb3, 0xb0, 0xb0, 0xb0, 0xbc, 0x9c, 0x80, 0x7f,
};
void setup() {
    arduboy.begin();
    arduboy.clear();
}
```



```

void loop() {
  arduboy.clear();
  if(arduboy.pressed(LEFT_BUTTON)) {
    playerx = playerx - 1;
  }
  if(arduboy.pressed(RIGHT_BUTTON)) {
    playerx = playerx + 1;
  }
  if(arduboy.pressed(UP_BUTTON)) {
    playery = playery - 1;
  }
  if(arduboy.pressed(DOWN_BUTTON)) {
    playery = playery + 1;
  }
  //For each column on the screen
  for( int backgroundx = 0; backgroundx < 128; backgroundx = backgroundx
+ 8 ) {
    //For each row in the column
    for( int backgroundy = 0; backgroundy < 64; backgroundy =
backgroundy + 8 ) {
      //Draw a background tile
      arduboy.drawBitmap( backgroundx, backgroundy, background, 8,
8, WHITE );
    }
  }
  //Draw black square
  arduboy.fillRect(playerx, playery, 16, 16, BLACK);
  //Draw player sprite
  arduboy.drawBitmap(playerx, playery, player, 16, 16, WHITE);
  arduboy.display();
}

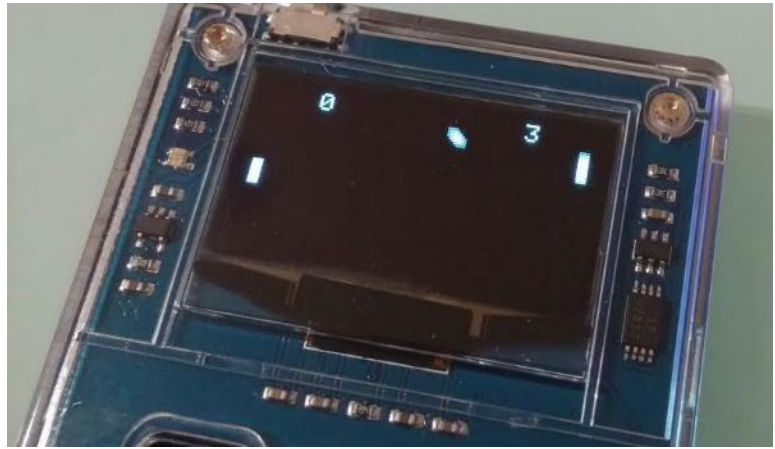
```



# Part 7: Make Pong from Scratch

## This Tutorial

In this tutorial, we're going to learn more about writing our own functions, switches, and basically combining everything we've learned up to this point in order to make a playable version of Pong that you can play against a simple AI.



## Switch

Before we begin, I want to teach you about the `switch` keyword. This one is easy and can really save the amount of code you have to write.

Look at this code for example:

```
if( dude == 0 ) {  
    a = 10;  
    b = 2;  
} else if( dude == 1 ) {  
    c = 3;  
} else if( dude == 2 ) {  
    a = 0;  
    b = 3;  
    c = 4;  
} else if( dude == 3 ) {  
    a = 1;  
} else if( dude == 4 ) {  
    a = 2;  
}
```

Just by looking at this code, you can tell that there are different sections that are executed depending on the value of `dude`. You can re-write this same code using the `switch` keyword. Check this out:

```
switch( dude ) {  
  case 0:  
    a = 10;  
    b = 2;  
    break;  
  case 1:  
    c = 3;  
    break;  
  case 2:  
    a = 0;  
    b = 3;  
    c = 4;  
    break;  
  case 3:  
    a = 1;  
    break;  
  case 4:  
    a = 2;  
    break;  
}
```

The different sections of code are called cases. The value of `dude` is checked and then the right case is executed. For example, if `dude` is equal to 3 the code in the case 3 is run, which is `a = 1`; . Whenever the `break`; instruction is reached, the Arduino jumps out of the `switch`.

Easy, right? We'll use this to structure our game's code.

## Planning Our Game

Whew! We're almost ready to start coding! Let's plan this game out and then start coding.

By breaking down the project into a series of small steps, it helps me realize how much further I have to go as well as keep me motivated to finish the next step. After each major step, I typically test my game to make sure I didn't break anything and that the new mechanics are working properly.

To break down our Pong game, we'll need to follow these steps in development:

1. Set up the new sketch
2. Create a title screen, game screen, win screen, and lose screen
3. Create a ball that bounces around the screen
4. Create and control a paddle

5. Create the computer's paddle
6. Programming collision
7. Adjusting the AI
8. Scoring
9. Creating a new ball and resetting the game

Don't worry if these don't all make sense! I will explain them as we move forward!

## 1. Creating A New Sketch

Open your Arduino IDE and create a new sketch. (**File > New**) In the sketch, like always, we're going to include the Arduboy library, create an Arduboy object, as well as updating our setup()/loop() functions to include the standard display functions. If that sounds like a lot, don't worry! Just make sure your code looks like this!

```
//Jonathan Holmes (crait)
//December 7th, 2016
//A simple Pong clone

#include <Arduboy.h>
Arduboy arduboy;

//Variables declared here

void setup() {
  arduboy.begin();
  //Set-up here
  arduboy.clear();
}

void loop() {
  arduboy.clear();
  //Game code here
  arduboy.display();
}
```

Notice that I added some comments at the top for the project as well as some placeholders. Up until now, this is usually how we started each project, but let's add a little more to this to give us a better starting point for any future projects.

## Framerates

Most game consoles can play games at 30 or 60 frames per second. That means that the screen is updated 30 or 60 times a second. The more instructions that a computer has to run in a second, the slower that framerate can be. If the frame rate is too slow, some games

are unplayable. Have you ever played a game and it played so slowly that it wasn't fun anymore?

During the **setup()** function, we can declare what framerate we want our Arduboy game to be using the **arduboy.setFrameRate()** function. To make the Arduboy game run at 60 frames per second, include this line of code.

```
arduboy.setFrameRate(60);
```

A game slowing down can ruin the gameplay experience, but if a game runs too fast, that can be bad, too! We want the game to run at a constant framerate. In order to do that, add this to the beginning of the **loop()** function.

```
if(!arduboy.nextFrame()) {  
    return;  
}
```

This isn't really important, but what this does is check if it's too early to display the next frame. If it is, the Arduboy will loop again until it's ready. Pretty much making sure that everything runs as smoothly as possible.

## You're ready!

During the **setup()** function, we also want to seed the random number generator so that we can grab some random numbers later. Include the `srand(7/8);` to your **setup()** function.

Okay! You should be ready! Here's the code that you should be able to use at the start of all your projects!

```
//Jonathan Holmes (crait)  
//December 7th, 2016  
//A simple Pong clone  
  
#include <Arduboy.h>  
Arduboy arduboy;  
  
//Variables declared here  
  
void setup() {  
    arduboy.begin();  
    //Seed the random number generator  
    srand(7/8);  
    //Set the game to 60 frames per second  
    arduboy.setFrameRate(60);  
    arduboy.clear();  
}
```

```

void loop() {
  //Prevent the Arduboy from running too fast
  if(!arduboy.nextFrame()) {
    return;
  }
  arduboy.clear();
  //Game code here
  arduboy.display();
}

```

## 2. Structuring The Game

We need to set up a title screen for our game as well as a win and game over screen. We also need a section for our gameplay. These will be called game states. Let's create a variable to keep track of the game state as well as use the `switch` keyword to set up different blocks of code for each state.

Let's start by declaring a variable called `gamestate`.

```

int gamestate = 0;

```

In the area of the `loop()` for the *game code*, let's add the `switch` and check the value of `gamestate`.

```

switch( gamestate ) {
}

```

If `gamestate` is 0, we want the title screen. If it's 1, we want the gameplay state. If it's 2 or 3, then we'll show a win screen or a game over screen. Let's set up those cases:

```

switch( gamestate ) {
  case 0:
    //Title screen
    break;
  case 1:
    //Gameplay screen
    break;
  case 2:
    //Win screen
    break;
  case 3:
    //Game over screen
    break;
}

```

Let's also add text so that we can test this.

```

switch( gamestate ) {
  case 0:
    //Title screen
    arduboy.setCursor(0, 0);
    arduboy.print("Title Screen");
    break;
  case 1:
    //Gameplay screen
    arduboy.setCursor(0, 0);
    arduboy.print("Gameplay");
    break;
  case 2:
    //Win screen
    arduboy.setCursor(0, 0);
    arduboy.print("Win Screen");
    break;
  case 3:
    //Game over screen
    arduboy.setCursor(0, 0);
    arduboy.print("Game Over Screen");
    break;
}

```

Here's the code so far! Test it out and see how it works!

```

//Jonathan Holmes (crait)
//December 7th, 2016
//A simple Pong clone

#include <Arduboy.h>
Arduboy arduboy;

//Variables declared here
int gamestate = 0;

void setup() {
  arduboy.begin();
  //Seed the random number generator
  srand(7/8);
  //Set the game to 60 frames per second
  arduboy.setFrameRate(60);
  arduboy.clear();
}

```

```

void loop() {
    //Prevent the Arduboy from running too fast
    if(!arduboy.nextFrame()) {
        return;
    }
    arduboy.clear();
    //Game code here
    switch( gamestate ) {
        case 0:
            //Title screen
            arduboy.setCursor(0, 0);
            arduboy.print("Title Screen");
            break;
        case 1:
            //Gameplay screen
            arduboy.setCursor(0, 0);
            arduboy.print("Gameplay");
            break;
        case 2:
            //Win screen
            arduboy.setCursor(0, 0);
            arduboy.print("Win Screen");
            break;
        case 3:
            //Game over screen
            arduboy.setCursor(0, 0);
            arduboy.print("Game Over Screen");
            break;
    }
    arduboy.display();
}

```

So, the title screen appears, but nothing else works. Let's add the ability to go to different screens when you push the A button.

Inside of each case, let's check if the A button is pressed, and if it is, then change the value of gamestate. Usually, we'll increase gamestate by 1, but during the last case, we'll set it back to 0 in order to cycle through all of the screens.

```

switch( gamestate ) {
    case 0:
        //Title screen
        arduboy.setCursor(0, 0);
        arduboy.print("Title Screen");
        if(arduboy.pressed(A_BUTTON)) {
            gamestate = 1;
        }
        break;

```



```

case 1:
    //Gameplay screen
    arduboy.setCursor(0, 0);
    arduboy.print("Gameplay");
    if(arduboy.pressed(A_BUTTON)) {
        gamestate = 2;
    }
    break;
case 2:
    //Win screen
    arduboy.setCursor(0, 0);
    arduboy.print("Win Screen");
    if(arduboy.pressed(A_BUTTON)) {
        gamestate = 3;
    }
    break;
case 3:
    //Game over screen
    arduboy.setCursor(0, 0);
    arduboy.print("Game Over Screen");
    if(arduboy.pressed(A_BUTTON)) {
        gamestate = 0;
    }
    break;
}

```

Okay, let's run the completed code to test it!

## Oops! Too Fast!

Does that look familiar? When you test this out, the `gamestate` is changing *too fast*! We went over this in the 5th part of this tutorial series.

We want the Arduboy to know if the A button is being pressed or if it's being held down. If it's held down, If it's being held down, we need to ignore it. That way, the Arduboy won't go through the different cases too fast.

Last time, we used a buffer variable. Let's make a new one in the *variables* area.

```
int justpressed = 0;
```

Whenever we push a button, we'll change `justpressed` to 1. Whenever we let go of the button, we'll change `justpressed` back to 0.

If we are holding down the A button, we'll be pressing down the A button **and** `justpressed` will be 1. On the other hand, the first moment you push the A button, you'll be pressing the A button **and** `justpressed` will be 0.

When we check if the A button is checked...

```
if(arduboy.pressed(A_BUTTON)) {
```

... let's change it to...

```
if(arduboy.pressed(A_BUTTON) and justpressed == 0) {
```

Of course, if we need to add a line right after it to change justpressed.

```
if(arduboy.pressed(A_BUTTON) and justpressed == 0) {  
    justpressed = 1;
```

Replace all of them and after your switch, add this:

```
if(arduboy.notPressed(A_BUTTON)) {  
    justpressed = 0;  
}
```

Here is the example:

```
//Jonathan Holmes (crait)  
//December 7th, 2016  
//A simple Pong clone  
  
#include <Arduboy.h>  
Arduboy arduboy;  
  
//Variables declared here  
int gamestate = 0;  
int justpressed = 0;  
  
void setup() {  
    arduboy.begin();  
    //Seed the random number generator  
    srand(7/8);  
    //Set the game to 60 frames per second  
    arduboy.setFrameRate(60);  
    arduboy.clear();  
}  
  
void loop() {  
    //Prevent the Arduboy from running too fast  
    if(!arduboy.nextFrame()) {  
        return;
```

```

}
arduboy.clear();
//Game code here
switch( gamestate ) {
  case 0:
    //Title screen
    arduboy.setCursor(0, 0);
    arduboy.print("Title Screen");
    //Change the gamestate
    if(arduboy.pressed(A_BUTTON) and justpressed == 0) {
      justpressed = 1;
      gamestate = 1;
    }
    break;
  case 1:
    //Gameplay screen
    arduboy.setCursor(0, 0);
    arduboy.print("Gameplay");
    //Change the gamestate
    if(arduboy.pressed(A_BUTTON) and justpressed == 0) {
      justpressed = 1;
      gamestate = 2;
    }
    break;
  case 2:
    //Win screen
    arduboy.setCursor(0, 0);
    arduboy.print("Win Screen");
    //Change the gamestate
    if(arduboy.pressed(A_BUTTON) and justpressed == 0) {
      justpressed = 1;
      gamestate = 3;
    }
    break;
  case 3:
    //Game over screen
    arduboy.setCursor(0, 0);
    arduboy.print("Game Over Screen");
    //Change the gamestate
    if(arduboy.pressed(A_BUTTON) and justpressed == 0) {
      justpressed = 1;
      gamestate = 0;
    }
    break;
}
//Check if the button is being held down
if(arduboy.notPressed(A_BUTTON)) {
  justpressed = 0;
}

```

```
}  
  arduboy.display();  
}
```

Test out the code and let's finally move on to programming the gameplay section of this game! It's definitely the most fun! Sorry that it's been taking so long to get to!

### 3. Bouncing Ball

Alright, let's create the ball for this game. It'll bounce around the screen. If it hits the top, bottom, or sides of the screen, it'll change direction.

We'll need to declare variables for the ball's location as well as the ball's size.

```
int ballx = 62;  
int bally = 0;  
int ballsize = 4;
```

We'll also create one for the vertical and one for the horizontal direction of on the screen.

```
int ballright = 1;  
int balldown = 1;
```

Whenever `ballright` is 1, we'll move the ball to the right. If `ballright` is -1, we'll move the ball to the left. Likewise, when `balldown` is 1, we'll move the ball down. When `balldown` is -1, we'll move the ball up.

Inside of the gameplay case, we'll start by drawing the ball using the `arduboy.fillRect()` function. Add this line:

```
arduboy.fillRect(ballx, bally, ballsize, ballsize, WHITE);
```

Next, let's handle the movement of the ball horizontally by changing the value of `ballx`.

```
if(ballright == 1) {  
  ballx = ballx + 1;  
}  
if(ballright == -1) {  
  ballx = ballx - 1;  
}
```

Whenever `ballx` is at the left or right edge of the screen, we'll change the value of `ballright`. Where are the left and right edges of the screen? Well, the Arduboy screen is 128 pixels wide, so if `ballx` is 0, the ball is on the left side of the screen. If `ballx` is 127, then it's on the right side of the screen.

```

if(ballx == 0) {
    ballright = 1;
}
if(ballx == 127) {
    ballright = -1;
}

```

This is what your code should look like. Give it a test and see the ball bounce from side to side!

```

//Jonathan Holmes (crait)
//December 7th, 2016
//A simple Pong clone

#include <Arduboy.h>
Arduboy arduboy;

//Variables declared here
int gamestate = 0;
int justpressed = 0;
int ballx = 62;
int bally = 0;
int ballsize = 4;
int ballright = 1;
int balldown = 1;

void setup() {
    arduboy.begin();
    //Seed the random number generator
    srand(7/8);
    //Set the game to 60 frames per second
    arduboy.setFrameRate(60);
    arduboy.clear();
}

void loop() {
    //Prevent the Arduboy from running too fast
    if(!arduboy.nextFrame()) {
        return;
    }
    arduboy.clear();
    //Game code here
    switch( gamestate ) {
        case 0:
            //Title screen
            arduboy.setCursor(0, 0);
            arduboy.print("Title Screen");
            //Change the gamestate

```

```

    if(arduboy.pressed(A_BUTTON) and justpressed == 0) {
        justpressed = 1;
        gamestate = 1;
    }
    break;
case 1:
    //Gameplay screen
    arduboy.setCursor(0, 0);
    arduboy.print("Gameplay");
    //Draw the ball
    arduboy.fillRect(ballx, bally, ballsize, ballsize, WHITE);
    //Move the ball right
    if(ballright == 1) {
        ballx = ballx + 1;
    }
    //Move the ball left
    if(ballright == -1) {
        ballx = ballx - 1;
    }
    //Reflect the ball off of the left side of the screen
    if(ballx == 0) {
        ballright = 1;
    }
    //Reflect the ball off of the right side of the screen
    if(ballx == 127) {
        ballright = -1;
    }
    //Change the gamestate
    if(arduboy.pressed(A_BUTTON) and justpressed == 0) {
        justpressed = 1;
        gamestate = 2;
    }
    break;
case 2:
    //Win screen
    arduboy.setCursor(0, 0);
    arduboy.print("Win Screen");
    //Change the gamestate
    if(arduboy.pressed(A_BUTTON) and justpressed == 0) {
        justpressed = 1;
        gamestate = 3;
    }
    break;
case 3:
    //Game over screen
    arduboy.setCursor(0, 0);
    arduboy.print("Game Over Screen");
    //Change the gamestate

```

```

    if(arduboy.pressed(A_BUTTON) and justpressed == 0) {
        justpressed = 1;
        gamestate = 0;
    }
    break;
}
//Check if the button is being held down
if(arduboy.notPressed(A_BUTTON)) {
    justpressed = 0;
}
arduboy.display();
}

```



Cool! Let's add vertical movement! It's pretty much the exact same thing that we did with the `ballrightvariable`.

When `balldown` is 1, we'll add 1 to `bally`. When `balldown` is -1, we'll subtract 1 from `bally`.

```

if(balldown == 1) {
    bally = bally + 1;
}
if(balldown == -1) {
    bally = bally - 1;
}

```

Of course, we'll also have to reflect it off of the top and bottom of the screen, right? The Arduboy's screen is 64 pixels high, so let's add this code:

```
if(bally == 0) {  
    balldown = 1;  
}  
if(bally == 63) {  
    balldown = -1;  
}
```

That makes sense, right? Good! Here's all the code we have so far! Let's test it out!

```
//Jonathan Holmes (crait)  
//December 7th, 2016  
//A simple Pong clone  
  
#include <Arduboy.h>  
Arduboy arduboy;  
  
//Variables declared here  
int gamestate = 0;  
int justpressed = 0;  
int ballx = 62;  
int bally = 0;  
int ballsize = 4;  
int ballright = 1;  
int balldown = 1;  
  
void setup() {  
    arduboy.begin();  
    //Seed the random number generator  
    srand(7/8);  
    //Set the game to 60 frames per second  
    arduboy.setFrameRate(60);  
    arduboy.clear();  
}  
  
void loop() {  
    //Prevent the Arduboy from running too fast  
    if(!arduboy.nextFrame()) {  
        return;  
    }  
    arduboy.clear();  
    //Game code here  
    switch( gamestate ) {  
        case 0:  
            //Title screen  
            arduboy.setCursor(0, 0);
```



```

arduboy.print("Title Screen");
//Change the gamestate
if(arduboy.pressed(A_BUTTON) and justpressed == 0) {
    justpressed = 1;
    gamestate = 1;
}
break;
case 1:
    //Gameplay screen
    arduboy.setCursor(0, 0);
    arduboy.print("Gameplay");
    //Draw the ball
    arduboy.fillRect(ballx, bally, ballsize, ballsize, WHITE);
    //Move the ball right
    if(ballright == 1) {
        ballx = ballx + 1;
    }
    //Move the ball left
    if(ballright == -1) {
        ballx = ballx - 1;
    }
    //Reflect the ball off of the left side of the screen
    if(ballx == 0) {
        ballright = 1;
    }
    //Reflect the ball off of the right side of the screen
    if(ballx == 127) {
        ballright = -1;
    }
    //Move the ball down
    if(balldown == 1) {
        bally = bally + 1;
    }
    //Move the ball up
    if(balldown == -1) {
        bally = bally - 1;
    }
    //Reflect the ball off of the top of the screen
    if(bally == 0) {
        balldown = 1;
    }
    //Reflect the ball off of the bottom of the screen
    if(bally == 63) {
        balldown = -1;
    }
}

```

```

    //Change the gamestate
    if(arduboy.pressed(A_BUTTON) and justpressed == 0) {
        justpressed = 1;
        gamestate = 2;
    }
    break;
case 2:
    //Win screen
    arduboy.setCursor(0, 0);
    arduboy.print("Win Screen");
    //Change the gamestate
    if(arduboy.pressed(A_BUTTON) and justpressed == 0) {
        justpressed = 1;
        gamestate = 3;
    }
    break;
case 3:
    //Game over screen
    arduboy.setCursor(0, 0);
    arduboy.print("Game Over Screen");
    //Change the gamestate
    if(arduboy.pressed(A_BUTTON) and justpressed == 0) {
        justpressed = 1;
        gamestate = 0;
    }
    break;
}
//Check if the button is being held down
if(arduboy.notPressed(A_BUTTON)) {
    justpressed = 0;
}
arduboy.display();
}

```

There's one more little tiny thing I want use to change before moving on. If you watch the ball carefully as it moves, you'll notice that it moves off of the screen on the bottom and right-side.

The reason it does that is because we change the ball's vertical direction based on the top of the ball hitting the top of the screen or the top of the ball hitting the bottom of the screen as well as checking if the left-side of the ball hits the left-side of the screen or the left-side of the ball hitting the right-side of the screen.

We need to make sure the bottom of the ball bounces against the bottom of the screen and make sure the right-side of the ball bounces off of the right-side of the screen. This is an easy adjustment.

Instead of checking if ballx is equal to 127, we need to check if ballx + ballsize is equal to 127. Same for bally.

Change the reflection cod to...

```
if(ballx + ballsize == 127) {  
    ballright = -1;  
}
```

... and...

```
if(bally + ballsize == 63) {  
    balldown = -1;  
}
```

Run the code on your Arduboy! Mesmorizing, isn't it?

```
//Jonathan Holmes (crait)  
//December 7th, 2016  
//A simple Pong clone  
  
#include <Arduboy.h>  
Arduboy arduboy;  
  
//Variables declared here  
int gamestate = 0;  
int justpressed = 0;  
int ballx = 62;  
int bally = 0;  
int ballsize = 4;  
int ballright = 1;  
int balldown = 1;  
  
void setup() {  
    arduboy.begin();  
    //Seed the random number generator  
    srand(7/8);  
    //Set the game to 60 frames per second  
    arduboy.setFrameRate(60);  
    arduboy.clear();  
}
```

## 4. Creating Your Paddle

This part is my favorite part since it really gives you control over the game. We're going to make your paddle that you can move with the Up and Down buttons!

In the variable section, let's declare these variables... These should be self-explanatory.

```
int paddlewidth = 4;
int paddleheight = 9;
int playerx = 0;
int playery = 0;
```

Back inside of the gameplay section, let's draw the paddle, then allow it to be moved.

```
arduboy.fillRect(playerx, playery, paddlewidth, paddleheight, WHITE);
```

Next, let's add some code to change the value of `playery` with the Up and Down buttons. Luckily, we don't need to worry about coding any buffers for this since we want to be able to hold down the button for movement.

```
if(arduboy.pressed(UP_BUTTON)) {
    playery = playery - 1;
}
if(arduboy.pressed(DOWN_BUTTON)) {
    playery = playery + 1;
}
```

You probably could have figured that part out, but what if we only want to allow a player to move their paddle if they have room to move their paddle? We need to use the `and` keyword.

- If the player presses the Up button and the top of the paddle (`playery`) is greater than (`>`) the top of the screen (`0`), then allow them to change the value of `playery`.
- If the player presses the Down button and the bottom of the paddle (`playery + paddleheight`) is less than (`<`) the bottom of the screen (`127`), then allow them to change the value of `playery`.

```
if(arduboy.pressed(UP_BUTTON) and playery > 0) {
    playery = playery - 1;
}
if(arduboy.pressed(DOWN_BUTTON) and playery + paddleheight < 63) {
    playery = playery + 1;
}
```

That was relatively simple, right? Okay, here's the code you should end up with. Let's run it and test moving your paddle around!

```
//Jonathan Holmes (crait)
//December 7th, 2016
//A simple Pong clone
```

```

#include <Arduboy.h>
Arduboy arduboy;

//Variables declared here
int gamestate = 0;
int justpressed = 0;
int ballx = 62;
int bally = 0;
int ballsize = 4;
int ballright = 1;
int balldown = 1;
int paddlewidth = 4;
int paddleheight = 9;
int playerx = 0;
int playery = 0;

void setup() {
  arduboy.begin();
  //Seed the random number generator
  srand(7/8);
  //Set the game to 60 frames per second
  arduboy.setFrameRate(60);
  arduboy.clear();
}

void loop() {
  //Prevent the Arduboy from running too fast
  if(!arduboy.nextFrame()) {
    return;
  }
  arduboy.clear();
  //Game code here
  switch( gamestate ) {
    case 0:
      //Title screen
      arduboy.setCursor(0, 0);
      arduboy.print("Title Screen");
      //Change the gamestate
      if(arduboy.pressed(A_BUTTON) and justpressed == 0) {
        justpressed = 1;
        gamestate = 1;
      }
      break;
  }

```

```

case 1:
    //Gameplay screen
    arduboy.setCursor(0, 0);
    arduboy.print("Gameplay");
    //Draw the ball
    arduboy.fillRect(ballx, bally, ballsize, ballsize, WHITE);
    //Move the ball right
    if(ballright == 1) {
        ballx = ballx + 1;
    }
    //Move the ball left
    if(ballright == -1) {
        ballx = ballx - 1;
    }
    //Reflect the ball off of the left side of the screen
    if(ballx == 0) {
        ballright = 1;
    }
    //Reflect the ball off of the right side of the screen
    if(ballx + ballsize == 127) {
        ballright = -1;
    }
    //Move the ball down
    if(balldown == 1) {
        bally = bally + 1;
    }
    //Move the ball up
    if(balldown == -1) {
        bally = bally - 1;
    }
    //Reflect the ball off of the top of the screen
    if(bally == 0) {
        balldown = 1;
    }
    //Reflect the ball off of the bottom of the screen
    if(bally + ballsize == 63) {
        balldown = -1;
    }
    //Draw the player's paddle
    arduboy.fillRect(playerx, playery, paddlewidth, paddleheight,
WHITE);
    //If the player presses Up and the paddle is not touching the top of
the screen, move the paddle up
    if(arduboy.pressed(UP_BUTTON) and playery > 0) {
        playery = playery - 1;
    }

```

```

    //If the player presses down and the paddle is not touching the
    bottom of the screen, move the paddle down
    if(arduboy.pressed(DOWN_BUTTON) and playery + paddleheight < 63) {
        playery = playery + 1;
    }
    //Change the gamestate
    if(arduboy.pressed(A_BUTTON) and justpressed == 0) {
        justpressed = 1;
        gamestate = 2;
    }
    break;
case 2:
    //Win screen
    arduboy.setCursor(0, 0);
    arduboy.print("Win Screen");
    //Change the gamestate
    if(arduboy.pressed(A_BUTTON) and justpressed == 0) {
        justpressed = 1;
        gamestate = 3;
    }
    break;
case 3:
    //Game over screen
    arduboy.setCursor(0, 0);
    arduboy.print("Game Over Screen");
    //Change the gamestate
    if(arduboy.pressed(A_BUTTON) and justpressed == 0) {
        justpressed = 1;
        gamestate = 0;
    }
    break;
}
//Check if the button is being held down
if(arduboy.notPressed(A_BUTTON)) {
    justpressed = 0;
}
arduboy.display();
}

```



## 5. Creating The Computer Paddle

Creating the computer's paddle will be very similar to ours, except instead of moving the paddle with buttons, the paddle will move based on the position of the ball.

Let's add these variables near the top of the sketch for the computer paddle:

```
int computery = 0;
int computerx = 127 - paddlewidth;
```

Notice that I set the computer's paddle all the way to the right-side of the screen, then move it over a few pixels to fit it onto the screen.

Down in the gameplay part of the code, let's draw the computer's paddle.

```
arduboy.fillRect(computerx, computery, paddlewidth, paddleheight, WHITE);
```

Great! Now, we need to set the rules for when the computer's paddle moves. We can adjust the AI later, but for now, if the top of the ball (ballx) is higher than the computer's paddle's top (computery), then the computer's paddle should move up. If the bottom of the ball (bally + ballsize) is lower than the computer's paddle's bottom (computery + paddleheight), then we need to move the computer's paddle down.

In code form, it would look like this:

```
if(bally < computery) {
    computery = computery - 1;
}
if(bally + ballsize > computery + paddleheight) {
    computery = computery + 1;
}
```

Cool! That was actually pretty easy! Run the code!

```
//Jonathan Holmes (crait)
//December 7th, 2016
//A simple Pong clone

#include <Arduboy.h>
Arduboy arduboy;

//Variables declared here
int gamestate = 0;
int justpressed = 0;
int ballx = 62;
int bally = 0;
int ballsize = 4;
```



```

int ballright = 1;
int balldown = 1;
int paddlewidth = 4;
int paddleheight = 9;
int playerx = 0;
int playery = 0;
int computerx = 127 - paddlewidth;
int computery = 0;

void setup() {
  arduboy.begin();
  //Seed the random number generator
  srand(7/8);
  //Set the game to 60 frames per second
  arduboy.setFrameRate(60);
  arduboy.clear();
}

void loop() {
  //Prevent the Arduboy from running too fast
  if(!arduboy.nextFrame()) {
    return;
  }
  arduboy.clear();
  //Game code here
  switch( gamestate ) {
    case 0:
      //Title screen
      arduboy.setCursor(0, 0);
      arduboy.print("Title Screen");
      //Change the gamestate
      if(arduboy.pressed(A_BUTTON) and justpressed == 0) {
        justpressed = 1;
        gamestate = 1;
      }
      break;
    case 1:
      //Gameplay screen
      arduboy.setCursor(0, 0);
      arduboy.print("Gameplay");
      //Draw the ball
      arduboy.fillRect(ballx, bally, ballsize, ballsize, WHITE);
      //Move the ball right
      if(ballright == 1) {
        ballx = ballx + 1;
      }
      //Move the ball left
      if(ballright == -1) {

```

```
    ballx = ballx - 1;
}
//Reflect the ball off of the left side of the screen
if(ballx == 0) {
    ballright = 1;
}
//Reflect the ball off of the right side of the screen
if(ballx + ballsize == 127) {
    ballright = -1;
}
//Move the ball down
if(balldown == 1) {
    bally = bally + 1;
}
//Move the ball up
if(balldown == -1) {
    bally = bally - 1;
}
```

```

    //Reflect the ball off of the top of the screen
    if(bally == 0) {
        balldown = 1;
    }
    //Reflect the ball off of the bottom of the screen
    if(bally + ballsize == 63) {
        balldown = -1;
    }
    //Draw the player's paddle
    arduboy.fillRect(playerx, playery, paddlewidth, paddleheight,
WHITE);
    //If the player presses Up and the paddle is not touching the top of
the screen, move the paddle up
    if(arduboy.pressed(UP_BUTTON) and playery > 0) {
        playery = playery - 1;
    }
    //If the player presses down and the paddle is not touching the
bottom of the screen, move the paddle down
    if(arduboy.pressed(DOWN_BUTTON) and playery + paddleheight < 63) {
        playery = playery + 1;
    }
    //Draw the computer's paddle
    arduboy.fillRect(computerx, computery, paddlewidth, paddleheight,
WHITE);
    //If the ball is higher than the computer's paddle, move the
computer's paddle up
    if(bally < computery) {
        computery = computery - 1;
    }
    //If the bottom of the ball is lower than the bottom of the
computer's paddle, move the computer's paddle down
    if(bally + ballsize > computery + paddleheight) {
        computery = computery + 1;
    }

    //Change the gamestate
    if(arduboy.pressed(A_BUTTON) and justpressed == 0) {
        justpressed = 1;
        gamestate = 2;
    }
    break;

```

```

case 2:
    //Win screen
    arduboy.setCursor(0, 0);
    arduboy.print("Win Screen");
    //Change the gamestate
    if(arduboy.pressed(A_BUTTON) and justpressed == 0) {
        justpressed = 1;
        gamestate = 3;
    }
    break;
case 3:
    //Game over screen
    arduboy.setCursor(0, 0);
    arduboy.print("Game Over Screen");
    //Change the gamestate
    if(arduboy.pressed(A_BUTTON) and justpressed == 0) {
        justpressed = 1;
        gamestate = 0;
    }
    break;
}
//Check if the button is being held down
if(arduboy.notPressed(A_BUTTON)) {
    justpressed = 0;
}
arduboy.display();
}

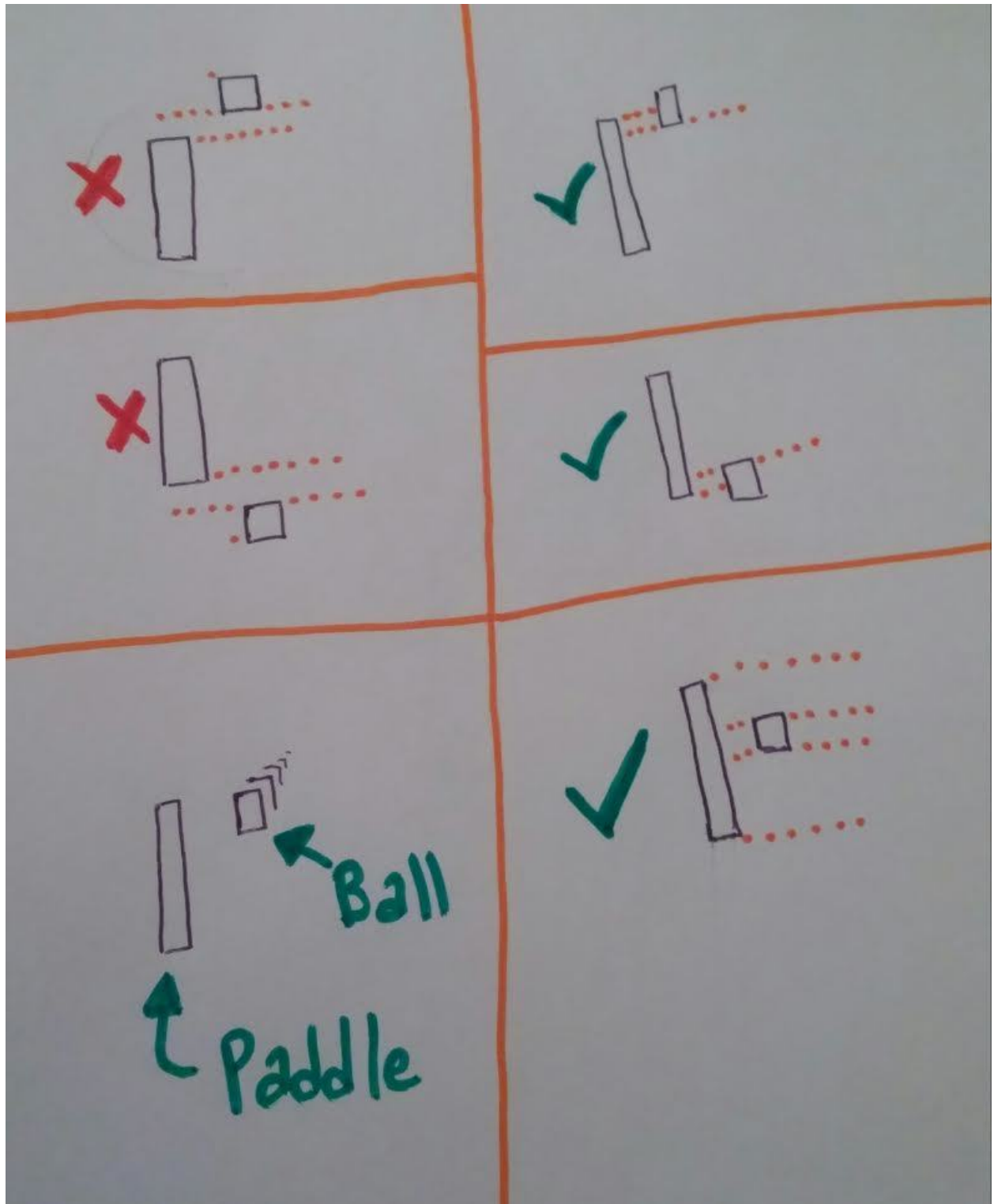
```

Notice how good the AI is. The computer's paddle is *never* miss! It's *too* good! Don't worry! We'll adjust it later to make it possible to beat.

## 6. Collision Checking

What we need to do next is change the ball up some. Instead of bouncing against the right and left side of the screen, we need to make it bounce off of the paddles instead. Once we get the ball bouncing off of the paddles, we'll remove the original bouncing code!

Okay! Let's begin by talking about when the ball will bounce off of the paddle. If the ball is higher than the paddle or lower than the paddle, then it will not bounce. If the ball is near the top, bottom, or middle of the paddle, then we'll change it's direction to bounce it back. Take a look at this beautiful masterpiece I drew for you to help illustrate this.



Let's look at the code for the ball hitting the left side of the screen.

```
if(ballx == 0) {  
    ballright = 1;  
}
```

Instead of checking if the ball's left side (ballx) hits the edge of the screen, we need to do our check where the ball's left side (ballx) hits the player's paddle on the right side. The player's paddle's right is going to be at playerx + paddlewidth.

We now have:

```
if(ballx == playerx + paddlewidth) {  
    ballright = 1;  
}
```

So, now that we know where we need to check along the X-axis, where do we check along the Y-axis?

Well, the paddle will make contact if the top of the paddle (playery) is higher on the screen than the bottom of the ball (bally + ballsize). Not only that, but the bottom of the paddle (playery + paddleheight) has to be lower on the screen than the top of the ball (bally).

```
if(ballx == playerx + paddlewidth and playery < bally + ballsize and  
playery + paddleheight > bally) {  
    ballright = 1;  
}
```

For the computer's paddle, we need to offset ballx by ballsize since we are checking the right-side of the ball. Here's what it would look like:

```
if(ballx + ballsize == computerx and computery < bally + ballsize and  
computery + paddleheight > bally) {  
    ballright = -1;  
}
```

That's it! We're sooo close to being done with this game! Here's the code we have so far, but go ahead and test it out!

```
//Jonathan Holmes (crait)  
//December 7th, 2016  
//A simple Pong clone  
  
#include <Arduboy.h>  
Arduboy arduboy;
```

```

//Variables declared here
int gamestate = 0;
int justpressed = 0;
int ballx = 62;
int bally = 0;
int ballsize = 4;
int ballright = 1;
int balldown = 1;
int paddlewidth = 4;
int paddleheight = 9;
int playerx = 0;
int playery = 0;
int computerx = 127 - paddlewidth;
int computery = 0;

void setup() {
  arduboy.begin();
  //Seed the random number generator
  srand(7/8);
  //Set the game to 60 frames per second
  arduboy.setFrameRate(60);
  arduboy.clear();
}

void loop() {
  //Prevent the Arduboy from running too fast
  if(!arduboy.nextFrame()) {
    return;
  }
  arduboy.clear();
  //Game code here
  switch( gamestate ) {
    case 0:
      //Title screen
      arduboy.setCursor(0, 0);
      arduboy.print("Title Screen");
      //Change the gamestate
      if(arduboy.pressed(A_BUTTON) and justpressed == 0) {
        justpressed = 1;
        gamestate = 1;
      }
      break;
    case 1:
      //Gameplay screen
      arduboy.setCursor(0, 0);
      arduboy.print("Gameplay");
      //Draw the ball
      arduboy.fillRect(ballx, bally, ballsize, ballsize, WHITE);

```

```

//Move the ball right
if(ballright == 1) {
    ballx = ballx + 1;
}
//Move the ball left
if(ballright == -1) {
    ballx = ballx - 1;
}
//Reflect the ball off of the left side of the screen
if(ballx == 0) {
    ballright = 1;
}
//Reflect the ball off of the right side of the screen
if(ballx + ballsize == 127) {
    ballright = -1;
}
//Move the ball down
if(balldown == 1) {
    bally = bally + 1;
}
//Move the ball up
if(balldown == -1) {
    bally = bally - 1;
}
//Reflect the ball off of the top of the screen
if(bally == 0) {
    balldown = 1;
}
//Reflect the ball off of the bottom of the screen
if(bally + ballsize == 63) {
    balldown = -1;
}
//Draw the player's paddle
arduboy.fillRect(playerx, playery, paddlewidth, paddleheight,
WHITE);
//If the player presses Up and the paddle is not touching the top of
the screen, move the paddle up
if(arduboy.pressed(UP_BUTTON) and playery > 0) {
    playery = playery - 1;
}
//If the player presses down and the paddle is not touching the
bottom of the screen, move the paddle down
if(arduboy.pressed(DOWN_BUTTON) and playery + paddleheight < 63) {
    playery = playery + 1;
}
//Draw the computer's paddle
arduboy.fillRect(computerx, computery, paddlewidth, paddleheight,
WHITE);

```



```

    //If the ball is higher than the computer's paddle, move the
computer's paddle up
    if(bally < computery) {
        computery = computery - 1;
    }
    //If the bottom of the ball is lower than the bottom of the
computer's paddle, move the comptuer's paddle down
    if(bally + ballsize > computery + paddleheight) {
        computery = computery + 1;
    }
    //If the ball makes contact with the player's paddle, bounce it back
to the right
    if(ballx == playerx + paddlewidth and playery < bally + ballsize and
playery + paddleheight > bally) {
        ballright = 1;
    }
    //If the ball makes contact with the computer's paddle, bounce it
back to the left
    if(ballx + ballsize == computerx and computery < bally + ballsize
and computery + paddleheight > bally) {
        ballright = -1;
    }
    //Change the gamestate
    if(arduboy.pressed(A_BUTTON) and justpressed == 0) {
        justpressed = 1;
        gamestate = 2;
    }
    break;
case 2:
    //Win screen
    arduboy.setCursor(0, 0);
    arduboy.print("Win Screen");
    //Change the gamestate
    if(arduboy.pressed(A_BUTTON) and justpressed == 0) {
        justpressed = 1;
        gamestate = 3;
    }
    break;
case 3:
    //Game over screen
    arduboy.setCursor(0, 0);
    arduboy.print("Game Over Screen");
    //Change the gamestate
    if(arduboy.pressed(A_BUTTON) and justpressed == 0) {
        justpressed = 1;
        gamestate = 0;
    }
    break;

```

```

}
//Check if the button is being held down
if(arduboy.notPressed(A_BUTTON)) {
  justpressed = 0;
}
arduboy.display();
}

```

## Or Operator

We already talked about using the `and` operator. It's a way we connect two comparisons inside of an `if` statement. To illustrate it, look at this code:

```

if( a == 5 and b > 20 ) {
  c = 0;
}

```

`c` will only be set to 0 if `a` is equal to 5 **and** `b` is greater than 20.

There are some other operators that we can use instead of the `and` operator. One of them is the `or` operator. The `or` operator can be used to check if one of a few comparisons are true. Check out this code:

```

if( a == 5 or b > 20 ) {
  c = 20;
}

```

In the above example, `c` will be set to 20 if either `a` is equal to 5 **or** `b` is greater than 20.

Keep the `or` operator in mind. We'll use it soon!

## Modulo Operation

Do you know the answer to  $12 \div 6$ ? The answer is **2**! That was easy, right? Okay, what about  $23 \div 5$ ? The answer is either **4.6** or **4 with a remainder of 3**. Which is the correct answer??

If you asked the Arduboy what  $23 \div 5$  was, the answer would be **4**! That seems wrong! Where did the remainder go?? We need give the Arduboy another instruction to figure out the remainder.

Remember when I said that computers have to be very precise when given instructions? When computers were young, people tried to come up with the best way to do division. There were a lot of ideas, but the best way seemed to be introducing the **modulo** operator.

To get the remainder, we ask the Arduboy for the answer of **23 % 5**. It will output **3**! If you asked **12 % 6**, you would get **0**!

Why is this important? Well, we can use this to figure out more complex math, but there's something special in game development that we can use this for.

Anytime you use two numbers with the modulo operator, the answer will be somewhere between 0 and the second number. **a % b** will never be a number more than **b**.

Keep this in mind and we'll do something really fun with it later!

## 7. Adjusting The AI

Okay, so, the computer player is way too good. We'll never be able to win. One thing we can do to stop it from being so good is to only allow it to move some of the time. When will we allow it to move its paddle? Well, for starters, we could let it move its paddle only whenever the ball is close to the right side of the screen!

Find these lines of code:

```
if(bally < computery) {  
    computery = computery - 1;  
}  
if(bally + ballsize > computery + paddleheight) {  
    computery = computery + 1;  
}
```

These are the lines of code that move the computer's paddle. Let's put an `if` statement around them that checks if the ball's horizontal position (`ballx`) is close to the right-side of the screen. I put 115.

```
if(ballx > 115) {  
    if(bally < computery) {  
        computery = computery - 1;  
    }  
    if(bally + ballsize > computery + paddleheight) {  
        computery = computery + 1;  
    }  
}
```

Test it out and think about it for a second. Notice how the computer's paddle doesn't always move like it used to? It's a little more precise right now, but only moves whenever the ball is close to scoring.

But, we need to add more variance... More randomness... We need to make it so that the game is different every time you play. I KNOW! Let's let the computer paddle move when

the ball is close **or** let the computer paddle move when the computer picks a random number.

Back in Part 5 of this series, I showed you how to pick a random number. I'm going to explain to you how that works, now.

First of all, we need to know about the `rand()` function. This will give us a random number, but the number can really be anything. It can be 2,000, or 9 or -200. There's really no control over it. HOWEVER, we can use the **modulo operator** that we learned about earlier!! Take a look at this code and think about what this will output!!

```
rand() % 20
```

Well, we'll randomly pick a number, then divide it by 20, then only have the remainder left. It doesn't matter what number is randomly picked, if we just looked at the remainder, the remainder will be some number between 0 and 20. Which number? I don't know. Isn't that awesome?!

That little line of code will essentially pick a random number between 0 and 20!

Let's adjust that `if` statement from above!

```
if(ballx > 115 or rand() % 20 == 1) {
```

This means that if `ballx` is greater than 115, **or** a randomly-picked number is equal to 1, then we move the computer's paddle. Basically, the computer's paddle will randomly move 1/20th of the time.

Here's the full code so far!

```
//Jonathan Holmes (crait)
//December 7th, 2016
//A simple Pong clone

#include <Arduboy.h>
Arduboy arduboy;

//Variables declared here
int gamestate = 0;
int justpressed = 0;
int ballx = 62;
int bally = 0;
int ballsize = 4;
int ballright = 1;
int balldown = 1;
int paddlewidth = 4;
int paddleheight = 9;
```

```

int playerx = 0;
int playery = 0;
int computerx = 127 - paddewidth;
int computery = 0;

void setup() {
  arduboy.begin();
  //Seed the random number generator
  srand(7/8);
  //Set the game to 60 frames per second
  arduboy.setFrameRate(60);
  arduboy.clear();
}

void loop() {
  //Prevent the Arduboy from running too fast
  if(!arduboy.nextFrame()) {
    return;
  }
  arduboy.clear();
  //Game code here
  switch( gamestate ) {
    case 0:
      //Title screen
      arduboy.setCursor(0, 0);
      arduboy.print("Title Screen");
      //Change the gamestate
      if(arduboy.pressed(A_BUTTON) and justpressed == 0) {
        justpressed = 1;
        gamestate = 1;
      }
      break;
    case 1:
      //Gameplay screen
      arduboy.setCursor(0, 0);
      arduboy.print("Gameplay");
      //Draw the ball
      arduboy.fillRect(ballx, bally, ballsize, ballsize, WHITE);
      //Move the ball right
      if(ballright == 1) {
        ballx = ballx + 1;
      }
      //Move the ball left
      if(ballright == -1) {
        ballx = ballx - 1;
      }
  }
}

```

```

//Reflect the ball off of the left side of the screen
if(ballx == 0) {
    ballright = 1;
}
//Reflect the ball off of the right side of the screen
if(ballx + ballsize == 127) {
    ballright = -1;
}
//Move the ball down
if(balldown == 1) {
    bally = bally + 1;
}
//Move the ball up
if(balldown == -1) {
    bally = bally - 1;
}
//Reflect the ball off of the top of the screen
if(bally == 0) {
    balldown = 1;
}
//Reflect the ball off of the bottom of the screen
if(bally + ballsize == 63) {
    balldown = -1;
}
//Draw the player's paddle
arduboy.fillRect(playerx, playery, paddlewidth, paddleheight,
WHITE);
//If the player presses Up and the paddle is not touching the top of
the screen, move the paddle up
if(arduboy.pressed(UP_BUTTON) and playery > 0) {
    playery = playery - 1;
}
//If the player presses down and the paddle is not touching the
bottom of the screen, move the paddle down
if(arduboy.pressed(DOWN_BUTTON) and playery + paddleheight < 63) {
    playery = playery + 1;
}
//Draw the computer's paddle
arduboy.fillRect(computerx, computery, paddlewidth, paddleheight,
WHITE);
//If the ball is close to the edge of the screen or if a random
number out of 20 is equal to 1
if(ballx > 115 or rand() % 20 == 1) {
    //If the ball is higher than the computer's paddle, move the
computer's paddle up

```

```

    if(bally < computery) {
        computery = computery - 1;
    }
    //If the bottom of the ball is lower than the bottom of the
computer's paddle, move the comptuer's paddle down
    if(bally + ballsize > computery + paddleheight) {
        computery = computery + 1;
    }
}

    //If the ball makes contact with the player's paddle, bounce it back
to the right
    if(ballx == playerx + paddlewidth and playery < bally + ballsize and
playery + paddleheight > bally) {
        ballright = 1;
    }
    //If the ball makes contact with the computer's paddle, bounce it
back to the left
    if(ballx + ballsize == computerx and computery < bally + ballsize
and computery + paddleheight > bally) {
        ballright = -1;
    }
    //Change the gamestate
    if(arduboy.pressed(A_BUTTON) and justpressed == 0) {
        justpressed = 1;
        gamestate = 2;
    }
    break;
case 2:
    //Win screen
    arduboy.setCursor(0, 0);
    arduboy.print("Win Screen");
    //Change the gamestate
    if(arduboy.pressed(A_BUTTON) and justpressed == 0) {
        justpressed = 1;
        gamestate = 3;
    }
    break;
case 3:
    //Game over screen
    arduboy.setCursor(0, 0);
    arduboy.print("Game Over Screen");
    //Change the gamestate

```

```

    if(arduboy.pressed(A_BUTTON) and justpressed == 0) {
        justpressed = 1;
        gamestate = 0;
    }
    break;
}
//Check if the button is being held down
if(arduboy.notPressed(A_BUTTON)) {
    justpressed = 0;
}
arduboy.display();
}

```

## 8. Scoring

Since we're going to introduce scoring, we need a way for the ball to actually be scored. This means we need to remove some of our old bouncing code.

```

if(ballx == 0) {
    ballright = 1;
}
if(ballx + ballsize == 127) {
    ballright = -1;
}

```

Find those lines and remove them. This way, the ball will continue to move beyond the screen. In fact, we'll check to see if the ball has moved off of the screen in order to give points!

Feel free to test the code out once you remove that code.

To keep track of the scores, let's declare some variables at the top of the sketch.

```

int playerscore = 0;
int computerscore = 0;

```

Now, look for the code in case 1 that displayed the text, "Gameplay." Instead of just displaying that text, let's modify it to display the player's and computer's scores.

```

arduboy.setCursor(20, 0);
arduboy.print(playerscore);

arduboy.setCursor(101, 0);
arduboy.print(computerscore);

```



Next, we need a way to actually score! Since the code now allows for the ball to move off of the screen, let's check if it's off the screen and if it is, then give someone a point, and then move it back into the middle of the screen.

Here's how we can check if the ball is off of the screen.

```
if(ballx < -10) {  
  
}  
if(ballx > 130) {  
  
}
```

To change the points, and put it back into the middle of the screen, we simply change the appropriate variables...

```
if(ballx < -10) {  
    ballx = 63;  
    computerscore = computerscore + 1;  
}  
if(ballx > 130) {  
    ballx = 63;  
    playerscore = playerscore + 1;  
}
```

I really suggest testing the game out, so far. Here's the code that I have:

```
//Jonathan Holmes (crait)  
//December 7th, 2016  
//A simple Pong clone  
  
#include <Arduboy.h>  
Arduboy arduboy;  
  
//Variables declared here  
int gamestate = 0;  
int justpressed = 0;  
int ballx = 62;  
int bally = 0;  
int ballsize = 4;  
int ballright = 1;  
int balldown = 1;  
int paddlewidth = 4;  
int paddleheight = 9;  
int playerx = 0;  
int playery = 0;  
int computerx = 127 - paddlewidth;  
int computery = 0;
```

```

int playerscore = 0;
int computerscore = 0;

void setup() {
  arduboy.begin();
  //Seed the random number generator
  srand(7/8);
  //Set the game to 60 frames per second
  arduboy.setFrameRate(60);
  arduboy.clear();
}

void loop() {
  //Prevent the Arduboy from running too fast
  if(!arduboy.nextFrame()) {
    return;
  }
  arduboy.clear();
  //Game code here
  switch( gamestate ) {
    case 0:
      //Title screen
      arduboy.setCursor(0, 0);
      arduboy.print("Title Screen");
      //Change the gamestate
      if(arduboy.pressed(A_BUTTON) and justpressed == 0) {
        justpressed = 1;
        gamestate = 1;
      }
      break;
    case 1:
      //Gameplay screen
      //Display the player's score
      arduboy.setCursor(20, 0);
      arduboy.print(playerscore);
      //Display the computer's score
      arduboy.setCursor(101, 0);
      arduboy.print(computerscore);
      //Draw the ball
      arduboy.fillRect(ballx, bally, ballsize, ballsize, WHITE);
      //Move the ball right
      if(ballright == 1) {
        ballx = ballx + 1;
      }
      //Move the ball left
      if(ballright == -1) {
        ballx = ballx - 1;
      }
  }
}

```

```

    //Move the ball down
    if(balldown == 1) {
        bally = bally + 1;
    }
    //Move the ball up
    if(balldown == -1) {
        bally = bally - 1;
    }
    //Reflect the ball off of the top of the screen
    if(bally == 0) {
        balldown = 1;
    }
    //Reflect the ball off of the bottom of the screen
    if(bally + ballsize == 63) {
        balldown = -1;
    }
    //Draw the player's paddle
    arduboy.fillRect(playerx, playery, paddlewidth, paddleheight,
WHITE);
    //If the player presses Up and the paddle is not touching the top of
the screen, move the paddle up
    if(arduboy.pressed(UP_BUTTON) and playery > 0) {
        playery = playery - 1;
    }
    //If the player presses down and the paddle is not touching the
bottom of the screen, move the paddle down
    if(arduboy.pressed(DOWN_BUTTON) and playery + paddleheight < 63) {
        playery = playery + 1;
    }
    //Draw the computer's paddle
    arduboy.fillRect(computerx, computery, paddlewidth, paddleheight,
WHITE);
    //If the ball is close to the edge of the screen or if a random
number out of 20 is equal to 1
    if(ballx > 115 or rand() % 20 == 1) {
        //If the ball is higher than the computer's paddle, move the
computer's paddle up
        if(bally < computery) {
            computery = computery - 1;
        }
        //If the bottom of the ball is lower than the bottom of the
computer's paddle, move the comptuer's paddle down
        if(bally + ballsize > computery + paddleheight) {
            computery = computery + 1;
        }
    }
}

```

```

//If the ball moves off of the screen to the left...
if(ballx < -10) {
    //Move the ball back to the middle of the screen
    ballx = 63;
    //Give the computer a point
    computerscore = computerscore + 1;
}
//If the ball moves off of the screen to the right....
if(ballx > 130) {
    //Move the ball back to the middle of the screen
    ballx = 63;
    //Give the player a point
    playerscore = playerscore + 1;
}

//If the ball makes contact with the player's paddle, bounce it back
to the right
if(ballx == playerx + paddlewidth and playery < bally + ballsize and
playery + paddleheight > bally) {
    ballright = 1;
}
//If the ball makes contact with the computer's paddle, bounce it
back to the left
if(ballx + ballsize == computerx and computery < bally + ballsize
and computery + paddleheight > bally) {
    ballright = -1;
}
//Change the gamestate
if(arduboy.pressed(A_BUTTON) and justpressed == 0) {
    justpressed = 1;
    gamestate = 2;
}
break;
case 2:
    //Win screen
    arduboy.setCursor(0, 0);
    arduboy.print("Win Screen");
    //Change the gamestate
    if(arduboy.pressed(A_BUTTON) and justpressed == 0) {
        justpressed = 1;
        gamestate = 3;
    }
    break;

```

```

    case 3:
        //Game over screen
        arduboy.setCursor(0, 0);
        arduboy.print("Game Over Screen");
        //Change the gamestate
        if(arduboy.pressed(A_BUTTON) and justpressed == 0) {
            justpressed = 1;
            gamestate = 0;
        }
        break;
    }
    //Check if the button is being held down
    if(arduboy.notPressed(A_BUTTON)) {
        justpressed = 0;
    }
    arduboy.display();
}

```

Awesome, huh? It's pretty much a finished game at this point! We just need to clean it up, some!

One thing we need to do is check if someone wins! Let's do a check to see if someone has 5 points. If they do, then change the gamestate variable to the appropriate value.

```

if(playerscore == 5) {
    gamestate = 2;
}
if(computerscore == 5) {
    gamestate = 3;
}

```

Hmm... Maybe we should change the code inside of case 2 for pushing the A button. Instead of taking you to gamestate = 3, maybe we should use gamestate = 0 so that it takes you to the title screen of the game. Why would we want to show the win screen, then game over screen right afterwards, right?

Here's the completed code so far:

```

//Jonathan Holmes (crait)
//December 7th, 2016
//A simple Pong clone

#include <Arduboy.h>
Arduboy arduboy;

```

```

//Variables declared here
int gamestate = 0;
int justpressed = 0;
int ballx = 62;
int bally = 0;
int ballsize = 4;
int ballright = 1;
int balldown = 1;
int paddlewidth = 4;
int paddleheight = 9;
int playerx = 0;
int playery = 0;
int computerx = 127 - paddlewidth;
int computery = 0;
int playerscore = 0;
int computerscore = 0;

void setup() {
  arduboy.begin();
  //Seed the random number generator
  srand(7/8);
  //Set the game to 60 frames per second
  arduboy.setFrameRate(60);
  arduboy.clear();
}

void loop() {
  //Prevent the Arduboy from running too fast
  if(!arduboy.nextFrame()) {
    return;
  }
  arduboy.clear();
  //Game code here
  switch( gamestate ) {
    case 0:
      //Title screen
      arduboy.setCursor(0, 0);
      arduboy.print("Title Screen");
      //Change the gamestate
      if(arduboy.pressed(A_BUTTON) and justpressed == 0) {
        justpressed = 1;
        gamestate = 1;
      }
      break;
  }
}

```

```

case 1:
    //Gameplay screen
    //Display the player's score
    arduboy.setCursor(20, 0);
    arduboy.print(playerscore);
    //Display the computer's score
    arduboy.setCursor(101, 0);
    arduboy.print(computerscore);
    //Draw the ball
    arduboy.fillRect(ballx, bally, ballsize, ballsize, WHITE);
    //Move the ball right
    if(ballright == 1) {
        ballx = ballx + 1;
    }
    //Move the ball left
    if(ballright == -1) {
        ballx = ballx - 1;
    }
    //Move the ball down
    if(balldown == 1) {
        bally = bally + 1;
    }
    //Move the ball up
    if(balldown == -1) {
        bally = bally - 1;
    }
    //Reflect the ball off of the top of the screen
    if(bally == 0) {
        balldown = 1;
    }
    //Reflect the ball off of the bottom of the screen
    if(bally + ballsize == 63) {
        balldown = -1;
    }
    //Draw the player's paddle
    arduboy.fillRect(playerx, playery, paddlewidth, paddleheight,
WHITE);
    //If the player presses Up and the paddle is not touching the top of
the screen, move the paddle up
    if(arduboy.pressed(UP_BUTTON) and playery > 0) {
        playery = playery - 1;
    }
    //If the player presses down and the paddle is not touching the
bottom of the screen, move the paddle down
    if(arduboy.pressed(DOWN_BUTTON) and playery + paddleheight < 63) {
        playery = playery + 1;
    }

```

```

    //Draw the computer's paddle
    arduboy.fillRect(computerx, computery, paddlewidth, paddleheight,
WHITE);
    //If the ball is close to the edge of the screen or if a random
number out of 20 is equal to 1
    if(ballx > 115 or rand() % 20 == 1) {
        //If the ball is higher than the computer's paddle, move the
computer's paddle up
        if(bally < computery) {
            computery = computery - 1;
        }
        //If the bottom of the ball is lower than the bottom of the
computer's paddle, move the computer's paddle down
        if(bally + ballsize > computery + paddleheight) {
            computery = computery + 1;
        }
    }
    //If the ball moves off of the screen to the left...
    if(ballx < -10) {
        //Move the ball back to the middle of the screen
        ballx = 63;
        //Give the computer a point
        computerscore = computerscore + 1;
    }
    //If the ball moves off of the screen to the right....
    if(ballx > 130) {
        //Move the ball back to the middle of the screen
        ballx = 63;
        //Give the player a point
        playerscore = playerscore + 1;
    }
    //Check if the player wins
    if(playerscore == 5) {
        gamestate = 2;
    }
    //Check if the computer wins
    if(computerscore == 5) {
        gamestate = 3;
    }
}

```



```

    //If the ball makes contact with the player's paddle, bounce it back
    to the right
    if(ballx == playerx + paddlewidth and playery < bally + ballsize and
    playery + paddleheight > bally) {
        ballright = 1;
    }
    //If the ball makes contact with the computer's paddle, bounce it
    back to the left
    if(ballx + ballsize == computerx and computery < bally + ballsize
    and computery + paddleheight > bally) {
        ballright = -1;
    }
    //Change the gamestate
    if(arduboy.pressed(A_BUTTON) and justpressed == 0) {
        justpressed = 1;
        gamestate = 2;
    }
    break;
case 2:
    //Win screen
    arduboy.setCursor(0, 0);
    arduboy.print("Win Screen");
    //Change the gamestate
    if(arduboy.pressed(A_BUTTON) and justpressed == 0) {
        justpressed = 1;
        gamestate = 0;
    }
    break;
case 3:
    //Game over screen
    arduboy.setCursor(0, 0);
    arduboy.print("Game Over Screen");
    //Change the gamestate
    if(arduboy.pressed(A_BUTTON) and justpressed == 0) {
        justpressed = 1;
        gamestate = 0;
    }
    break;
}
//Check if the button is being held down
if(arduboy.notPressed(A_BUTTON)) {
    justpressed = 0;
}
arduboy.display();
}

```

## 9. Reset Function

If you tested this out, you're going to notice that once you win and try to play again, you won't really be able to. The reason this happens is that the Arduboy *remembers* the score from the last match! It starts the new match of Pong, checks the scores, and realizes someone has already won, then shows you either the win screen or game over screen.

What we need to do is reset the game's variables... `ballx`, `playerscore`, and `computerscore`.

Scroll all the way down to `case 2` and `case 3`. Inside of both of them, there's a check to see if we push the A button. This is a great place to put the code to reset our variables since pushing A is supposed to reset the game to the very beginning.

So, inside of the button checks of `case 2` and `case 3`, put this code:

```
ballx = 63;
playerscore = 0;
computerscore = 0;
```

Actually! I got an idea! Instead of writing the same code multiple times, I should teach you about writing functions to save you time!

You already kinda know what functions are, right? They're instructions that the computer follows. One example of this is `arduboy.clear()`, which will erase everything on the screen!

Well, one cool thing about programming is that you can write **your own** functions! It's really helpful! Sometimes, you'll have code that you want to use over and over again, but maybe you don't want to type over and over again...

Scroll all the way up... After you declare your variables, let's declare a function. We'll call it `resetgame()`. Type this out:

```
void resetgame() {
}
```

Inside of the braces, we'll be able to put the instructions that are contained within the `resetgame()` function. Let's update it with the code we used a moment ago!

```
void resetgame() {
  ballx = 63;
  playerscore = 0;
  computerscore = 0;
}
```

There we go! Any time we want to run those instructions, instead of typing out all 3 of those lines, we can save time by typing `resetgame()` instead!

Back down near the bottom, add the `resetgame()` function where you change `gamestate` to 0!

If you got confused, here's the full code!

```
//Jonathan Holmes (crait)
//December 7th, 2016
//A simple Pong clone

#include <Arduboy.h>
Arduboy arduboy;

//Variables declared here
int gamestate = 0;
int justpressed = 0;
int ballx = 62;
int bally = 0;
int ballsize = 4;
int ballright = 1;
int balldown = 1;
int paddlewidth = 4;
int paddleheight = 9;
int playerx = 0;
int playery = 0;
int computerx = 127 - paddlewidth;
int computery = 0;
int playerscore = 0;
int computerscore = 0;

void resetgame() {
    ballx = 63;
    playerscore = 0;
    computerscore = 0;
}

void setup() {
    arduboy.begin();
    //Seed the random number generator
    srand(7/8);
    //Set the game to 60 frames per second
    arduboy.setFrameRate(60);
    arduboy.clear();
}
```

```

void loop() {
    //Prevent the Arduboy from running too fast
    if(!arduboy.nextFrame()) {
        return;
    }
    arduboy.clear();
    //Game code here
    switch( gamestate ) {
        case 0:
            //Title screen
            arduboy.setCursor(0, 0);
            arduboy.print("Title Screen");
            //Change the gamestate
            if(arduboy.pressed(A_BUTTON) and justpressed == 0) {
                justpressed = 1;
                gamestate = 1;
            }
            break;
        case 1:
            //Gameplay screen
            //Display the player's score
            arduboy.setCursor(20, 0);
            arduboy.print(playerscore);
            //Display the computer's score
            arduboy.setCursor(101, 0);
            arduboy.print(computerscore);
            //Draw the ball
            arduboy.fillRect(ballx, bally, ballsize, ballsize, WHITE);
            //Move the ball right
            if(ballright == 1) {
                ballx = ballx + 1;
            }
            //Move the ball left
            if(ballright == -1) {
                ballx = ballx - 1;
            }
            //Move the ball down
            if(balldown == 1) {
                bally = bally + 1;
            }
            //Move the ball up
            if(balldown == -1) {
                bally = bally - 1;
            }
            //Reflect the ball off of the top of the screen
            if(bally == 0) {
                balldown = 1;
            }
    }
}

```

```

    //Reflect the ball off of the bottom of the screen
    if(bally + ballsize == 63) {
        balldown = -1;
    }
    //Draw the player's paddle
    arduboy.fillRect(playerx, playery, paddlewidth, paddleheight,
WHITE);
    //If the player presses Up and the paddle is not touching the top of
the screen, move the paddle up
    if(arduboy.pressed(UP_BUTTON) and playery > 0) {
        playery = playery - 1;
    }
    //If the player presses down and the paddle is not touching the
bottom of the screen, move the paddle down
    if(arduboy.pressed(DOWN_BUTTON) and playery + paddleheight < 63) {
        playery = playery + 1;
    }
    //Draw the computer's paddle
    arduboy.fillRect(computerx, computery, paddlewidth, paddleheight,
WHITE);
    //If the ball is close to the edge of the screen or if a random
number out of 20 is equal to 1
    if(ballx > 115 or rand() % 20 == 1) {
        //If the ball is higher than the computer's paddle, move the
computer's paddle up
        if(bally < computery) {
            computery = computery - 1;
        }
        //If the bottom of the ball is lower than the bottom of the
computer's paddle, move the comptuer's paddle down
        if(bally + ballsize > computery + paddleheight) {
            computery = computery + 1;
        }
    }
    //If the ball moves off of the screen to the left...
    if(ballx < -10) {
        //Move the ball back to the middle of the screen
        ballx = 63;
        //Give the computer a point
        computerscore = computerscore + 1;
    }
    //If the ball moves off of the screen to the right....
    if(ballx > 130) {
        //Move the ball back to the middle of the screen
        ballx = 63;
        //Give the player a point
        playerscore = playerscore + 1;
    }
}

```

```

    //Check if the player wins
    if(playerscore == 5) {
        gamestate = 2;
    }
    //Check if the computer wins
    if(computerscore == 5) {
        gamestate = 3;
    }

    //If the ball makes contact with the player's paddle, bounce it back
    to the right
    if(ballx == playerx + paddlewidth and playery < bally + ballsize and
    playery + paddleheight > bally) {
        ballright = 1;
    }
    //If the ball makes contact with the computer's paddle, bounce it
    back to the left
    if(ballx + ballsize == computerx and computery < bally + ballsize
    and computery + paddleheight > bally) {
        ballright = -1;
    }
    //Change the gamestate
    if(arduboy.pressed(A_BUTTON) and justpressed == 0) {
        justpressed = 1;
        gamestate = 2;
    }
    break;
case 2:
    //Win screen
    arduboy.setCursor(0, 0);
    arduboy.print("Win Screen");
    //Change the gamestate
    if(arduboy.pressed(A_BUTTON) and justpressed == 0) {
        justpressed = 1;
        resetgame();
        gamestate = 0;
    }
    break;

```

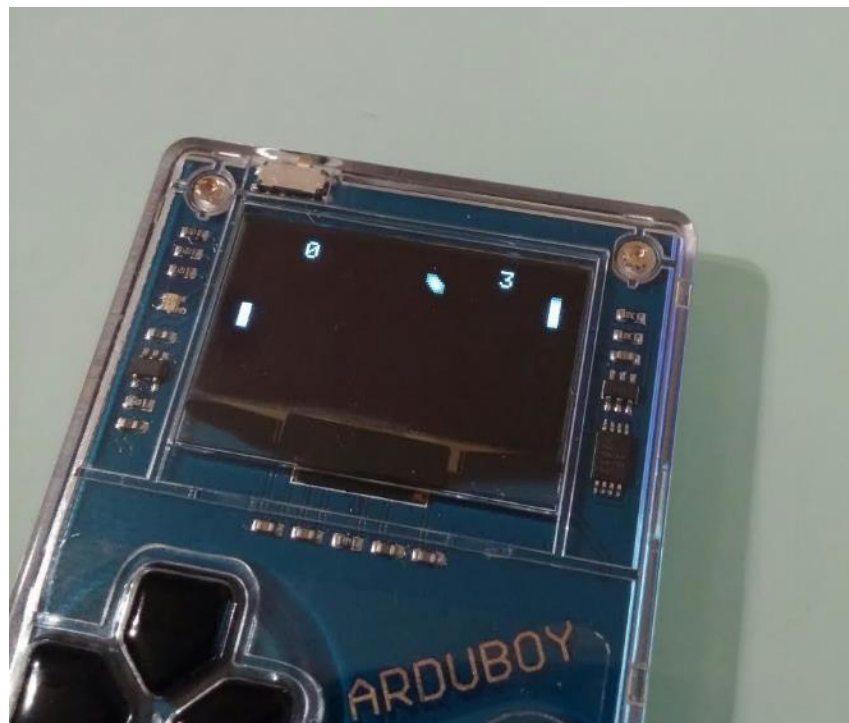
```

case 3:
    //Game over screen
    arduboy.setCursor(0, 0);
    arduboy.print("Game Over Screen");
    //Change the gamestate
    if(arduboy.pressed(A_BUTTON) and justpressed == 0) {
        justpressed = 1;
        resetgame();
        gamestate = 0;
    }
    break;
}
//Check if the button is being held down
if(arduboy.notPressed(A_BUTTON)) {
    justpressed = 0;
}
arduboy.display();
}

```

## Wow!

Give yourself a pat on the back! Good job with this one! It was a long tutorial, but if you made it this far, you really deserve a prize! You are ready to start experimenting and making your own game!



# Part 8: Starting Dinosmasher

## This Tutorial



This tutorial will be the first part of making a more complicated game than Pong. You're going to make a game to challenge a player to find and destroy as many buildings as they can as a dinosaur in a given time limit, getting points as they do so. First, we need to learn about arrays and a few other things.

This is a HUGE tutorial, so I'm going to be breaking this down into smaller steps. Here's what we're going to cover in Part 8.

1. Upgrading our library to Arduboy2
2. Creating our sketch
3. Learning about `#define`
4. Structuring our code
5. A moment about button presses
6. Creating a 1D array
7. Creating a 2D array



## 0. Upgrading our library to Arduboy2

So, I wrote the previous tutorials quite some time ago. Because of this, I used the original Arduboy library. I'm going to show you how to install the Arduboy2 library. If you are sure you have done this, you can skip this step.

Remember what I said back in tutorial 2... A library is a collection of instructions that someone else wrote that we can use that makes writing games easier.

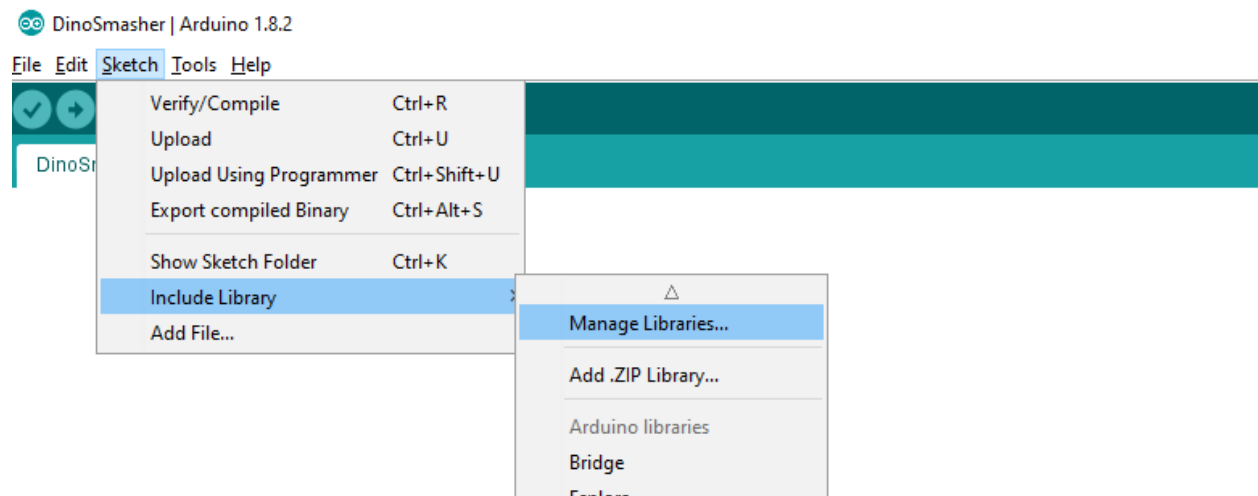
To use the original library, we were using the following in our code:

```
#include <Arduboy.h>
Arduboy arduboy;
```

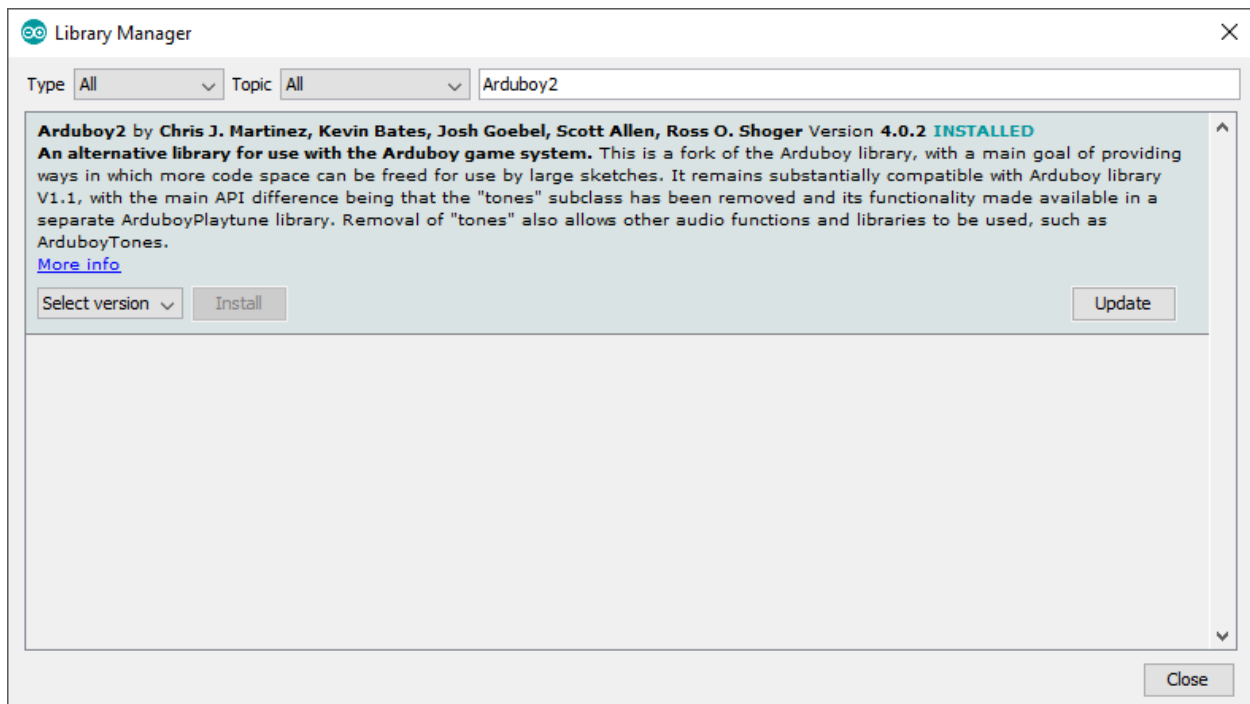
However, we want to use the updated Arduboy2 library, which was created by [@MLXXXp](#). His Arduboy2 library includes more functionality that we'll be using that the original does not include. Instead of the above, we'll need to use the following:

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

We need to also download the library to your computer through the Arduino IDE.



In the Arduino IDE, go to **Sketch > Include Library > Manage Libraries** .



Simply search for **Arduboy2** and select the latest version, then click **Install** .

## 1. Creating our sketch

Okay, let's start working on the game! In the Arduino IDE, go to **File > New**, then save the file to your computer. In the IDE, let's use all this.

```
//DinoSmasher

#include <Arduboy2.h>
Arduboy2 arduboy;

void setup() {
  arduboy.begin();
  arduboy.setFrameRate(45);
  arduboy.display();

  arduboy.initRandomSeed();

  arduboy.clear();
}
```

```

void loop() {
  if(!(arduboy.nextFrame())) {
    return;
  }

  arduboy.pollButtons();

  arduboy.clear();

  arduboy.setCursor(0, 0);
  arduboy.print(F("HELLO!"));

  arduboy.display();
}

```

You should be able to understand most of this, but if there are some new things that you don't recognize, don't worry! Just save the file and transfer it to your Arduboy. It should simply say "HELLO".

## 2. Learning about #define

There are a few reasons to use functions in our code. One reason is organize your code, but another is to prevent you from having to repeat a lot of the same code over and over again. There's another cool feature that helps with both of those things that uses the #define directive.

Pretend like you have the following code in your game to move your character around:

```

void move_left() {
  player_x -= 5;
}
void move_right() {
  player_x += 5;
}
void move_up() {
  player_y -= 5;
}
void move_down() {
  player_y += 5;
}

```

If you used this code and found that your character moves too fast and wanted to change its speed, you would have to change it in 4 places. Instead, we could use the #define directive to create a keyword called SPEED, set it to 5 and then use that.

```

#define SPEED 5

void move_left() {
    player_x -= SPEED;
}
void move_right() {
    player_x += SPEED;
}
void move_up() {
    player_y -= SPEED;
}
void move_down() {
    player_y += SPEED;
}

```

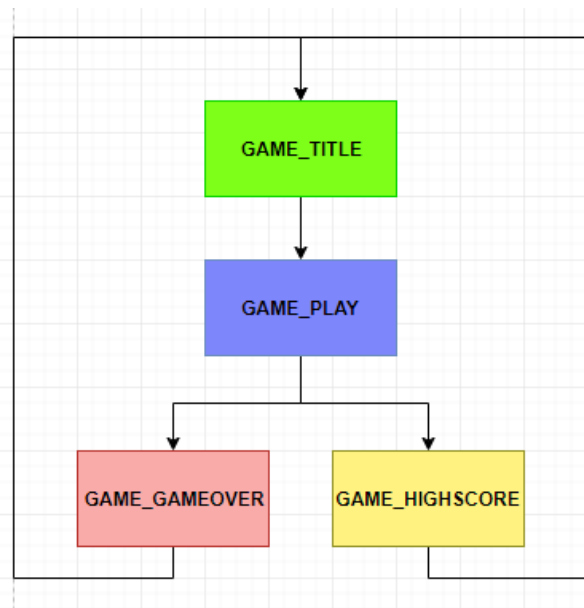
The way this works is that whenever you compile your code, the compiler will replace `SPEED` with `5` before it compiles.

Let's use this knowledge to change our code!

### 3. Structuring our code

Our Pong code started to get a little crazy, so let's make sure this game's code is a bit easier to read. We have already have the standard `loop()` function that's called many times. In our Pong game, that's where we had a switch statement that handled which part of the game was happening.

We're going to have to use a switch statement, as well, but let's put it into its own function. Let's make a `gameplay()` function a reference it near the end of our `loop()` function instead of printing "*HELLO*". Remember, we want use `arduboy.display()` last, so let's make sure it comes right before that.



When someone turns on the Arduboy, we want it to start at the title screen, then go to the gameplay screen, and then if they lose, we want them to go to the game over screen. However, if they beat the saved high score, we want to take them to the high score screen. From either of those, we want to go back to the title screen.

We need to create an `int` variable called `gamestate` to keep track of which state we're in. The switch case should look something like this:

```
switch(gamestate) {  
  case 0:  
    //Title screen stuff  
    break;  
  
  case 1:  
    //Gameplay stuff  
    break;  
  
  case 2:  
    //Game over screen  
    break;  
  
  case 3:  
    //High score screen  
    break;  
}
```

Putting that into our `gameloop()` function, our code now looks like this:

```
//DinoSmasher  
  
#include <Arduboy2.h>  
Arduboy2 arduboy;  
  
int gamestate = 0;  
  
void gameloop() {  
  switch(gamestate) {  
    case 0:  
      //Title screen stuff  
      break;  
  
    case 1:  
      //Gameplay stuff  
      break;  
  
    case 2:  
      //Game over screen  
      break;  
  }
```

```

    case 3:
        //High score screen
        break;
    }
}

void setup() {
    arduboy.begin();
    arduboy.setFrameRate(45);
    arduboy.display();

    arduboy.initRandomSeed();

    arduboy.clear();
}

void loop() {
    if(!(arduboy.nextFrame())) {
        return;
    }

    arduboy.pollButtons();

    arduboy.clear();

    gameloop();

    arduboy.display();
}

```

COOL!

Instead of having 0, 1, 2, and 3 represent the different states in our game, let's use `#define` to make the code easier to digest.

```

#define GAME_TITLE 0
#define GAME_PLAY 1
#define GAME_OVER 2
#define GAME_HIGH 3

```

Now, any time we want to check the value of `gamestate`, we can simply compare `gamestate` to one of these definitions! We can even use it when defining/initializing variables!

Take a look at how I've done it below:

```
//DinoSmasher

#include <Arduboy2.h>
Arduboy2 arduboy;

#define GAME_TITLE 0
#define GAME_PLAY 1
#define GAME_OVER 2
#define GAME_HIGH 3
int gamestate = GAME_TITLE;

void gameloop() {
    switch(gamestate) {
        case GAME_TITLE:
            //Title screen stuff
            break;

        case GAME_PLAY:
            //Gameplay stuff
            break;

        case GAME_OVER:
            //Game over screen
            break;

        case GAME_HIGH:
            //High score screen
            break;
    }
}

void setup() {
    arduboy.begin();
    arduboy.setFrameRate(45);
    arduboy.display();
    arduboy.initRandomSeed();
    arduboy.clear();
}
```

```

void loop() {
  if(!(arduboy.nextFrame())) {
    return;
  }

  arduboy.pollButtons();
  arduboy.clear();
  gameloop();
  arduboy.display();
}

```

If we compile this, the code would run, but nothing would be displayed on the Arduboy screen. We should add some text to the screen to make sure this compiles and can be put onto the screen! However, I want to emphasize that we are going to make sure this program is easy to read, so let's make sure we add it in sensibly!

Instead of putting a bunch of code into our `switch` statements, let's simply add a single function to each case. These functions will be `titlescreen()`, `gameplay()`, `gameoverscreen()`, and `highscorescreen()`. Any time we want to add code into one of those states, we'll simply put the code into those functions! This means we'll add the text to those functions. Here's the code for that:

```

//DinoSmasher

#include <Arduboy2.h>
Arduboy2 arduboy;

#define GAME_TITLE 0
#define GAME_PLAY 1
#define GAME_OVER 2
#define GAME_HIGH 3
int gamestate = GAME_TITLE;

void titlescreen() {
  arduboy.setCursor(0, 0);
  arduboy.print("Title Screen\n");
}

void gameplay() {
  arduboy.setCursor(0, 0);
  arduboy.print("Gameplay\n");
}

void gameoverscreen() {
  arduboy.setCursor(0, 0);
  arduboy.print("Game Over Screen\n");
}

```



```

void highscorescreen() {
  arduboy.setCursor(0, 0);
  arduboy.print("High Score Screen\n");
}

void gameloop() {
  switch(gamestate) {
    case GAME_TITLE:
      titlescreen();
      break;

    case GAME_PLAY:
      gameplay();
      break;

    case GAME_OVER:
      gameoverscreen();
      break;

    case GAME_HIGH:
      highscorescreen();
      break;
  }
}

void setup() {
  arduboy.begin();
  arduboy.setFrameRate(45);
  arduboy.display();

  arduboy.initRandomSeed();

  arduboy.clear();
}

void loop() {
  if(!(arduboy.nextFrame())) {
    return;
  }

  arduboy.pollButtons();
  arduboy.clear();
  gameloop();
  arduboy.display();
}

```

ALRIGHTY! Compile this and put it onto your Arduboy to see what happens!

## 4. A moment about button presses

Up until now, you've been learning about button presses with the first Arduboy library. The Arduboy2 library offers us a more simple way to handle button presses. We are able to check buttons 4 different ways:

1. The button is currently being held down using `arduboy.pressed()` like before
2. The button was just pressed in using `arduboy.justPressed()`
3. The button was just released using `arduboy.justReleased()`
4. A button is not being pressed using `arduboy.notPressed()`

The library automatically handles what we had been programming in all along! In order to check if a button was pressed this frame, but don't repeat, we can use `arduboy.justPressed()`. Changing the gamestate with the A button with something like the following would be easy.

```
if(arduboy.justPressed(A_BUTTON)) {  
    gamestate = GAME_PLAY;  
}
```

Try adding code like this to `titlescreen()`, `gameplay()`, `gameoverscreen()`, and `highscorescreen()` to cycle through all of the states. If you want to double-check your work, see my code below:

```
//DinoSmasher  
  
#include <Arduboy2.h>  
Arduboy2 arduboy;  
  
#define GAME_TITLE 0  
#define GAME_PLAY 1  
#define GAME_OVER 2  
#define GAME_HIGH 3  
int gamestate = GAME_TITLE;  
  
void titlescreen() {  
    arduboy.setCursor(0, 0);  
    arduboy.print("Title Screen\n");  
    if(arduboy.justPressed(A_BUTTON)) {  
        gamestate = GAME_PLAY;  
    }  
}
```

```
void gameplay() {
    arduboy.setCursor(0, 0);
    arduboy.print("Gameplay\n");
    if(arduboy.justPressed(A_BUTTON)) {
        gamestate = GAME_OVER;
    }
}

void gameoverscreen() {
    arduboy.setCursor(0, 0);
    arduboy.print("Game Over Screen\n");
    if(arduboy.justPressed(A_BUTTON)) {
        gamestate = GAME_HIGH;
    }
}

void highscorescreen() {
    arduboy.setCursor(0, 0);
    arduboy.print("High Score Screen\n");
    if(arduboy.justPressed(A_BUTTON)) {
        gamestate = GAME_TITLE;
    }
}

void gameloop() {
    switch(gamestate) {
        case GAME_TITLE:
            titlescreen();
            break;

        case GAME_PLAY:
            gameplay();
            break;

        case GAME_OVER:
            gameoverscreen();
            break;

        case GAME_HIGH:
            highscorescreen();
            break;
    }
}
```

```

void setup() {
  arduboy.begin();
  arduboy.setFrameRate(45);
  arduboy.display();
  arduboy.initRandomSeed();
  arduboy.clear();
}

void loop() {
  if(!(arduboy.nextFrame())) {
    return;
  }

  arduboy.pollButtons();
  arduboy.clear();
  gameloop();
  arduboy.display();
}

```

I suggest running this on your Arduboy and seeing how the states do not flicker too fast.

(OH, YEAH! Did you notice that `A_BUTTON` is a defined keyword?! It's just a number!)

## 5. Creating a 1D array

Remember back in Part 6 when I told you about arrays? I told you that they were just groups of variables. Let's talk a little more about them. They can be super helpful!

Let's pretend that you have 5 cars in your game and you wanted to store how fast each one can drive. You *could* do something like this.

```

int car_one = 4;
int car_two = 5;
int car_three = 7;
int car_four = 8;
int car_five = 3;

```

But what if you wanted to add 5 more cars?? That's a lot of typing!! Well, there's a quicker way to write this with arrays!

```

int cars[5] = { 4, 5, 7, 8, 3 };

```

Using the above code, you would be creating an `int` array with all the car data in it. Let's break this down.

`int`: This is the type of data you want a list of.

`cars`: This is the name of the array. Since you want to store many cars into it, you call it 'cars.'

`[5]`: This is how many `int`'s you want in your array.

`{ 4, 5, 7, 8, 3 }`: This is the data that you're storing. Notice that there's 1 number for each car.

To increase the number of cars, you increase the size of the array and simply add another value to the end of the list, which is a lot easier!

```
int cars[10] = { 4, 5, 7, 8, 3, 6, 3, 1, 7, 9 };
```

Now, what if we wanted to figure out the speed of one of the cars? Earlier, we would use `car_one` to access the number 4.

```
track_distance = car_four;
```

When using an array, we access using brackets like this:

```
track_distance = cars[4];
```

Does the above make sense to you? GOOD! Because there's one important thing that I need to tell you about arrays.

The **index** is the number that you use to access the data inside of the array at a given location. In the above code, 4 would be the index since you're seemingly trying to get the fourth car's speed.

However, when counting indexes, you don't start with 1, 2, 3, ... etc. Counting indexes actually starts at 0! What does this mean?? This means that the first car's speed is actually stored at `cars[0]`. The second's car's speed is stored at `cars[1]`. The third is at `cars[2]`, etc, etc.

**So, accessing the 4th car's speed with `cars[4]` is actually wrong. We need to use `cars[3]` !!**

```
track_distance = car_four; //This is lame
...
track_distance = cars[3]; //This is cool
```

This isn't hard to remember and I'll try to make sure I remind you! Besides, there are benefits to this!!

OKAY! Enough talking! Let's code an array!

Let's make an array called `world`. This will eventually hold our game's map! For now, we'll just have it hold random numbers. Let's make it have length of 20.

```
int world[20] = { 9, 5, 7, 2, 8, 3, 1, 7, 9, 3, 2, 1, 6, 4, 7, 8, 4, 3, 1, 4 };
```

We can also display some values from it in the `titlescreen()` function to test it! Add this!

```
arduboy.print(world[0]);
arduboy.print(world[1]);
arduboy.print(world[2]);
arduboy.print(world[3]);
```

Here's my full code:

```
//DinoSmasher

#include <Arduboy2.h>
Arduboy2 arduboy;

#define GAME_TITLE 0
#define GAME_PLAY 1
#define GAME_OVER 2
#define GAME_HIGH 3
int gamestate = GAME_TITLE;

int world[20] = { 9, 5, 7, 2, 8, 3, 1, 7, 9, 3, 2, 1, 6, 4, 7, 8, 4, 3, 1, 4 };

void titlescreen() {
  arduboy.setCursor(0, 0);
  arduboy.print("Title Screen\n");

  arduboy.print(world[0]);
  arduboy.print(world[1]);
  arduboy.print(world[2]);
  arduboy.print(world[3]);

  if(arduboy.justPressed(A_BUTTON)) {
    gamestate = GAME_PLAY;
  }
}
```

```
void gameplay() {
    arduboy.setCursor(0, 0);
    arduboy.print("Gameplay\n");
    if(arduboy.justPressed(A_BUTTON)) {
        gamestate = GAME_OVER;
    }
}

void gameoverscreen() {
    arduboy.setCursor(0, 0);
    arduboy.print("Game Over Screen\n");
    if(arduboy.justPressed(A_BUTTON)) {
        gamestate = GAME_HIGH;
    }
}

void highscorescreen() {
    arduboy.setCursor(0, 0);
    arduboy.print("High Score Screen\n");
    if(arduboy.justPressed(A_BUTTON)) {
        gamestate = GAME_TITLE;
    }
}

void gameloop() {
    switch(gamestate) {
        case GAME_TITLE:
            titlescreen();
            break;

        case GAME_PLAY:
            gameplay();
            break;

        case GAME_OVER:
            gameoverscreen();
            break;

        case GAME_HIGH:
            highscorescreen();
            break;
    }
}
```

```

void setup() {
  arduboy.begin();
  arduboy.setFrameRate(45);
  arduboy.display();
  arduboy.initRandomSeed();
  arduboy.clear();
}

void loop() {
  if(!(arduboy.nextFrame())) {
    return;
  }
  arduboy.pollButtons();
  arduboy.clear();
  gameloop();
  arduboy.display();
}

```

Isn't it annoying that you still must specify that you want to print each individual value? What if you wanted to print all the values of the world array?? Would you copy/paste the code 20 times? NO! We have for loops that can help us!

Remember, if we want to use a for loop to go through something 20 times, we'd do something like this:

```

for(int i = 0; i < 20; i++) {
  arduboy.print(i);
}

```

If we use the above code, the numbers 0~19 would be printed out to the screen. Instead, we want to print the 20 values inside of world, starting at index 0.

```

for(int i = 0; i < 20; i++) {
  arduboy.print( world[i] );
}

```

Put this code into your title screen function and run it! Full code, here:

```

//DinoSmasher

#include <Arduboy2.h>
Arduboy2 arduboy;

#define GAME_TITLE 0
#define GAME_PLAY 1
#define GAME_OVER 2
#define GAME_HIGH 3

```



```

int gamestate = GAME_TITLE;

int world[20] = { 9, 5, 7, 2, 8, 3, 1, 7, 9, 3, 2, 1, 6, 4, 7, 8, 4, 3, 1,
4 };

void titlescreen() {
    arduboy.setCursor(0, 0);
    arduboy.print("Title Screen\n");

    for(int i = 0; i < 20; i++) {
        arduboy.print(world[i]);
    }

    if(arduboy.justPressed(A_BUTTON)) {
        gamestate = GAME_PLAY;
    }
}

void gameplay() {
    arduboy.setCursor(0, 0);
    arduboy.print("Gameplay\n");
    if(arduboy.justPressed(A_BUTTON)) {
        gamestate = GAME_OVER;
    }
}

void gameoverscreen() {
    arduboy.setCursor(0, 0);
    arduboy.print("Game Over Screen\n");
    if(arduboy.justPressed(A_BUTTON)) {
        gamestate = GAME_HIGH;
    }
}

void highscorescreen() {
    arduboy.setCursor(0, 0);
    arduboy.print("High Score Screen\n");
    if(arduboy.justPressed(A_BUTTON)) {
        gamestate = GAME_TITLE;
    }
}

void gameloop() {
    switch(gamestate) {
        case GAME_TITLE:
            titlescreen();
            break;
    }
}

```

```

    case GAME_PLAY:
        gameplay();
        break;

    case GAME_OVER:
        gameoverscreen();
        break;

    case GAME_HIGH:
        highscorescreen();
        break;
}
}

void setup() {
    arduboy.begin();
    arduboy.setFrameRate(45);
    arduboy.display();
    arduboy.initRandomSeed();
    arduboy.clear();
}

void loop() {
    if(!(arduboy.nextFrame())) {
        return;
    }
    arduboy.pollButtons();
    arduboy.clear();
    gameloop();
    arduboy.display();
}

```

This is what you should get after compiling!



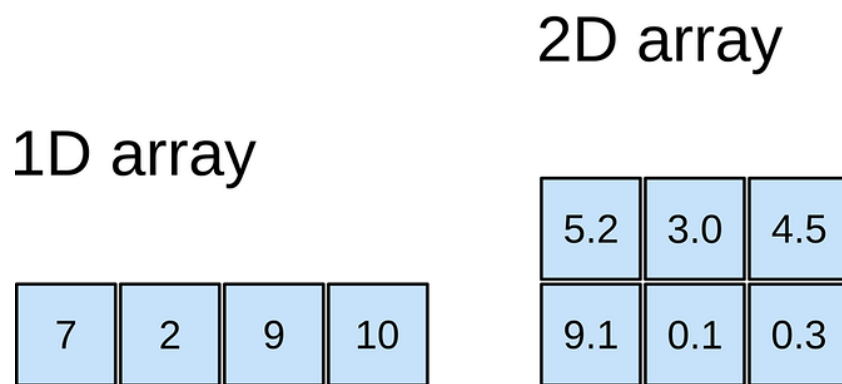
Alright, now that you have a good understanding of what arrays are, let's talk about 2D arrays!

## 6. Creating a 2D array

Arrays can store all sorts of data; You can have an array of `int`'s... an array of `boolean`'s... These are all called 1D arrays because they store data in 1 dimension. It's basically just a single list of variables all grouped together.

But, you can take arrays a step further! You can even have an array of arrays!! WHAT?! These are called 2D arrays because they store data in 2 dimensions... This means that instead of just a list of information, you can have a whole grid of it!

Here's a neat way to visualize the difference between a 1D array and a 2D array.



We can make a 2D array and store numbers into it and then print the numbers out like we did with a 1D array! Lemme show you how by starting by modifying our `world` array.

```
int world[4][20] = {  
    { 9, 5, 7, 2, 8, 3, 1, 7, 9, 3, 2, 1, 6, 4, 7, 8, 4, 3, 1, 4 },  
    { 8, 7, 2, 5, 6, 1, 4, 6, 6, 2, 6, 3, 5, 4, 2, 4, 3, 1, 5, 2 },  
    { 7, 2, 6, 3, 5, 4, 0, 2, 3, 5, 9, 2, 3, 5, 4, 2, 5, 4, 1, 9 },  
    { 6, 4, 5, 2, 6, 4, 5, 1, 2, 5, 4, 3, 2, 4, 6, 5, 4, 2, 4, 5 }  
};
```

Notice that our `world` array now has 2 lengths given to it. `[4]` refers to the the amount 1D arrays that will be stored. `[20]` is how long the 1D arrays will be. When creating this grid, you can think of the first number as the height of the grid and the second as the width.

In order to read one of the `int` values, we need to give two different indexes. Something like this: `world[2][6]` The first number is the row, and the second is the column. This would return the value `0`.

Now, how would you loop through all of these numbers to print them out? Well, we could write 4 separate `for` loops, but that would cause the same problem as before. Like in Part 6, we can have a `for` loop for the height and a `for` loop for the width!!

We can print the world by adjusting our for loop to look like this:

```
for(int y = 0; y < 4; y++) {  
    for(int x = 0; x < 20; x++) {  
        arduboy.print(world[y][x]);  
    }  
    arduboy.print("\n");  
}
```

Remember from Part 5 that `arduboy.print("\n")` is supposed to create a new line of text! Compile and test the completed code:

```
//DinoSmasher  
  
#include <Arduboy2.h>  
Arduboy2 arduboy;  
  
#define GAME_TITLE 0  
#define GAME_PLAY 1  
#define GAME_OVER 2  
#define GAME_HIGH 3  
int gamestate = GAME_TITLE;  
  
int world[4][20] = {  
    { 9, 5, 7, 2, 8, 3, 1, 7, 9, 3, 2, 1, 6, 4, 7, 8, 4, 3, 1, 4 },  
    { 8, 7, 2, 5, 6, 1, 4, 6, 6, 2, 6, 3, 5, 4, 2, 4, 3, 1, 5, 2 },  
    { 7, 2, 6, 3, 5, 4, 6, 2, 3, 5, 9, 2, 3, 5, 4, 2, 5, 4, 1, 9 },  
    { 6, 4, 5, 2, 6, 4, 5, 1, 2, 5, 4, 3, 2, 4, 6, 5, 4, 2, 4, 5 }  
};  
  
void titlescreen() {  
    arduboy.setCursor(0, 0);  
    arduboy.print("Title Screen\n");  
  
    for(int y = 0; y < 4; y++) {  
        for(int x = 0; x < 20; x++) {  
            arduboy.print(world[y][x]);  
        }  
        arduboy.print("\n");  
    }  
  
    if(arduboy.justPressed(A_BUTTON)) {  
        gamestate = GAME_PLAY;  
    }  
}
```

```
void gameplay() {
    arduboy.setCursor(0, 0);
    arduboy.print("Gameplay\n");
    if(arduboy.justPressed(A_BUTTON)) {
        gamestate = GAME_OVER;
    }
}

void gameoverscreen() {
    arduboy.setCursor(0, 0);
    arduboy.print("Game Over Screen\n");
    if(arduboy.justPressed(A_BUTTON)) {
        gamestate = GAME_HIGH;
    }
}

void highscorescreen() {
    arduboy.setCursor(0, 0);
    arduboy.print("High Score Screen\n");
    if(arduboy.justPressed(A_BUTTON)) {
        gamestate = GAME_TITLE;
    }
}

void gameloop() {
    switch(gamestate) {
        case GAME_TITLE:
            titlescreen();
            break;

        case GAME_PLAY:
            gameplay();
            break;

        case GAME_OVER:
            gameoverscreen();
            break;

        case GAME_HIGH:
            highscorescreen();
            break;
    }
}
```

```

void setup() {
  arduboy.begin();
  arduboy.setFrameRate(45);
  arduboy.display();
  arduboy.initRandomSeed();
  arduboy.clear();
}

void loop() {
  if(!(arduboy.nextFrame())) {
    return;
  }

  arduboy.pollButtons();
  arduboy.clear();
  gameloop();
  arduboy.display();
}

```

Cool, huh? Alright, let's try something very dangerous. Something that you may have even wondered on your own!! Our world array is current 4 tall and 20 wide, but what happens if we try to print out the 5th line of numbers? Or 6th line? Could we do it? What if those values don't exist? What will be printed out??? Replace the for loops with this and see for yourself!!

```

for(int y = 0; y < 6; y++) {
  for(int x = 0; x < 20; x++) {
    arduboy.print(world[y][x]);
  }
  arduboy.print("\n");
}

```

On my Arduboy, this is what it looks like!



Where did those exact numbers come from?? Shouldn't those be 0's?? Why does this happen? Well, whenever the Arduboy stores variables and things into memory, it puts them all into one big list. If you read outside of the array, you will be reading other variables. If you try to change the value that is outside of the array, you may accidentally change a random variable somewhere else in memory. This can cause all sorts of random stuff to happen that we don't want right now. We can experiment with this later, but for now, let's prevent this using our `#define` directive!

Let's define the height and width of `world` and then use those numbers in our definitions *and* for loops! OH, let's also change all the values to 0's or 1's just to check something.

```
#define WORLD_WIDTH    20
#define WORLD_HEIGHT   4
int world[WORLD_HEIGHT][WORLD_WIDTH] = {
  { 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
  { 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1 },
  { 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0 },
  { 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0 }
};

void titlescreen() {
  arduboy.setCursor(0, 0);
  arduboy.print("Title Screen\n");

  for(int y = 0; y < WORLD_HEIGHT; y++) {
    for(int x = 0; x < WORLD_WIDTH; x++) {
      arduboy.print(world[y][x]);
    }
    arduboy.print("\n");
  }

  if(arduboy.justPressed(A_BUTTON)) {
    gamestate = GAME_PLAY;
  }
}
```

This Let's move the for loops to their own function called `drawworld()` and reference it inside of the `gameplay()` function.

Here's the completed code:

```
//DinoSmasher

#include <Arduboy2.h>
Arduboy2 arduboy;

#define GAME_TITLE 0
#define GAME_PLAY 1
#define GAME_OVER 2
#define GAME_HIGH 3
int gamestate = GAME_TITLE;

#define WORLD_WIDTH 20
#define WORLD_HEIGHT 4
int world[WORLD_HEIGHT][WORLD_WIDTH] = {
  { 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
  { 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1 },
  { 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0 },
  { 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0 }
};

void drawworld() {
  for(int y = 0; y < WORLD_HEIGHT; y++) {
    for(int x = 0; x < WORLD_WIDTH; x++) {
      arduboy.print(world[y][x]);
    }
    arduboy.print("\n");
  }
}

void titlescreen() {
  arduboy.setCursor(0, 0);
  arduboy.print("Title Screen\n");
  if(arduboy.justPressed(A_BUTTON)) {
    gamestate = GAME_PLAY;
  }
}

void gameplay() {
  arduboy.setCursor(0, 0);
  arduboy.print("Gameplay\n");

  drawworld();
}
```



```
    if(arduboy.justPressed(A_BUTTON)) {
        gamestate = GAME_OVER;
    }
}

void gameoverscreen() {
    arduboy.setCursor(0, 0);
    arduboy.print("Game Over Screen\n");
    if(arduboy.justPressed(A_BUTTON)) {
        gamestate = GAME_HIGH;
    }
}

void highscorescreen() {
    arduboy.setCursor(0, 0);
    arduboy.print("High Score Screen\n");
    if(arduboy.justPressed(A_BUTTON)) {
        gamestate = GAME_TITLE;
    }
}

void gameloop() {
    switch(gamestate) {
        case GAME_TITLE:
            titlescreen();
            break;

        case GAME_PLAY:
            gameplay();
            break;

        case GAME_OVER:
            gameoverscreen();
            break;

        case GAME_HIGH:
            highscorescreen();
            break;
    }
}
```

```

void setup() {
  arduboy.begin();
  arduboy.setFrameRate(45);
  arduboy.display();
  arduboy.initRandomSeed();
  arduboy.clear();
}

void loop() {
  if(!(arduboy.nextFrame())) {
    return;
  }

  arduboy.pollButtons();
  arduboy.clear();
  gameloop();
  arduboy.display();
}

```

Let's test this code and see what happens! Don't the numbers in the gameplay screen kind of resemble a map? In fact, a lot of older games do something like draw the maps for their games.



But, we don't want that. We want to actually draw a map using the techniques we talked about in Part 6.

# Part 9: Mapping DinoSmasher

## This Tutorial



In this tutorial, we'll be adding graphics to our game, DinoSmasher!! Make sure you've worked on Part 8 of the tutorial series because this tutorial directly follows the previous one. We'll end up creating and moving a player around a map.

Here's what we've got to work on to finish this tutorial.

1. Converting & drawing some images
2. Formatting our world's images
3. Moving the map
4. Cropping the map
5. Cycling the map
6. Bounding the map

## Our Current Code

*//DinoSmasher*

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

#define GAME_TITLE 0
#define GAME_PLAY 1
#define GAME_OVER 2
#define GAME_HIGH 3
int gamestate = GAME_TITLE;

#define WORLD_WIDTH 20
#define WORLD_HEIGHT 4
int world[WORLD_HEIGHT][WORLD_WIDTH] = {
  { 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
  { 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1 },
  { 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0 },
  { 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0 }
};

void drawworld() {
  for(int y = 0; y < WORLD_HEIGHT; y++) {
    for(int x = 0; x < WORLD_WIDTH; x++) {
      arduboy.print(world[y][x]);
    }
    arduboy.print("\n");
  }
}

void titlescreen() {
  arduboy.setCursor(0, 0);
  arduboy.print("Title Screen\n");
  if(arduboy.justPressed(A_BUTTON)) {
    gamestate = GAME_PLAY;
  }
}

void gameplay() {
  arduboy.setCursor(0, 0);
  arduboy.print("Gameplay\n");
  drawworld();

  if(arduboy.justPressed(A_BUTTON)) {
    gamestate = GAME_OVER;
  }
}
```

```
void gameoverscreen() {
    arduboy.setCursor(0, 0);
    arduboy.print("Game Over Screen\n");
    if(arduboy.justPressed(A_BUTTON)) {
        gamestate = GAME_HIGH;
    }
}

void highscorescreen() {
    arduboy.setCursor(0, 0);
    arduboy.print("High Score Screen\n");
    if(arduboy.justPressed(A_BUTTON)) {
        gamestate = GAME_TITLE;
    }
}

void gameloop() {
    switch(gamestate) {
        case GAME_TITLE:
            titlescreen();
            break;

        case GAME_PLAY:
            gameplay();
            break;

        case GAME_OVER:
            gameoverscreen();
            break;

        case GAME_HIGH:
            highscorescreen();
            break;
    }
}

void setup() {
    arduboy.begin();
    arduboy.setFrameRate(45);
    arduboy.display();

    arduboy.initRandomSeed();

    arduboy.clear();
}
```

```

void loop() {
  if(!(arduboy.nextFrame())) {
    return;
  }

  arduboy.pollButtons();

  arduboy.clear();

  gameloop();

  arduboy.display();
}

```

## 1. Converting & drawing some images

In the previous tutorial, we made the `world` 2D array out of 0's and 1's. Instead of drawing those numbers, let's draw pictures, instead! Where we see a 0, let's draw some grass, and where we see a 1, let's draw some water!

Our maps will be made up of 16x16 images. Here are two sprites that I've made for us to use. They are 16x16 pixels.

GRASS: 

WATER: 

Converting them using my [ToChars](#) <sup>37</sup> website and putting them into their own byte arrays, we get:

```

const unsigned char grass[] PROGMEM = {
  0xff, 0x7f, 0xfb, 0xff, 0xff, 0xbf, 0xff, 0xff, 0xf7, 0xff, 0xfd, 0xff,
  0xff, 0xf7, 0x7f, 0xff, 0xdf, 0xff, 0xff, 0xfb, 0x7f, 0xff, 0xff, 0xff,
  0xef, 0xfe, 0xff, 0xff, 0xfb, 0xff, 0x7f, 0xff
};

```

```

const unsigned char water[] PROGMEM = {
  0x08, 0x10, 0x10, 0x08, 0x10, 0x08, 0x10, 0x10, 0x10, 0x08, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x20, 0x40, 0x40, 0x20, 0x00, 0x01, 0x02, 0x02,
  0x01, 0x02, 0x02, 0x01, 0x02, 0x21, 0x40, 0x40
};

```

Since we'll be dealing with 16x16, let's define a keyword to keep track of that called `TILE_SIZE` and set it to 16.

```
#define TILE_SIZE 16
```

Let's update our `drawworld()` function's for loops. Instead of printing each number, we need to use an `if` statement to determine which picture we're going to draw.

```
#define TILE_SIZE 16
void drawworld() {
    for(int y = 0; y < WORLD_HEIGHT; y++) {
        for(int x = 0; x < WORLD_WIDTH; x++) {
            if(world[y][x] == 0) {
                arduboy.drawBitmap(x, y, grass, TILE_SIZE, TILE_SIZE, WHITE);
            }
            if(world[y][x] == 1) {
                arduboy.drawBitmap(x, y, water, TILE_SIZE, TILE_SIZE, WHITE);
            }
        }
    }
}
```

If you run this code, you'll see this.



Our problem is that we need to tile the images 16 pixels apart and the way we are drawing the images, we are only separating them 1 pixel since the for loops increase the x and y variables by 1 each loop.

Instead, let's multiply the X and Y values by 16, or the `TILE_SIZE` when drawing the images.

```
#define TILE_SIZE 16
void drawworld() {
    for(int y = 0; y < WORLD_HEIGHT; y++) {
        for(int x = 0; x < WORLD_WIDTH; x++) {
            if(world[y][x] == 0) {
                arduboy.drawBitmap(x * TILE_SIZE, y * TILE_SIZE, grass, TILE_SIZE,
TILE_SIZE, WHITE);
            }
            if(world[y][x] == 1) {
                arduboy.drawBitmap(x * TILE_SIZE, y * TILE_SIZE, water, TILE_SIZE,
TILE_SIZE, WHITE);
            }
        }
    }
}
```

There we go!





## 2. Formatting our world's images

In our game's world, we want to have a lot of different kinds of images. It would be pretty annoying to copy/paste a bunch of `if` statements in our `drawworld()` function. We could save some time by using a `switch` statement, but there's a better way to condense this code now that we know more about arrays.

Remember when I said that we can have all sorts of stuff inside of arrays, including other arrays? Of course you do! What we can do is actually make an array of `char` arrays to hold all of our images.

Let's make a 2D `char` array called `tiles` to hold our water and grass data.

```
const unsigned char tiles[2][32] PROGMEM = {
    { 0xff, 0x7f, 0xfb, 0xff, 0xff, 0xbf, 0xff, 0xff, 0xf7, 0xff, 0xfd,
      0xff, 0xff, 0xf7, 0x7f, 0xff, 0xdf, 0xff, 0xff, 0xfb, 0x7f, 0xff, 0xff,
      0xff, 0xef, 0xfe, 0xff, 0xff, 0xfb, 0xff, 0x7f, 0xff },
    { 0x08, 0x10, 0x10, 0x08, 0x10, 0x08, 0x10, 0x10, 0x10, 0x10, 0x08, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x20, 0x40, 0x40, 0x20, 0x00, 0x01, 0x02,
      0x02, 0x01, 0x02, 0x02, 0x01, 0x02, 0x21, 0x40, 0x40 }
};
```

I put grass's data first, which would have index 0. Then, water's data is second, at index 1.

To draw the grass and water data with `tiles`, we'd do something like this:

```
arduboy.drawBitmap(a, b, tiles[0], TILE_SIZE, TILE_SIZE, WHITE);
arduboy.drawBitmap(c, e, tiles[1], TILE_SIZE, TILE_SIZE, WHITE);
```

Notice that the only thing different is the index 0 or 1. This means that we won't have to use any `if` statements in our `drawworld()` function! All we need to do is get the value at `world[y][x]`, which will be a 0 or 1, and put that into `tiles[]` to get the image data. This is what the updated `drawworld()` function would look like:

```
void drawworld() {
    for(int y = 0; y < WORLD_HEIGHT; y++) {
        for(int x = 0; x < WORLD_WIDTH; x++) {
            arduboy.drawBitmap(x * TILE_SIZE, y * TILE_SIZE, tiles[ world[y][x]
], TILE_SIZE, TILE_SIZE, WHITE);

        }
    }
}
```

Compile the code and see how it the result is the same as before!

```
//DinoSmasher

#include <Arduboy2.h>
Arduboy2 arduboy;

#define GAME_TITLE 0
#define GAME_PLAY 1
#define GAME_OVER 2
#define GAME_HIGH 3
int gamestate = GAME_TITLE;

const unsigned char tiles[2][32] PROGMEM = {
    { 0xff, 0x7f, 0xfb, 0xff, 0xff, 0xbf, 0xff, 0xff, 0xf7, 0xff, 0xfd,
      0xff, 0xff, 0xf7, 0x7f, 0xff, 0xdf, 0xff, 0xff, 0xfb, 0x7f, 0xff, 0xff,
      0xff, 0xef, 0xfe, 0xff, 0xff, 0xfb, 0xff, 0x7f, 0xff },
    { 0x08, 0x10, 0x10, 0x08, 0x10, 0x08, 0x10, 0x10, 0x10, 0x08, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x20, 0x40, 0x40, 0x20, 0x00, 0x01, 0x02,
      0x02, 0x01, 0x02, 0x02, 0x01, 0x02, 0x21, 0x40, 0x40 }
};

#define WORLD_WIDTH 20
#define WORLD_HEIGHT 4
int world[WORLD_HEIGHT][WORLD_WIDTH] = {
    { 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0 },
    { 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1 },
    { 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0 },
    { 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0 }
};

#define TILE_SIZE 16
void drawworld() {
    for(int y = 0; y < WORLD_HEIGHT; y++) {
        for(int x = 0; x < WORLD_WIDTH; x++) {
            arduboy.drawBitmap(x * TILE_SIZE, y * TILE_SIZE, tiles[world[y][x]],
TILE_SIZE, TILE_SIZE, WHITE);
        }
    }
}

void titlescreen() {
    arduboy.setCursor(0, 0);
    arduboy.print("Title Screen\n");
    if(arduboy.justPressed(A_BUTTON)) {
        gamestate = GAME_PLAY;
    }
}
```

```

void gameplay() {
    arduboy.setCursor(0, 0);
    arduboy.print("Gameplay\n");

    drawworld();

    if(arduboy.justPressed(A_BUTTON)) {
        gamestate = GAME_OVER;
    }
}

void gameoverscreen() {
    arduboy.setCursor(0, 0);
    arduboy.print("Game Over Screen\n");
    if(arduboy.justPressed(A_BUTTON)) {
        gamestate = GAME_HIGH;
    }
}

void highscorescreen() {
    arduboy.setCursor(0, 0);
    arduboy.print("High Score Screen\n");
    if(arduboy.justPressed(A_BUTTON)) {
        gamestate = GAME_TITLE;
    }
}

void gameloop() {
    switch(gamestate) {
        case GAME_TITLE:
            titlescreen();
            break;

        case GAME_PLAY:
            gameplay();
            break;

        case GAME_OVER:
            gameoverscreen();
            break;

        case GAME_HIGH:
            highscorescreen();
            break;
    }
}

```

```

void setup() {
  arduboy.begin();
  arduboy.setFrameRate(45);
  arduboy.display();

  arduboy.initRandomSeed();



  arduboy.clear();
}

void loop() {
  if(!(arduboy.nextFrame())) {
    return;
  }

  arduboy.pollButtons();
  arduboy.clear();
  gameloop();
  arduboy.display();
}

```

Even though the results are the same, we are doing this for a very important reason! If we want to add in more images, we can do that very easily! Let's add two more in!

TREES:   
 STONE: 

I converted the images with ToChars and then added them to the `tiles` array. The tree image should be the third and stone image should be the fourth image. Don't forget to update the size of `tiles`.

```

const unsigned char tiles[4][32] PROGMEM = {
  { 0xff, 0x7f, 0xfb, 0xff, 0xff, 0xbf, 0xff, 0xff, 0xf7, 0xff, 0xfd,
    0xff, 0xff, 0xf7, 0x7f, 0xff, 0xdf, 0xff, 0xff, 0xfb, 0x7f, 0xff, 0xff,
    0xff, 0xef, 0xfe, 0xff, 0xff, 0xfb, 0xff, 0x7f, 0xff },
  { 0x08, 0x10, 0x10, 0x08, 0x10, 0x08, 0x10, 0x10, 0x10, 0x08, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x20, 0x40, 0x40, 0x20, 0x00, 0x01, 0x02,
    0x02, 0x01, 0x02, 0x02, 0x01, 0x02, 0x21, 0x40, 0x40 },
  { 0xff, 0x1f, 0x5b, 0x3f, 0xeb, 0xdd, 0xff, 0xf7, 0xbb, 0xef, 0xfd,
    0x7f, 0xe3, 0xcb, 0xe3, 0xff, 0xff, 0xc7, 0x96, 0xc7, 0xff, 0xff, 0xef,
    0xfd, 0xff, 0xe3, 0xcb, 0xe3, 0xff, 0xff, 0x7b, 0xff },
  { 0xff, 0xdf, 0x7b, 0x3f, 0x9f, 0x6f, 0x77, 0xab, 0xdb, 0xd7, 0xcd,
    0x5f, 0xbf, 0x77, 0xff, 0xff, 0xff, 0xc1, 0xdc, 0xd3, 0xaf, 0x9f, 0xae,
    0xb0, 0xbb, 0xbd, 0xbd, 0xba, 0xd7, 0xcc, 0x63, 0xff }
};

```

Now that the images are added, let's update the `world` array to include the numbers 2 and 3. These will cause stones and trees to be drawn.

```
int world[WORLD_HEIGHT][WORLD_WIDTH] = {
  { 2, 0, 0, 1, 0, 0, 0, 2, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0 },
  { 0, 1, 1, 1, 0, 1, 0, 0, 1, 2, 0, 0, 0, 3, 3, 0, 2, 1, 1, 1 },
  { 0, 0, 0, 0, 0, 1, 3, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 2, 0 },
  { 3, 0, 0, 3, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0 }
};
```

This is great! Except there's two more things that I want to do before we get on to cooler things.

First of all, let's adjust the `WORLD_WIDTH`. As it, we can't see the entire map on the Arduboy's screen, so we don't really need one that big. Let's change it to 8 and remove some values.

```
#define WORLD_WIDTH    8
...
int world[WORLD_HEIGHT][WORLD_WIDTH] = {
  { 2, 0, 0, 1, 0, 0, 0, 2 },
  { 0, 1, 1, 1, 0, 1, 0, 0 },
  { 0, 0, 0, 0, 0, 1, 3, 0 },
  { 3, 0, 0, 3, 2, 1, 1, 1 }
};
```

Now that the `world` array is small enough, take a look at your Arduboy screen, you could (in theory) create any kind of combination of map using the trees, grass, stone, and water tiles that we have. But, it may be hard visualizing the result because the numbers could get confusing, especially if we end up with over 100 different numbers!

Let's use `#define` to replace these numbers with words that makes sense to us. Use the following code:

```
#define GRASS      0
#define WATER     1
#define TREES     2
#define STONE     3
```

Since we're defining GRASS, WATER, TREES, and STONE, we can replace the way we store information in the world array to match.

```
int world[WORLD_HEIGHT][WORLD_WIDTH] = {
  { TREES, GRASS, GRASS, WATER, GRASS, GRASS, GRASS, TREES },
  { GRASS, WATER, WATER, WATER, GRASS, WATER, GRASS, GRASS },
  { GRASS, GRASS, GRASS, GRASS, GRASS, WATER, STONE, GRASS },
  { STONE, GRASS, GRASS, STONE, TREES, WATER, WATER, WATER }
};
```

When making a game with 2D arrays for maps and stuff, I like to do this. I did this in [Circuit Dude](#) and [Midnight Wild](#). This means you could download and modify the maps for those games by changing their 2D arrays.

Another thing about doing this is that you could have a different 2D array for each level.

Anyway, there is another really great reason to do this, but we'll have to wait for that! For now, try this code on your device!

```
//DinoSmasher

#include <ArduBoy2.h>
ArduBoy2 arduBoy2;

#define GAME_TITLE 0
#define GAME_PLAY 1
#define GAME_OVER 2
#define GAME_HIGH 3
int gameState = GAME_TITLE;

const unsigned char tiles[4][32] PROGMEM = {
  { 0xff, 0x7f, 0xfb, 0xff, 0xff, 0xbf, 0xff, 0xff, 0xf7, 0xff, 0xfd,
    0xff, 0xff, 0xf7, 0x7f, 0xff, 0xdf, 0xff, 0xff, 0xfb, 0x7f, 0xff, 0xff,
    0xff, 0xef, 0xfe, 0xff, 0xff, 0xfb, 0xff, 0x7f, 0xff },
  { 0x08, 0x10, 0x10, 0x08, 0x10, 0x08, 0x10, 0x10, 0x10, 0x08, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x20, 0x40, 0x40, 0x20, 0x00, 0x01, 0x02,
    0x02, 0x01, 0x02, 0x02, 0x01, 0x02, 0x21, 0x40, 0x40 },
  { 0xff, 0x1f, 0x5b, 0x3f, 0xeb, 0xdd, 0xff, 0xf7, 0xbb, 0xef, 0xfd,
    0x7f, 0xe3, 0xcb, 0xe3, 0xff, 0xff, 0xc7, 0x96, 0xc7, 0xff, 0xff, 0xef,
    0xfd, 0xff, 0xe3, 0xcb, 0xe3, 0xff, 0xff, 0x7b, 0xff },
  { 0xff, 0xdf, 0x7b, 0x3f, 0x9f, 0x6f, 0x77, 0xab, 0xdb, 0xd7, 0xcd,
    0x5f, 0xbf, 0x77, 0xff, 0xff, 0xff, 0xc1, 0xdc, 0xd3, 0xaf, 0x9f, 0xae,
    0xb0, 0xbb, 0xbd, 0xbd, 0xba, 0xd7, 0xcc, 0x63, 0xff }
};

#define WORLD_WIDTH 8
#define WORLD_HEIGHT 4
#define GRASS 0
```

```

#define WATER      1
#define TREES      2
#define STONE      3
int world[WORLD_HEIGHT][WORLD_WIDTH] = {
    { TREES, GRASS, GRASS, WATER, GRASS, GRASS, GRASS, TREES },
    { GRASS, WATER, WATER, WATER, GRASS, WATER, GRASS, GRASS },
    { GRASS, GRASS, GRASS, GRASS, GRASS, WATER, STONE, GRASS },
    { STONE, GRASS, GRASS, STONE, TREES, WATER, WATER, WATER }
};

#define TILE_SIZE  16
void drawworld() {
    for(int y = 0; y < WORLD_HEIGHT; y++) {
        for(int x = 0; x < WORLD_WIDTH; x++) {
            arduboy.drawBitmap(x * TILE_SIZE, y * TILE_SIZE, tiles[world[y][x]],
TILE_SIZE, TILE_SIZE, WHITE);

        }
    }
}

void titlescreen() {
    arduboy.setCursor(0, 0);
    arduboy.print("Title Screen\n");
    if(arduboy.justPressed(A_BUTTON)) {
        gamestate = GAME_PLAY;
    }
}

void gameplay() {
    arduboy.setCursor(0, 0);
    arduboy.print("Gameplay\n");

    drawworld();

    if(arduboy.justPressed(A_BUTTON)) {
        gamestate = GAME_OVER;
    }
}

void gameoverscreen() {
    arduboy.setCursor(0, 0);
    arduboy.print("Game Over Screen\n");
    if(arduboy.justPressed(A_BUTTON)) {
        gamestate = GAME_HIGH;
    }
}

```

```
void highscorescreen() {  
    arduboy.setCursor(0, 0);  
    arduboy.print("High Score Screen\n");  
    if(arduboy.justPressed(A_BUTTON)) {  
        gamestate = GAME_TITLE;  
    }  
}
```

```
void gameloop() {  
    switch(gamestate) {  
        case GAME_TITLE:  
            titlescreen();  
            break;  
  
        case GAME_PLAY:  
            gameplay();  
            break;  
  
        case GAME_OVER:  
            gameoverscreen();  
            break;  
  
        case GAME_HIGH:  
            highscorescreen();  
            break;  
    }  
}
```

```
void setup() {  
    arduboy.begin();  
    arduboy.setFrameRate(45);  
    arduboy.display();  
    arduboy.initRandomSeed();  
    arduboy.clear();  
}
```

```
void loop() {  
    if(!(arduboy.nextFrame())) {  
        return;  
    }  
  
    arduboy.pollButtons();  
    arduboy.clear();  
    gameloop();  
    arduboy.display();  
}
```



### 3. Moving the map

Now, in Part 6, we were able to draw some grass and allowed the player to move a character around by changing their character's X and Y position values. However, not all games work like that. Some games, the player is stuck in the center of the screen and the game's map moves in the background. Games like Zelda and Pokemon do this.

To learn how to do it, there's a few things we need to understand, but first, let's just simply move the X and Y position of the background based on the player's position to offset the map on the screen. Let's make a variable for both the X and Y called `mapx` and `mapy`.

```
int mapx = 0;
int mapy = 0;
```

In the game, we are going to allow the player to move around and change the values of `mapx` and `mapy`. After they are updated, we want to draw the world. This means that we need to get the player's input before drawing the world.

We should make a function called `playerinput()` to handle these changes. In the `gameplay()` function, before `drawworld()`, let's reference the `playerinput()` function.

```
void playerinput() {
}

void gameplay() {
    arduboy.setCursor(0, 0);
    arduboy.print("Gameplay\n");

    playerinput();
    drawworld();

    if(arduboy.justPressed(A_BUTTON)) {
        gamestate = GAME_OVER;
    }
}
```

Changing the values of `mapx` and `mapy` will be easy. Use this code, which should be familiar to you:

```
void playerinput() {
    if(arduboy.pressed(UP_BUTTON)) {
        mapy += 1;
    }
}
```

```

    if(arduboy.pressed(DOWN_BUTTON)) {
        mapy -= 1;
    }
    if(arduboy.pressed(LEFT_BUTTON)) {
        mapx += 1;
    }
    if(arduboy.pressed(RIGHT_BUTTON)) {
        mapx -= 1;
    }
}

```

When we draw the game world, let's offset it by `mapx` and `mapy`.

```

void drawworld() {
    for(int y = 0; y < WORLD_HEIGHT; y++) {
        for(int x = 0; x < WORLD_WIDTH; x++) {
            arduboy.drawBitmap(x * TILE_SIZE + mapx, y * TILE_SIZE + mapy,
tiles[world[y][x]], TILE_SIZE, TILE_SIZE, WHITE);

        }
    }
}

```

After compiling your code, you should be able to use the direction buttons to move the map around the screen. Try the following code. (I've adjusted the world map's size.)

```

//DinoSmasher

#include <Arduboy2.h>
Arduboy2 arduboy;

#define GAME_TITLE 0
#define GAME_PLAY 1
#define GAME_OVER 2
#define GAME_HIGH 3
int gamestate = GAME_TITLE;

int mapx = 0;
int mapy = 0;

const unsigned char tiles[4][32] PROGMEM = {
    { 0xff, 0x7f, 0xfb, 0xff, 0xff, 0xbf, 0xff, 0xff, 0xf7, 0xff, 0xfd,
0xff, 0xff, 0xf7, 0x7f, 0xff, 0xdf, 0xff, 0xff, 0xfb, 0x7f, 0xff, 0xff,
0xff, 0xef, 0xfe, 0xff, 0xff, 0xfb, 0xff, 0x7f, 0xff },
    { 0x08, 0x10, 0x10, 0x08, 0x10, 0x08, 0x10, 0x10, 0x10, 0x08, 0x00,
0x00, 0x00, 0x00, 0x00, 0x20, 0x40, 0x40, 0x20, 0x00, 0x01, 0x02,
0x02, 0x01, 0x02, 0x02, 0x01, 0x02, 0x21, 0x40, 0x40 },

```

```

    { 0xff, 0x1f, 0x5b, 0x3f, 0xeb, 0xdd, 0xff, 0xf7, 0xbb, 0xef, 0xfd,
    0x7f, 0xe3, 0xcb, 0xe3, 0xff, 0xff, 0xc7, 0x96, 0xc7, 0xff, 0xff, 0xef,
    0xfd, 0xff, 0xe3, 0xcb, 0xe3, 0xff, 0xff, 0x7b, 0xff },
    { 0xff, 0xdf, 0x7b, 0x3f, 0x9f, 0x6f, 0x77, 0xab, 0xdb, 0xd7, 0xcd,
    0x5f, 0xbf, 0x77, 0xff, 0xff, 0xff, 0xc1, 0xdc, 0xd3, 0xaf, 0x9f, 0xae,
    0xb0, 0xbb, 0xbd, 0xbd, 0xba, 0xd7, 0xcc, 0x63, 0xff }
};

#define WORLD_WIDTH    14
#define WORLD_HEIGHT   7
#define GRASS          0
#define WATER          1
#define TREES          2
#define STONE          3
int world[WORLD_HEIGHT][WORLD_WIDTH] = {
    { TREES, GRASS, GRASS, WATER, GRASS, GRASS, GRASS, TREES, GRASS, GRASS,
    GRASS, GRASS, GRASS, TREES },
    { GRASS, WATER, WATER, WATER, GRASS, WATER, GRASS, GRASS, GRASS, GRASS,
    GRASS, STONE, GRASS, GRASS },
    { GRASS, GRASS, GRASS, GRASS, GRASS, WATER, STONE, GRASS, GRASS, GRASS,
    TREES, GRASS, GRASS, GRASS },
    { STONE, GRASS, GRASS, STONE, TREES, WATER, WATER, WATER, GRASS, WATER,
    WATER, GRASS, TREES, GRASS },
    { GRASS, GRASS, GRASS, GRASS, TREES, GRASS, GRASS, GRASS, TREES, WATER,
    GRASS, GRASS, STONE, TREES },
    { GRASS, GRASS, GRASS, WATER, STONE, GRASS, GRASS, TREES, TREES, TREES,
    GRASS, GRASS, WATER, WATER },
    { GRASS, WATER, WATER, TREES, GRASS, WATER, WATER, TREES, TREES, GRASS,
    GRASS, GRASS, GRASS, STONE }
};

#define TILE_SIZE 16
void drawworld() {
    for(int y = 0; y < WORLD_HEIGHT; y++) {
        for(int x = 0; x < WORLD_WIDTH; x++) {
            arduboy.drawBitmap(x * TILE_SIZE + mapx, y * TILE_SIZE + mapy,
tiles[world[y][x]], TILE_SIZE, TILE_SIZE, WHITE);
        }
    }
}

void titlescreen() {
    arduboy.setCursor(0, 0);
    arduboy.print("Title Screen\n");
    if(arduboy.justPressed(A_BUTTON)) {
        gamestate = GAME_PLAY;
    }
}

```

```

void playerinput() {
    if(arduboy.pressed(UP_BUTTON)) {
        mapy += 1;
    }
    if(arduboy.pressed(DOWN_BUTTON)) {
        mapy -= 1;
    }
    if(arduboy.pressed(LEFT_BUTTON)) {
        mapx += 1;
    }
    if(arduboy.pressed(RIGHT_BUTTON)) {
        mapx -= 1;
    }
}

void gameplay() {
    arduboy.setCursor(0, 0);
    arduboy.print("Gameplay\n");
    playerinput();
    drawworld();

    if(arduboy.justPressed(A_BUTTON)) {
        gamestate = GAME_OVER;
    }
}

void gameoverscreen() {
    arduboy.setCursor(0, 0);
    arduboy.print("Game Over Screen\n");
    if(arduboy.justPressed(A_BUTTON)) {
        gamestate = GAME_HIGH;
    }
}

void highscorescreen() {
    arduboy.setCursor(0, 0);
    arduboy.print("High Score Screen\n");
    if(arduboy.justPressed(A_BUTTON)) {
        gamestate = GAME_TITLE;
    }
}

void gameloop() {
    switch(gamestate) {
        case GAME_TITLE:
            titlescreen();
            break;
    }
}

```

```

    case GAME_PLAY:
        gameplay();
        break;
    case GAME_OVER:
        gameoverscreen();
        break;
    case GAME_HIGH:
        highscorescreen();
        break;
}
}

void setup() {
    arduboy.begin();
    arduboy.setFrameRate(45);
    arduboy.display();
    arduboy.initRandomSeed();
    arduboy.clear();
}

void loop() {
    if(!(arduboy.nextFrame())) {
        return;
    }
    arduboy.pollButtons();
    arduboy.clear();
    gameloop();
    arduboy.display();
}

```

Right now, the map is being drawn that is bigger than the Arduboy screen. The bigger the map is, the more images the Arduboy will try to draw. This will slow down the Arduboy at some point, so we need to crop the map and only draw a portion that is slightly bigger than the Arduboy's screen.

We also want to stop the player from being able to move the map off of the screen. There needs to be some kinds of boundaries!

## 4. Cropping the map

Attempting to loop through and draw everything in our `world` array will slow our game down so much that it'll be unplayable if `world` gets too big.

We really only need to draw 8 tiles wide and 4 tiles high to fill the entire screen. The way we get these numbers is by finding the how many tiles can span across the the width of the screen and height of the screen. The tiles are 16 pixels wide and 16 pixels tall. The screen is 128 pixels wide and 64 pixels tall.

**Screen width of 128 pixels wide ÷ tile width of 16 pixels = 8**  
**Screen height of 64 pixels wide ÷ tile height of 16 pixels = 4**

Since these values won't change, we could store these into constant `int`'s!

```
const int tileswide = 128 / TILE_SIZE;  
const int tilestall = 64 / TILE_SIZE;
```

The Arduboy2 library has defined `WIDTH` and `HEIGHT` for us to use in a situation like this to represent the screen's size so we don't have to always remember the exact values. Let's use those:

```
const int tileswide = WIDTH / TILE_SIZE;  
const int tilestall = HEIGHT / TILE_SIZE;
```

Alright, let's adjust our draw function's for loops. Instead of looping through the entire `WORLD_HEIGHT` and `WORLD_WIDTH`, we only need to go to 8 tall and 4 wide, which are the values of `tileswide` and `tilestall`. Let's swap those out:

```
void drawworld() {  
    const int tileswide = WIDTH / TILE_SIZE;  
    const int tilestall = HEIGHT / TILE_SIZE;  
  
    for(int y = 0; y < tilestall; y++) {  
        for(int x = 0; x < tileswide; x++) {  
            arduboy.drawBitmap(x * TILE_SIZE + mapx, y * TILE_SIZE + mapy,  
tiles[world[y][x]], TILE_SIZE, TILE_SIZE, WHITE);  
        }  
    }  
}
```

Compiling and running this code would cause you to realize that you can now move a smaller section of the map around the screen.

When we draw the map tiles, we are offsetting them with the `mapx` and `mapy` variables. If these get too big, the map will move off of the screen and we won't see anything!!

Instead, we need to keep the map on the screen and prevent the `mapx` and `mapy` variables from getting too big.

What we could do is use some code like this:

```
if(mapx > 10) {  
    mapx = 0;  
}  
if(mapx < 0) {  
    mapx = 10;  
}
```

This would ensure that `mapx` is between 0 and 10. This isn't optimal, though. Remember, the **modulo** operator could be used to do this!

```
mapx = mapx % 10;
```

In this case, we'd be letting the map move up to 10 pixels before the screen moved back. We would also be overwriting our `mapx` variable. This isn't something we want to save- We only want to use this value when drawing our map tiles. Let's replace `mapx` and `mapy` with `mapx % 10` and `mapy % 10` in our `arduboy.drawBitmap()` function. This way, we can continue to keep track of the map's X/Y position, no matter how big and still confine the map to be drawn on the screen.

```
void drawworld() {
    const int tileswide = WIDTH / TILE_SIZE;
    const int tilestall = HEIGHT / TILE_SIZE;

    for(int y = 0; y < tilestall; y++) {
        for(int x = 0; x < tileswide; x++) {
            arduboy.drawBitmap(x * TILE_SIZE + mapx % 10, y * TILE_SIZE + mapy % 10, tiles[world[y][x]], TILE_SIZE, TILE_SIZE, WHITE);
        }
    }
}
```

Compiling and running your code will result in the ability to move the map but not too far from the screen! It even works going backwards! But do you notice that there is a lot of black space when moving the map around? To fix this, we should increase how many images we draw across and tall by 1 to fill in that space.

```
const int tileswide = WIDTH / TILE_SIZE + 1;
const int tilestall = HEIGHT / TILE_SIZE + 1;
```

Compile your code and test it out!

Alright, there's another problem... When moving around, you can't even see the entirety of the bottom and right-most tiles. This is because we are only allowing the player to see 10 pixels of those tiles at a time. To stop it from cutting, let's change the 10 to 16. Actually, no.

Let's change the 10 to TILE\_SIZE so that we can see the entire tile.

```
void drawworld() {
    const int tileswide = WIDTH / TILE_SIZE + 1;
    const int tilestall = HEIGHT / TILE_SIZE + 1;

    for(int y = 0; y < tilestall; y++) {
        for(int x = 0; x < tileswide; x++) {
            arduboy.drawBitmap(x * TILE_SIZE + mapx % TILE_SIZE, y * TILE_SIZE +
mapy % TILE_SIZE, tiles[world[y][x]], TILE_SIZE, TILE_SIZE, WHITE);

        }
    }
}
```

Now that we confine the map to the screen the BIGGEST problem that I've ignored is that the map's tiles don't actually change, so you can't actually pan around the world. You'll only be looking at the same section of the map!

Let's do that, now!

## 5. Cycling the map

To understand how cycling through the map works, let's verify a few things, including our X/Y coordinates. Let's print those out.

In the `gameplay()` function, remove the `arduboy.setCursor()` and `arduboy.print()` functions since we don't need that, anymore.

In the `drawworld()` functions, let's draw a black rectangle and then print our `mapx` and `mapy` coordinates *after* drawing the tiles in the `for` loops.

OH, we should also add a black 16x16 square to the *middle* of the screen to represent the player.

```
arduboy.fillRect(WIDTH / 2 - 8, HEIGHT / 2 - 8, 16, 16, BLACK);
arduboy.fillRect(0, 0, 48, 8, BLACK);
arduboy.setCursor(0, 0);
arduboy.print(mapx);
arduboy.print(",");
arduboy.print(mapy);
```

Let's run these quick changes and observe what happens when we move the screen around.



Here's the full code so far:

```
//DinoSmasher

#include <Arduboy2.h>
Arduboy2 arduboy;

#define GAME_TITLE 0
#define GAME_PLAY 1
#define GAME_OVER 2
#define GAME_HIGH 3
int gamestate = GAME_TITLE;

int mapx = 0;
int mapy = 0;

const unsigned char tiles[4][32] PROGMEM = {
    { 0xff, 0x7f, 0xfb, 0xff, 0xff, 0xbf, 0xff, 0xff, 0xf7, 0xff, 0xfd,
      0xff, 0xff, 0xf7, 0x7f, 0xff, 0xdf, 0xff, 0xff, 0xfb, 0x7f, 0xff, 0xff,
      0xff, 0xef, 0xfe, 0xff, 0xff, 0xfb, 0xff, 0x7f, 0xff },
    { 0x08, 0x10, 0x10, 0x08, 0x10, 0x08, 0x10, 0x10, 0x10, 0x08, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x20, 0x40, 0x40, 0x20, 0x00, 0x01, 0x02,
      0x02, 0x01, 0x02, 0x02, 0x01, 0x02, 0x21, 0x40, 0x40 },
    { 0xff, 0x1f, 0x5b, 0x3f, 0xeb, 0xdd, 0xff, 0xf7, 0xbb, 0xef, 0xfd,
      0x7f, 0xe3, 0xcb, 0xe3, 0xff, 0xff, 0xc7, 0x96, 0xc7, 0xff, 0xff, 0xef,
      0xfd, 0xff, 0xe3, 0xcb, 0xe3, 0xff, 0xff, 0x7b, 0xff },
    { 0xff, 0xdf, 0x7b, 0x3f, 0x9f, 0x6f, 0x77, 0xab, 0xdb, 0xd7, 0xcd,
      0x5f, 0xbf, 0x77, 0xff, 0xff, 0xff, 0xc1, 0xdc, 0xd3, 0xaf, 0x9f, 0xae,
      0xb0, 0xbb, 0xbd, 0xbd, 0xba, 0xd7, 0xcc, 0x63, 0xff }
};

#define WORLD_WIDTH 14
#define WORLD_HEIGHT 7
#define GRASS 0
#define WATER 1
#define TREES 2
#define STONE 3
int world[WORLD_HEIGHT][WORLD_WIDTH] = {
    { TREES, GRASS, GRASS, WATER, GRASS, GRASS, GRASS, GRASS, TREES, GRASS, GRASS,
      GRASS, GRASS, GRASS, TREES },
    { GRASS, WATER, WATER, WATER, GRASS, WATER, GRASS, GRASS, GRASS, GRASS,
      GRASS, STONE, GRASS, GRASS },
    { GRASS, GRASS, GRASS, GRASS, GRASS, WATER, STONE, GRASS, GRASS, GRASS,
      TREES, GRASS, GRASS, GRASS },
    { STONE, GRASS, GRASS, STONE, TREES, WATER, WATER, WATER, GRASS, WATER,
      WATER, GRASS, TREES, GRASS },
    { GRASS, GRASS, GRASS, GRASS, TREES, GRASS, GRASS, GRASS, TREES, WATER,
      GRASS, GRASS, STONE, TREES },
};
```

```

    { GRASS, GRASS, GRASS, WATER, STONE, GRASS, GRASS, TREES, TREES, TREES,
    GRASS, GRASS, WATER, WATER },
    { GRASS, WATER, WATER, TREES, GRASS, WATER, WATER, TREES, TREES, GRASS,
    GRASS, GRASS, GRASS, STONE }
};

#define TILE_SIZE 16
void drawworld() {
    const int tileswide = WIDTH / TILE_SIZE + 1;
    const int tilestall = HEIGHT / TILE_SIZE + 1;

    for(int y = 0; y < tilestall; y++) {
        for(int x = 0; x < tileswide; x++) {
            arduboy.drawBitmap(x * TILE_SIZE + mapx % TILE_SIZE, y * TILE_SIZE +
mapy % TILE_SIZE, tiles[world[y][x]], TILE_SIZE, TILE_SIZE, WHITE);

        }
    }

    arduboy.fillRect(0, 0, 48, 8, BLACK);
    arduboy.setCursor(0, 0);
    arduboy.print(mapx);
    arduboy.print(",");
    arduboy.print(mapy);
}

void titlescreen() {
    arduboy.setCursor(0, 0);
    arduboy.print("Title Screen\n");
    if(arduboy.justPressed(A_BUTTON)) {
        gamestate = GAME_PLAY;
    }
}

void playerinput() {
    if(arduboy.pressed(UP_BUTTON)) {
        mapy += 1;
    }
    if(arduboy.pressed(DOWN_BUTTON)) {
        mapy -= 1;
    }
    if(arduboy.pressed(LEFT_BUTTON)) {
        mapx += 1;
    }
    if(arduboy.pressed(RIGHT_BUTTON)) {
        mapx -= 1;
    }
}

```

```

void gameplay() {
    playerinput();
    drawworld();
    if(arduboy.justPressed(A_BUTTON)) {
        gamestate = GAME_OVER;
    }
}

void gameoverscreen() {
    arduboy.setCursor(0, 0);
    arduboy.print("Game Over Screen\n");
    if(arduboy.justPressed(A_BUTTON)) {
        gamestate = GAME_HIGH;
    }
}

void highscorescreen() {
    arduboy.setCursor(0, 0);
    arduboy.print("High Score Screen\n");
    if(arduboy.justPressed(A_BUTTON)) {
        gamestate = GAME_TITLE;
    }
}

void gameloop() {
    switch(gamestate) {
        case GAME_TITLE:
            titlescreen();
            break;
        case GAME_PLAY:
            gameplay();
            break;
        case GAME_OVER:
            gameoverscreen();
            break;
        case GAME_HIGH:
            highscorescreen();
            break;
    }
}

```

```

void setup() {
    arduboy.begin();
    arduboy.setFrameRate(45);
    arduboy.display();
    arduboy.initRandomSeed();
    arduboy.clear();
}

void loop() {
    if(!(arduboy.nextFrame())) {
        return;
    }
    arduboy.pollButtons();
    arduboy.clear();
    gameloop();
    arduboy.display();
}

```

Notice that every time we move the screen 16 pixels in any direction, the screen jumps back. If you move slowly and watch, you'll notice that the tiles align perfectly, even if there is a jump. The only thing that is wrong is that the next tiles are not displayed and instead, the same tiles are displayed over and over.

In order to display the next tiles, we need to figure out which tiles we are going to draw. We must keep track every time the map jumps around. We could do this with another variable, but we could also calculate this based off of `mapx` and `mapy`.

Remember from Part 7, whenever we divide an `int` in C++, it only returns whole numbers! Dividing gets rid of the remainder. We could divide the `mapx` and `mapy` by 16 to find out how many times we've jumped the map around.

In the `drawworld()` function, we should change what we're printing out:

```

arduboy.print(mapx / TILE_SIZE);
arduboy.print(",");
arduboy.print(mapy / TILE_SIZE);

```

Run this code and move the map around!

Remember that we are drawing the top-right of the map, so moving left and down is what we really want to focus on. When we use the left and down buttons to move, notice that it does what we want except the result of our calculations are negative... This isn't a big deal! There is a way to reverse it by subtracting the values from 0.

```

arduboy.print(0 - mapx / TILE_SIZE);
arduboy.print(",");
arduboy.print(0 - mapy / TILE_SIZE);

```

Now that we can calculate these values to keep track of how many times we *jump* the map around, we can add that to our `arduboy.drawBitmap()` function to offset which tile from world we are going to draw.

Currently, we are just using the `x` and `y` variables. Let's create new `const int` variables called `tilex` and `tiley` to hold the values of `x` and `y` plus the calculations, then use those instead of `x` and `y`.

```
for(int y = 0; y < tilestall; y++) {
  for(int x = 0; x < tileswide; x++) {
    const int tilex = x - mapx / TILE_SIZE;
    const int tiley = y - mapy / TILE_SIZE;
    arduboy.drawBitmap(x * TILE_SIZE + mapx % TILE_SIZE, y * TILE_SIZE +
mapy % TILE_SIZE, tiles[world[tiley][tilex]], TILE_SIZE, TILE_SIZE, WHITE);
  }
}
```

Just to clarify, to get the tile bitmap of the tile that is being drawn, we need to get the value inside of `world` at `(tilex,tiley)`, then use that value as the index of the `tiles` array.

```
tiles[ world[ tiley ][ tilex ] ]
```

Now, what I want you to do is compile your code and put it onto the Arduboy. (I'm including mine right below this.) Use the bottom and right buttons to move around. You should see the screen scroll, but what happens if you keep going and view parts of the map that don't exist, yet?

```
//DinoSmasher

#include <Arduboy2.h>
Arduboy2 arduboy;

#define GAME_TITLE 0
#define GAME_PLAY 1
#define GAME_OVER 2
#define GAME_HIGH 3
int gamestate = GAME_TITLE;

int mapx = 0;
int mapy = 0;

const unsigned char tiles[4][32] PROGMEM = {
  { 0xff, 0x7f, 0xfb, 0xff, 0xff, 0xbf, 0xff, 0xff, 0xf7, 0xff, 0xfd,
    0xff, 0xff, 0xf7, 0x7f, 0xff, 0xdf, 0xff, 0xff, 0xfb, 0x7f, 0xff, 0xff,
    0xff, 0xef, 0xfe, 0xff, 0xff, 0xfb, 0xff, 0x7f, 0xff },
```

```

    { 0x08, 0x10, 0x10, 0x08, 0x10, 0x08, 0x10, 0x10, 0x10, 0x08, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x20, 0x40, 0x40, 0x20, 0x00, 0x01, 0x02,
    0x02, 0x01, 0x02, 0x02, 0x01, 0x02, 0x21, 0x40, 0x40 },
    { 0xff, 0x1f, 0x5b, 0x3f, 0xeb, 0xdd, 0xff, 0xf7, 0xbb, 0xef, 0xfd,
    0x7f, 0xe3, 0xcb, 0xe3, 0xff, 0xff, 0xc7, 0x96, 0xc7, 0xff, 0xff, 0xef,
    0xfd, 0xff, 0xe3, 0xcb, 0xe3, 0xff, 0xff, 0x7b, 0xff },
    { 0xff, 0xdf, 0x7b, 0x3f, 0x9f, 0x6f, 0x77, 0xab, 0xdb, 0xd7, 0xcd,
    0x5f, 0xbf, 0x77, 0xff, 0xff, 0xff, 0xc1, 0xdc, 0xd3, 0xaf, 0x9f, 0xae,
    0xb0, 0xbb, 0xbd, 0xbd, 0xba, 0xd7, 0xcc, 0x63, 0xff }
};

#define WORLD_WIDTH    14
#define WORLD_HEIGHT   7
#define GRASS          0
#define WATER          1
#define TREES          2
#define STONE          3
int world[WORLD_HEIGHT][WORLD_WIDTH] = {
    { TREES, GRASS, GRASS, WATER, GRASS, GRASS, GRASS, TREES, GRASS, GRASS,
    GRASS, GRASS, GRASS, TREES },
    { GRASS, WATER, WATER, WATER, GRASS, WATER, GRASS, GRASS, GRASS, GRASS,
    GRASS, STONE, GRASS, GRASS },
    { GRASS, GRASS, GRASS, GRASS, GRASS, WATER, STONE, GRASS, GRASS, GRASS,
    TREES, GRASS, GRASS, GRASS },
    { STONE, GRASS, GRASS, STONE, TREES, WATER, WATER, WATER, GRASS, WATER,
    WATER, GRASS, TREES, GRASS },
    { GRASS, GRASS, GRASS, GRASS, TREES, GRASS, GRASS, GRASS, TREES, WATER,
    GRASS, GRASS, STONE, TREES },
    { GRASS, GRASS, GRASS, WATER, STONE, GRASS, GRASS, TREES, TREES, TREES,
    GRASS, GRASS, WATER, WATER },
    { GRASS, WATER, WATER, TREES, GRASS, WATER, WATER, TREES, TREES, GRASS,
    GRASS, GRASS, GRASS, STONE }
};

#define TILE_SIZE 16
void drawworld() {
    const int tileswide = WIDTH / TILE_SIZE + 1;
    const int tilestall = HEIGHT / TILE_SIZE + 1;

    for(int y = 0; y < tilestall; y++) {
        for(int x = 0; x < tileswide; x++) {
            const int tilex = x - mapx / TILE_SIZE;
            const int tiley = y - mapy / TILE_SIZE;
            arduboy.drawBitmap(x * TILE_SIZE + mapx % TILE_SIZE, y * TILE_SIZE +
            mapy % TILE_SIZE, tiles[world[tiley][tilex]], TILE_SIZE, TILE_SIZE,
            WHITE);
        }
    }
}

```

```

    }

    arduboy.fillRect(WIDTH / 2 - 8, HEIGHT / 2 - 8, 16, 16, BLACK);
    arduboy.fillRect(0, 0, 48, 8, BLACK);
    arduboy.setCursor(0, 0);
    arduboy.print(0 - mapx / TILE_SIZE);
    arduboy.print(",");
    arduboy.print(0 - mapy / TILE_SIZE);
}

void titlescreen() {
    arduboy.setCursor(0, 0);
    arduboy.print("Title Screen\n");
    if(arduboy.justPressed(A_BUTTON)) {
        gamestate = GAME_PLAY;
    }
}

void playerinput() {
    if(arduboy.pressed(UP_BUTTON)) {
        mapy += 1;
    }
    if(arduboy.pressed(DOWN_BUTTON)) {
        mapy -= 1;
    }
    if(arduboy.pressed(LEFT_BUTTON)) {
        mapx += 1;
    }
    if(arduboy.pressed(RIGHT_BUTTON)) {
        mapx -= 1;
    }
}

void gameplay() {
    playerinput();
    drawworld();
    if(arduboy.justPressed(A_BUTTON)) {
        gamestate = GAME_OVER;
    }
}

void gameoverscreen() {
    arduboy.setCursor(0, 0);
    arduboy.print("Game Over Screen\n");
    if(arduboy.justPressed(A_BUTTON)) {
        gamestate = GAME_HIGH;
    }
}

```

```

void highscorescreen() {
    arduboy.setCursor(0, 0);
    arduboy.print("High Score Screen\n");
    if(arduboy.justPressed(A_BUTTON)) {
        gamestate = GAME_TITLE;
    }
}

```

```

void gameloop() {
    switch(gamestate) {
        case GAME_TITLE:
            titlescreen();
            break;

        case GAME_PLAY:
            gameplay();
            break;

        case GAME_OVER:
            gameoverscreen();
            break;

        case GAME_HIGH:
            highscorescreen();
            break;
    }
}

```

```

void setup() {
    arduboy.begin();
    arduboy.setFrameRate(45);
    arduboy.display();
    arduboy.initRandomSeed();
    arduboy.clear();
}

```

```

void loop() {
    if(!(arduboy.nextFrame())) {
        return;
    }

    arduboy.pollButtons();
    arduboy.clear();
    gameloop();
    arduboy.display();
}

```





Isn't it cool that you can navigate outside of our `world` array? Like I said near the end of Part 8, what's being displayed is data from other variables and other parts of the ArduBoy's memory.

## 6. Bounding the map

Next, what we should do is prevent the player from seeing all this extra stuff outside of our `world`. We also should prevent them from moving the black square outside of it, as well! Let's start by choosing to only draw the tiles that are in range during our `drawworld()` function's for loops.

Remember, we are already taking into consideration which tiles are being drawn from `world` with `tilex` and `tiley`. What we could do is draw the tile if `tilex/tiley` are positive (greater than -1) and less than the `WORLD_WIDTH/WORLD_HEIGHT`.

```
if(tilex >= 0 && tiley >= 0 && tilex < WORLD_WIDTH && tiley <
WORLD_HEIGHT) {
    arduboy.drawBitmap(x * TILE_SIZE + mapx % TILE_SIZE, y * TILE_SIZE +
mapy % TILE_SIZE, tiles[world[tiley][tilex]], TILE_SIZE, TILE_SIZE,
WHITE);
}
```

Now, to make sure the player cannot move outside of our world, we need to realize that when the world map is at (0, 0), the player is not at (0, 0). That is because we're drawing it in the middle of what's on screen. Incidentally, we are also drawing the character in the drawworld() function. Let's remove that arduboy.fillRect() call and do in a better place.

Let's make a new function right above drawworld() and call it drawplayer(). We want to also reference it inside of gameplay(), right after drawworld(). Remember, if we draw something to the screen before drawworld(), then it will be drawn prior, and drawn under the tiles. We want to see the player on top of them.

Alongside drawworld(), we should define size of our player as 16 using #define PLAYER\_SIZE 16. We can use that when drawing our rectangle similar to what we did earlier:

```
arduboy.fillRect(WIDTH / 2 - PLAYER_SIZE / 2, HEIGHT / 2 - PLAYER_SIZE / 2, PLAYER_SIZE, PLAYER_SIZE, BLACK);
```

That is a little hard to read, so let's actually make *another* definition! Let's store the rectangle's X offset and rectangle's Y offset and use those values when drawing our rectangle.

```
#define PLAYER_SIZE      16
#define PLAYER_X_OFFSET  WIDTH / 2 - PLAYER_SIZE / 2
#define PLAYER_Y_OFFSET  HEIGHT / 2 - PLAYER_SIZE / 2
void drawplayer() {
    arduboy.fillRect(PLAYER_X_OFFSET, PLAYER_Y_OFFSET, PLAYER_SIZE,
    PLAYER_SIZE, BLACK);
}
```

Storing the PLAYER\_X\_OFFSET and PLAYER\_Y\_OFFSET can come in handy when figuring out if the player's character is touching the edge of the world.

For instance, if we are letting the map slide down the screen, we don't want the top of it to slide below the top of the player's square. The top of the map is at mapy and the top of the player's square is at PLAYER\_Y\_OFFSET. This works with the X variables, too. Let's update the playerinput() functions to include these restrictions.

```
void playerinput() {
    if(arduboy.pressed(UP_BUTTON)) {
        if(mapy < PLAYER_Y_OFFSET) {
            mapy += 1;
        }
    }
    if(arduboy.pressed(DOWN_BUTTON)) {
        mapy -= 1;
    }
}
```

```

    if(arduboy.pressed(LEFT_BUTTON)) {
        if(mapx < PLAYER_X_OFFSET) {
            mapx += 1;
        }
    }
    if(arduboy.pressed(RIGHT_BUTTON)) {
        mapx -= 1;
    }
}

```

Testing this code, you will see that it works, but only on the top and left-side of the map.

For the bottom of the map, we only want the down button to move the map up if the bottom of the map is still below the bottom of the player's square. These can be calculated with `mapy + TILE_SIZE * WORLD_HEIGHT` and `PLAYER_Y_OFFSET + PLAYER_SIZE`. You can do this for the X axis, too!

```

void playerinput() {
    if(arduboy.pressed(UP_BUTTON)) {
        if(mapy < PLAYER_Y_OFFSET) {
            mapy += 1;
        }
    }
    if(arduboy.pressed(DOWN_BUTTON)) {
        if(PLAYER_Y_OFFSET + PLAYER_SIZE < mapy + TILE_SIZE * WORLD_HEIGHT) {
            mapy -= 1;
        }
    }
    if(arduboy.pressed(LEFT_BUTTON)) {
        if(mapx < PLAYER_X_OFFSET) {
            mapx += 1;
        }
    }
    if(arduboy.pressed(RIGHT_BUTTON)) {
        if(PLAYER_X_OFFSET + PLAYER_SIZE < mapx + TILE_SIZE * WORLD_WIDTH) {
            mapx -= 1;
        }
    }
}

```

Here is our completed code!!

```
//DinoSmasher
```

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

```
Arduboy2 arduboy;
```

```
#define GAME_TITLE 0
```

```
#define GAME_PLAY 1
```

```
#define GAME_OVER 2
```

```
#define GAME_HIGH 3
```

```
int gamestate = GAME_TITLE;
```

```
#define PLAYER_SIZE 16
```

```
int mapx = 0;
```

```
int mapy = 0;
```

```
const unsigned char tiles[4][32] PROGMEM = {
```

```
{ 0xff, 0x7f, 0xfb, 0xff, 0xff, 0xbf, 0xff, 0xff, 0xf7, 0xff, 0xfd,  
0xff, 0xff, 0xf7, 0x7f, 0xff, 0xdf, 0xff, 0xff, 0xfb, 0x7f, 0xff, 0xff,  
0xff, 0xef, 0xfe, 0xff, 0xff, 0xfb, 0xff, 0x7f, 0xff },
```

```
{ 0x08, 0x10, 0x10, 0x08, 0x10, 0x08, 0x10, 0x10, 0x10, 0x08, 0x00,  
0x00, 0x00, 0x00, 0x00, 0x00, 0x20, 0x40, 0x40, 0x20, 0x00, 0x01, 0x02,  
0x02, 0x01, 0x02, 0x02, 0x01, 0x02, 0x21, 0x40, 0x40 },
```

```
{ 0xff, 0x1f, 0x5b, 0x3f, 0xeb, 0xdd, 0xff, 0xf7, 0xbb, 0xef, 0xfd,  
0x7f, 0xe3, 0xcb, 0xe3, 0xff, 0xff, 0xc7, 0x96, 0xc7, 0xff, 0xff, 0xef,  
0xfd, 0xff, 0xe3, 0xcb, 0xe3, 0xff, 0xff, 0x7b, 0xff },
```

```
{ 0xff, 0xdf, 0x7b, 0x3f, 0x9f, 0x6f, 0x77, 0xab, 0xdb, 0xd7, 0xcd,  
0x5f, 0xbf, 0x77, 0xff, 0xff, 0xff, 0xc1, 0xdc, 0xd3, 0xaf, 0x9f, 0xae,  
0xb0, 0xbb, 0xbd, 0xbd, 0xba, 0xd7, 0xcc, 0x63, 0xff }
```

```
};
```

```
#define WORLD_WIDTH 14
```

```
#define WORLD_HEIGHT 7
```

```
#define GRASS 0
```

```
#define WATER 1
```

```
#define TREES 2
```

```
#define STONE 3
```

```
int world[WORLD_HEIGHT][WORLD_WIDTH] = {
```

```
{ TREES, GRASS, GRASS, WATER, GRASS, GRASS, GRASS, TREES, GRASS, GRASS,  
GRASS, GRASS, GRASS, TREES },
```

```
{ GRASS, WATER, WATER, WATER, GRASS, WATER, GRASS, GRASS, GRASS, GRASS,  
GRASS, STONE, GRASS, GRASS },
```

```
{ GRASS, GRASS, GRASS, GRASS, GRASS, WATER, STONE, GRASS, GRASS, GRASS,  
TREES, GRASS, GRASS, GRASS },
```

```
{ STONE, GRASS, GRASS, STONE, TREES, WATER, WATER, WATER, GRASS, WATER,  
WATER, GRASS, TREES, GRASS },
```

```
{ GRASS, GRASS, GRASS, GRASS, TREES, GRASS, GRASS, GRASS, TREES, WATER,  
GRASS, GRASS, STONE, TREES },
```

```

    { GRASS, GRASS, GRASS, WATER, STONE, GRASS, GRASS, TREES, TREES, TREES,
    GRASS, GRASS, WATER, WATER },
    { GRASS, WATER, WATER, TREES, GRASS, WATER, WATER, TREES, TREES, GRASS,
    GRASS, GRASS, GRASS, STONE }
};

#define PLAYER_SIZE      16
#define PLAYER_X_OFFSET  WIDTH / 2 - PLAYER_SIZE / 2
#define PLAYER_Y_OFFSET  HEIGHT / 2 - PLAYER_SIZE / 2
void drawplayer() {
    arduboy.fillRect(PLAYER_X_OFFSET, PLAYER_Y_OFFSET, PLAYER_SIZE,
    PLAYER_SIZE, BLACK);
}

#define TILE_SIZE      16
void drawworld() {
    const int tileswide = WIDTH / TILE_SIZE + 1;
    const int tilestall = HEIGHT / TILE_SIZE + 1;

    for(int y = 0; y < tilestall; y++) {
        for(int x = 0; x < tileswide; x++) {
            const int tilex = x - mapx / TILE_SIZE;
            const int tiley = y - mapy / TILE_SIZE;
            if(tilex >= 0 && tiley >= 0 && tilex < WORLD_WIDTH && tiley <
WORLD_HEIGHT) {
                arduboy.drawBitmap(x * TILE_SIZE + mapx % TILE_SIZE, y * TILE_SIZE
+ mapy % TILE_SIZE, tiles[world[tiley][tilex]], TILE_SIZE, TILE_SIZE,
WHITE);
            }
        }
    }

    arduboy.fillRect(0, 0, 48, 8, BLACK);
    arduboy.setCursor(0, 0);
    arduboy.print(0 - mapx / TILE_SIZE);
    arduboy.print(",");
    arduboy.print(0 - mapy / TILE_SIZE);
}

void titlescreen() {
    arduboy.setCursor(0, 0);
    arduboy.print("Title Screen\n");
    if(arduboy.justPressed(A_BUTTON)) {
        gamestate = GAME_PLAY;
    }
}

```

```

void playerinput() {
    if(arduboy.pressed(UP_BUTTON)) {
        if(mapy < PLAYER_Y_OFFSET) {
            mapy += 1;
        }
    }
    if(arduboy.pressed(DOWN_BUTTON)) {
        if(PLAYER_Y_OFFSET + PLAYER_SIZE < mapy + TILE_SIZE * WORLD_HEIGHT) {
            mapy -= 1;
        }
    }
    if(arduboy.pressed(LEFT_BUTTON)) {
        if(mapx < PLAYER_X_OFFSET) {
            mapx += 1;
        }
    }
    if(arduboy.pressed(RIGHT_BUTTON)) {
        if(PLAYER_X_OFFSET + PLAYER_SIZE < mapx + TILE_SIZE * WORLD_WIDTH) {
            mapx -= 1;
        }
    }
}

void gameplay() {
    playerinput();
    drawworld();
    drawplayer();

    if(arduboy.justPressed(A_BUTTON)) {
        gamestate = GAME_OVER;
    }
}

void gameoverscreen() {
    arduboy.setCursor(0, 0);
    arduboy.print("Game Over Screen\n");
    if(arduboy.justPressed(A_BUTTON)) {
        gamestate = GAME_HIGH;
    }
}

void highscorescreen() {
    arduboy.setCursor(0, 0);
    arduboy.print("High Score Screen\n");
    if(arduboy.justPressed(A_BUTTON)) {
        gamestate = GAME_TITLE;
    }
}

```

```

void gameloop() {
    switch(gamestate) {
        case GAME_TITLE:
            titlescreen();
            break;

        case GAME_PLAY:
            gameplay();
            break;

        case GAME_OVER:
            gameoverscreen();
            break;

        case GAME_HIGH:
            highscorescreen();
            break;
    }
}

void setup() {
    arduboy.begin();
    arduboy.setFrameRate(45);
    arduboy.display();

    arduboy.initRandomSeed();

    arduboy.clear();
}

void loop() {
    if(!(arduboy.nextFrame())) {
        return;
    }
    arduboy.pollButtons();
    arduboy.clear();
    gameloop();
    arduboy.display();
}

```

Upload your code to your Arduboy and play around with it! If you want a challenge while waiting for the next tutorial, you should try to make a bigger map and put a cool design into it.

To be continued...