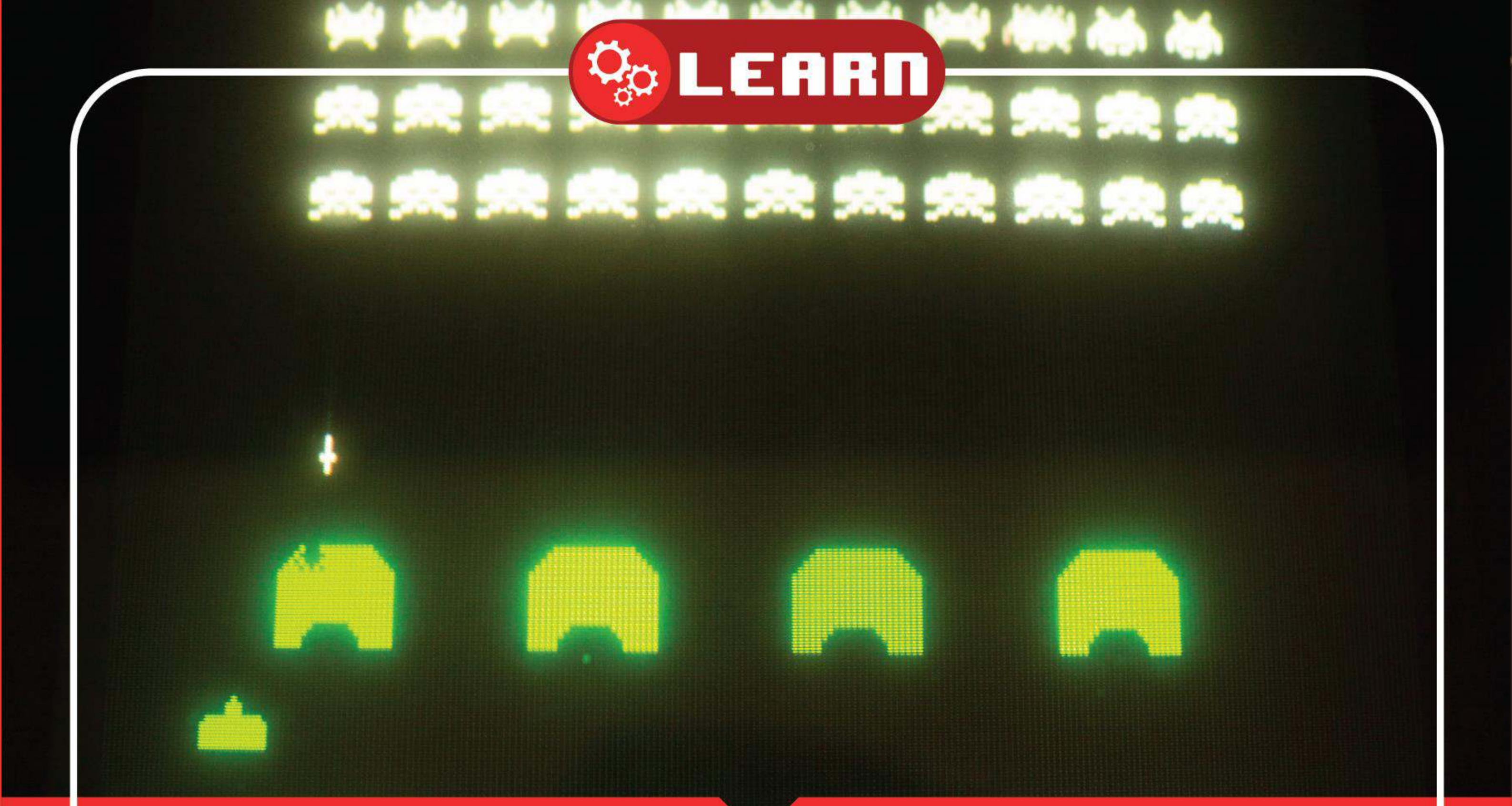
LEARN



CODING SPACE INVADERS

PART 1

With 8BitCADE

Compatible with:

8BitCADE XL

AND

8BitCADE

Contents

Tutorial by 8BitCADE adapted from the amazing work by Xtronical

The original guide can be followed online at
https://www.xtronical.com/projects/space-invaders/

Lesson Number: Game Development 1a

Lesson Title: Plotting Graphics and Basic Movement
Code: Full Code for Lesson
System: Arduboy + Project ABE

Prerequisites to completing this tutorial

1. **Completed tutorials 1-5 in Getting Started in Coding**
2. Know how to write code in either **Arduino IDE** or **Project ABE online Emulator**
3. Know how to draw pixel art by watching **Pixel Art Tutorial Space Invader**
4. Know how to convert pixel art into hexadecimal code by watching **Converting Pixel Art into Hexadecimal Code**

Step 1 ADD ANY INFORMATION AND SET-UP LIBRARY FILES

Type the following code into line 1-6

```
1.  /*   www.pixilart.com for sprite generation
2.  http://javl.github.io/image2cpp/ for image conversion
3.  */
4.  //libraries for graphics/display
5.  #include <Arduboy2.h>
6.  Arduboy2 arduboy;
```

Explanation (line 1-6)

/* and */ is code syntax to indicate that the information **BETWEEN** the syntax is to be ignored by the compiler within the IDE software, and that it is just information for the programmer. If demonstrating good coding habits then you might include who created the code, date, version, any licensing or hardware requirements, like button, sound, and screen connections. All I have included is the links to Pixel Art website for sprite (graphics) creation and the website for conversion of your image to hexadecimal code.
// is syntax for IGNORE everything on this line, because like the above syntax, the information is only for the programmer.

#include <Arduboy2.h> Libraries are files which provide sketches with extra **functionality** (e.g. the ability to control an LED matrix, or read a sensor, etc.). To use a library we normally download it and add it to our IDE, unless it is a standard library and we already have it as a part of the IDE. To use a library you need to add a **#include** statement at the top of the sketch. This will allow the Arduino IDE to locate and use the additional code within the library. This must be done for each library. **Arduino** Libraries consist of a minimum of **two files**: a **header file** (with the extension . h) and a source **file** (with extension . cpp). The **header file** (. h) contains all the functions for the library (also called Classes), which is a listing of everything that's inside, including **commands** which enable us to use all the libraries **functions**. The .cpp file has all the source code for the commands/functions to work. So when we pick a command or function like **arduboy.clear, the compiler looks at the header file for the function (class), arduboy.clear** and then looks at the **.cpp file** to execute the code related to that **function (class).** This is also referred to as **Object Orientated Programming (OOP). OOP is faster and easier** to use and allows people learning to program to use more complex code, written by another programmer **(for example, in the form of a library),** by executing basic commands. **In programming terms**, a library has lots of classes, classes act like a template for objects, and an object is referred to as an instance of a class. When the individual objects are created, they inherit all the variables and functions from the class. We create/use these objects when programming to use blocks of code to make life easier.

For more information:
https://mlxxxp.github.io/documents/Arduino/libraries/Arduboy2/Doxygen/html/index.html,
https://www.w3schools.com/cpp/cpp_oop.asp.

Arduboy2 arduboy; // Arduboy 2 is a library with many classes (functions). This line of code creates a variable (object) called **arduboy**, which inherits all the functions and variables of Arduboy2 library and its classes.

Step 2 SET_UP TIMING USING FRAME RATE

Type the following code into line 17-25

```
17. void setup() { // put your setup code here, to run once:
18. arduboy.setFrameRate(1);
19. arduboy.begin();
20. }
21.
22. void loop() { // put your main code here, to run repeatedly:
23. if (!arduboy.nextFrame()) {
24. return;
25. }
```

Explanation (line 17-25)

The setup() function is called when a sketch starts. Use it to initialize variables, pin modes, start using libraries, etc. The setup() function will only run once, after each power up or reset of the Arduino board. When we develop games we set a frame rate. Most game consoles can play games at 30 or 60 frames per second. That means that the screen is updated 30 or 60 times in one second. The more instructions that a computer has to run in a second, the slower that frame rate the game can be run at without problems occurring. If the frame rate is too slow, some games are unplayable. During the **void setup()** function, we can declare what frame rate we want our Arduboy game to use by calling **arduboy.setFrameRate()** function. To make the Arduboy game run very slowly we use a low framerate and to run faster we use a higher frame rate. The **void loop()** function does precisely what its name suggests, and loops consecutively, allowing your program to change and respond. Use it to actively control the Arduino board. **if (!arduboy.nextFrame()) { return;** is called to check if it's too early to display the next frame (timing). If it is, the Arduboy will loop again until it's ready for the next frame, making sure that the frames run as smoothly as possible.

Overview: Well done you have now completed a basic gaming sketch layout, the bare bones! This sketch will not actually do much so we need to add some functionality.

```
/* www.pixilart.com for sprite generation; http://javl.github.io/image2cpp/ for image conversion
  This code displays Space Invaders and moves a sprite from left to right.
*/
//libraries for graphics/display
#include <Arduboy2.h>
Arduboy2 arduboy;

void setup() { // put your setup code here, to run once:
  arduboy.setFrameRate(1);
  arduboy.begin();
}

void loop() { // put your main code here, to run repeatedly:
  if (!arduboy.nextFrame()) {
    return;
  }
 }
```

Step 3: STORING THE GRAPHIC (SPRITE) IMAGE IN PROGRAMME MEMORY

Type the following code into line 8-12

```
8.  // Invader Middle Sprite W11,H8
9.  const unsigned char PROGMEM InvaderMiddleGfx [] =
10. {
11. 0x1e, 0xb8, 0x7d, 0x36, 0x3c, 0x3c, 0x3c, 0x36, 0x7d, 0xb8, 0x1e
12. };
```

Explanation (line 8-12)

The line: **static const unsigned char PROGMEM InvaderMiddleGfx** sets up the start of the graphics definition. const unsigned char means this is a **character data type used to store characters in memory. Although please note, these characters are represented in memory as numbers,** these numbers that are unsigned (cannot be negative) and that this is const (constant), which means you cannot change like a normal variable. PROGMEM means the data is stored in program memory (ROM, 32K) and not RAM (only about 2K). **PROGMEM is used** because graphics often take up a significant amount of memory and the small amount in the arduino (Atmel) processors would be quickly used up so we use the more generous program memory to store our images.

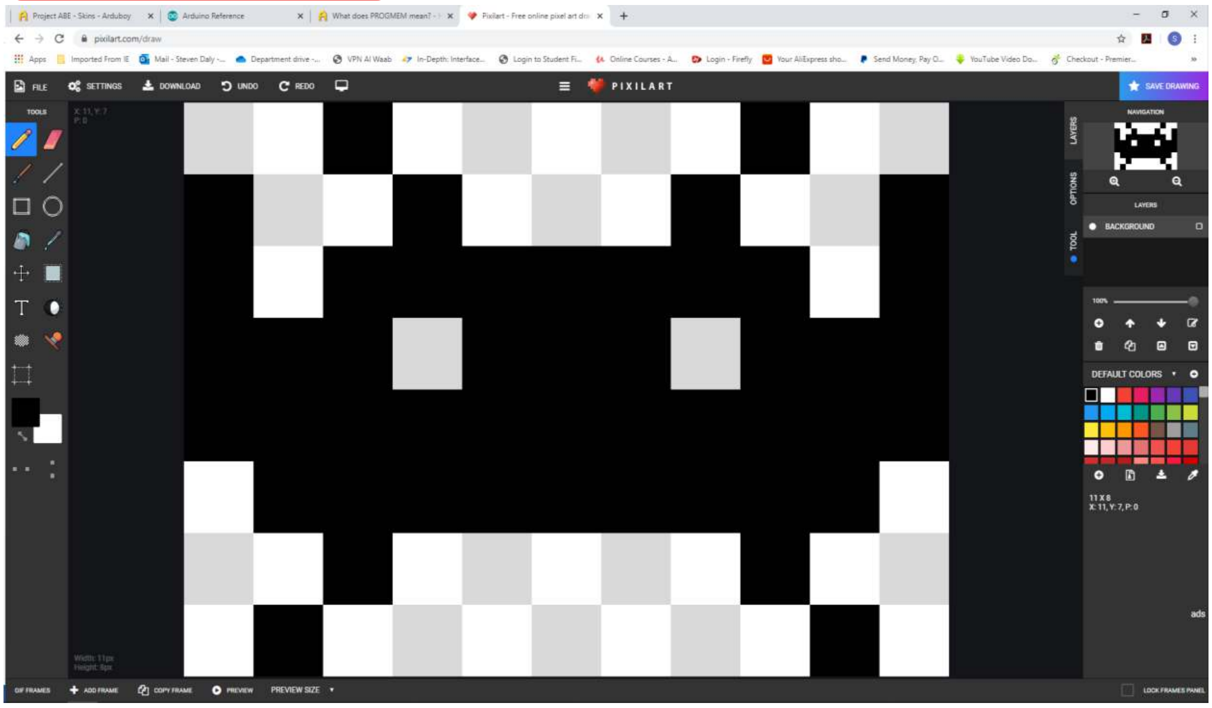
The line: 0x1e, 0xb8, 0x7d, 0x36, 0x3c, 0x3c, 0x3c, 0x36, 0x7d, 0xb8, 0x1e is the actual graphic or sprite which was drawn using pixilart.com and then converted to hexadecimal code using image2cpp, a website capable of converting .png files into hexadecimal code. The graphic is created in https://www.pixilart.com/draw. As you can see the space invader called Invader Middle GFX is 11 pixels wide and 8 pixels high. We need to send information to the OLED Screen by switching white pixels on or off. Looking at the space invader image, imagine the top row of pixels were represented by 000s and 111s. 0 means pixel off and 1 means pixel on. We must represent this information in 1 byte blocks at a time which is no more than 8 numbers (ones or zeros) in each block until we have covered the whole graphic. The top of the graphic would start off as B0010000 B100….. we have to add more zeros to the end of the second block as the information is stored in 8 bit blocks (1 byte) so the top row would finally be B00100000,B10000000, (extra zeros added to complete the 2 bytes (2 lots of 8 bits).

Binary code for the image

B00100000,B10000000,
B00010001,B00000000,
B00111111,B10000000,
B01101110,B11000000,
B11111111,B11100000,
B10111111,B10100000,
B10100000,B10100000,
B00011011,B00000000

Converted to hexadecimal coding.

0x1e, 0xb8, 0x7d, 0x36, 0x3c
, 0x3c, 0x3c, 0x36, 0x7d, 0xb
8, 0x1e



1. We tend not to store this information in binary in our programme, as it is more efficient to store it in hexadecimal, less memory is taken up. To convert it to hexadecimal we can use http://javl.github.io/image2cpp/ for image conversion.

Step 4 INITIAL POSITION OF THE SPRITE

Type the following code into line 14-15

```
14. // move sprite left to right
15. int XPos = 0; // integer for the initial position of the sprite
```

Explanation (line 14-15)

Line: Int means integer, Integers are your primary data-type variable for number storage. An integer variable stores a 16-bit (2-byte) value. This yields a range of -32,768 to 32,767. Xpos is the name given to the variable, this is selected by the programmer and the name helps us remember what the variable is related to. XPos=0 is used to store the initial position of the sprite (sprite is another word for graphic).

Once the sprite starts moving this variable will change its value as it is holding the X position.

STEP 5 CLEAR SCREEN ADD TEXT

Type the following code into line 26-29

```
26. arduboy.clear();
27. arduboy.setTextColor(WHITE);
28. arduboy.setCursor (20, 0);
29. arduboy.print("Space Invaders");
```

Explanation (line 26-29)

arduboy.clear is a function within the arduboy2 library which clears the OLED screen. We then set the text colour white (only colour we can use as it is a black and white screen), set the cursor (where we want to print text on our 128 x 64 pixel screen) using X, and Y values, so 20 in X means 20 pixels along on the X axis (width of the screen) and 0 along on the Y axis, and then give the command to print the words Space Invaders, using the arduboy print to screen function. All printed text must be in speech marks.

STEP 6 DRAW AND DISPLY SPRITE AS A BITMAP

Type the following code into line 30-31

```
30. arduboy.drawBitmap(XPos, 20, InvaderMiddleGfx, 11, 8, WHITE); // XPos is the integer to move x,y pos, name of sprite, size of sprite, colour
31. arduboy.display();
```

Explanation (line 30-31)

Plotting the image; the line: **arduboy.drawBitmap(XPos,20,InvaderMiddleGfx,11,8,WHITE);** plots the graphic to screen. It has 6 arguments or parameters which must be added. The first two parameters are X and Y positions on the OLED screen. **XPos** is not a static position but a variable, initially 0. As **XPos** gets updated (later on in the code) the graphic will start moving in the X direction. Y position is set to "20", moving the graphic down the screen by 20 pixels. The next parameter is the name of the graphic, **InvaderMiddleGfx**, then the graphics data variable related to size in pixels, the "11" is the width of the graphic and "8" the height. The last parameter is not useful for use as we only have the one colour but we must still include it. **arduboy.display** is a function in the library to send data to the OLED display.

STEP 7 MOVE THE SPRITE ONE PIXEL AT A TIME UPDATING THE DRAWING COMMAND LINE 30

Type the following code into line 32-35

```
32. XPos += 1; // move every pixel
33. if (XPos > 127) // move sprite all the way across the screen
34. XPos = 0; // reset back to start of screen
35. }
```

Explanation (line 32-35)

We set a **XPos** variable to store the current position of the graphic in line 15. Now we want to increment this. When the graphic reaches the end of the screen we reset it back to the start of the screen and repeat forever. The screen is 128 pixels long and 64 pixels wide. So when we get to 127 pixels we need to go back to the beginning of the screen. **XPos += 1**; means add 1 to XPos every loop of **void loop**. The next line of code means that **if XPos** is **>** more than 127 than reset it back to zero. This will essentially bring the sprite back to its original starting position and XPos will be back to 0. The original starting position of the sprite is actually determined by line 30 as it specifies within its argument (variables) the X and Y position of the sprite. This is the actual starting point. XPos moves the sprite from this point.

Overview – Xpos is a variable which is being increased by one pixel every loop. This variable relates to the X position of the graphical sprite being drawn using two frames (line 30) each loop, once X reaches 127 it will go back to 0.

Final Code

```
2.  /*www.pixilart.com for sprite generation
3.  http://javl.github.io/image2cpp/ for image conversion
4.  */
5.  //libraries for graphics/display
6.  #include <Arduboy2.h>
7.  Arduboy2 arduboy;
8.
9.  // Invader Middle Sprite W11,H8
10. const unsigned char PROGMEM InvaderMiddleGfx [] =
11. {
12. 0x1e, 0xb8, 0x7d, 0x36, 0x3c, 0x3c, 0x3c, 0x36, 0x7d, 0xb8, 0x1e
13. };
14.
15. // move sprite left to right
16. int XPos = 0; // integer for the initial position of the sprite
17.
18. void setup() { // put your setup code here, to run once:
19. arduboy.setFrameRate(1);
20. arduboy.begin();
21. }
22.
23. void loop() { // put your main code here, to run repeatedly:
24. if (!arduboy.nextFrame()) {
25. return;
26. }
27. arduboy.clear();
28. arduboy.setTextColor(WHITE);
29. arduboy.setCursor (20, 0);
30. arduboy.print("Space Invaders");
31. arduboy.drawBitmap(XPos, 20, InvaderMiddleGfx, 11, 8, WHITE); // XPos is the integer t
    o move x,y pos, name of sprite, size of sprite, colour
32. arduboy.display();
33. XPos += 1; // move every one pixel
34. if (XPos > 127) // move sprite all the way across the screen
35. XPos = 0; // reset back to start of screen
36. }
```

Lesson Number: Game Development 1b

Lesson Title: Plotting Graphics and 2 Frame Movement
Code: Full Code for Lesson
System: Arduboy + Project ABE

Prerequisites to completing this tutorial

1. Game Development 1a
2. Pixel

Step 1 NAMING VARIABLES

Type the following code into line 1-14

```
1.  /* www.pixilart.com for sprite generation
2.    http://javl.github.io/image2cpp/ for image conversion
3.    This code displays Space Invaders and moves a sprite from left to right.
4.  */
5.  //libraries for graphics/display
6.  #include <Arduboy2.h>
7.  Arduboy2 arduboy;
8.
9.  constexpr uint8_t frameCount = 2;      // The number of frames in the image
10. constexpr uint8_t firstFrame = 0;      // The first frame index (always 0)
11. constexpr uint8_t lastFrame = (frameCount - 1);     // The last frame index (one less than the number of frames)
12.
13. uint8_t currentFrame = firstFrame;      // The current frame being drawn
14.
```

Explanation (line 9-13)

Line 9 and 10: **constexpr uint8_t frameCount = 2** and **constexpr uint8_t firstFrame**; These are variables written to memory using a data type. The variable is used to record to memory, the number of frames in the graphic, as shown in line 16, which in this sketch is the maximum of 2 (frame 1 and frame2). const, means not changing as this variable value will not change while the code is running, **expr**, means expression. uint8_t means an unsigned integer data **type** (positive values only) requiring only 8 bit memory (00000000) because the size of the data being handled is in 8 bit blocks, and frameCount and firstFrame are the variable names given by the programmer so this variable can be referred to later in the code. Having all these constraints on these variables and data types does not tie up much memory which graphics can quickly do!

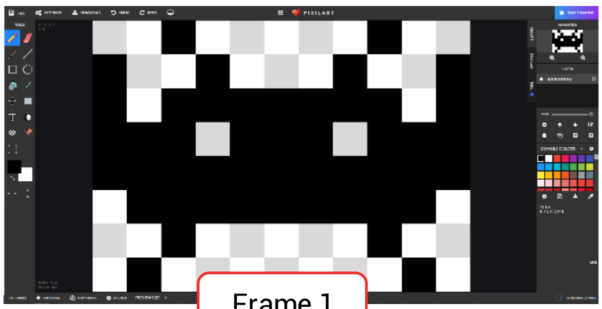
Step 2 ADD ADDITIONAL GRAPHIC IN AN ARRAY

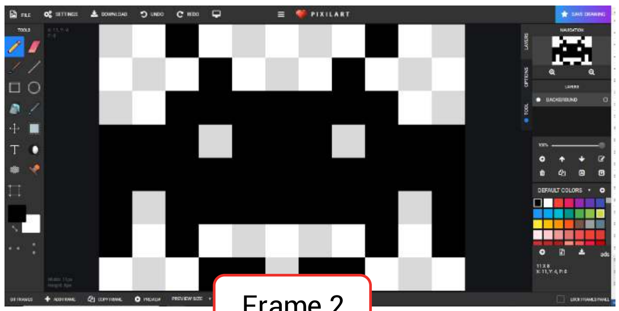
Type the following code into line 15-38

```
15. // Sprites
16. static const unsigned char PROGMEM InvaderMiddleGfx [] = {
17.   11, 8,                                 // width, height,pixel size of image
18.   0x1e, 0xb8, 0x7d, 0x36, 0x3c, 0x3c, 0x3c, 0x36, 0x7d, 0xb8, 0x1e,// Frame 1
19.
20.   0x78, 0x18, 0x7d, 0xb6, 0xbc, 0x3c, 0xbc, 0xb6, 0x7d, 0x18, 0x78,// Frame 2
21. };
22.
23. // move sprite left to right
24. int XPos = 0; // integer for the initial position of the sprite
25.
26. void setup() { // put your setup code here, to run once:
27.   arduboy.setFrameRate(1);
28.   arduboy.begin();
29. }
30.
31. void loop() { // put your main code here, to run repeatedly:
32.   arduboy.clear();
33.   arduboy.setTextColor(WHITE);
34.   arduboy.setCursor (20, 0);
35.   arduboy.print("Space Invaders");
36. if (!arduboy.nextFrame())              // Limit the frame rate (default 60fps)
37.     return;
38.
```

Explanation (line 16-21)

Most of the code for this section is discussed in Lesson Number: Game Development 1. What has changed here is that we have on variable InvaderMiddleGfx being written to programme memory, as 2 frames, each one is an array (list) of data. When the code is executed, it will treat the first array as Frame 0 (even though I called it in my notes // Frame 1) and the second Frame 1. The frame arrays are written in hexadecimal but the images for these arrays are below. Characters like our invader can have many frames to communicate movement. Just add more frames here but don't forget to adjust the number of frames in line 9, frame count. All the frames were drawn using pixilart.com and converted using the image to cpp website.


Frame 1


Frame 2

Step 3 INTRODUCING POLL BUTTONS

Type the following code into line 39

```
39.   arduboy.pollButtons(); // Update the button state;
```

Explanation (line 39)

Although we are not actually using this piece of code yet, it's a useful arduboy function to discuss at this stage. The Arduboy2 library has a few classes (functions) for game developers to check the state (on/off) of buttons on a game console so if a button is pressed it can action that command, for example, start a game, end a game, move left, right, shoot, go to a high score screen, turn volume off, reset the game. It really depends what you want the buttons to do. Hence arduboy.pollbuttons checks the button state and is executed at the start of each frame.

Step 4 IF STATEMENT TO LOOP THROUGH FRAMES TO MAKE THE ANIMATION WORK

Type the following code into line 40-51

```
40.
41.   if (arduboy.everyXFrames(1)) // If 15 frames have passed (at 60fps that's around 1/4 of a second)
42.   {
43.     if (currentFrame < lastFrame) // If the current frame is less than the last frame
44.     {
45.       ++currentFrame;       // Increment the current frame
46.     }
47.     else  // Otherwise
48.     {
49.       currentFrame = firstFrame;  // Cycle back to the start
50.     }
51. }
```

Explanation (line 40-51)

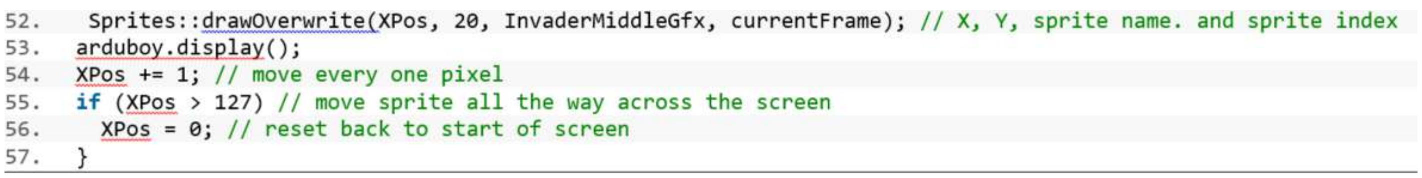
Line 41, if statement is a conditional statement with parentheses (brackets) which contain the conditions to be evaluated (checked). Symbols are used with these conditional statements to identify the relationship between variables. For example:

x == y (x is equal to y)
x != y (x is not equal to y)
x < y (x is less than y)
x > y (x is greater than y)
x <= y (x is less than or equal to y)
x >= y (x is greater than or equal to y)

arduboy.evereyXFrame(1) is a class within arduboy2 library which checks the number of frames has passed in the sketch. The only parameter in the brackets is the number of desired frames which once passed, should trigger an action, do something. We call this a true and false statement. Here we trigger an action once 1 frame has passed because we only have 2 frames in the whole animation. Line 43, If (currentFrame <lastFrame) is stating, if we have not reached the last frame yet then increase the current frame by one frame, as seen on line 45 ++currentFrame. The syntax ++ means increase by one. Else means that if none of the above is true, then currentFrame = firstFrame, essentially go back to the start, which is Frame 0.

Step 5 DRAW GRAPHIC USING SPRITE FUNCTION

Type the following code into line 52-57



```
52.   Sprites::drawOverwrite(XPos, 20, InvaderMiddleGfx, currentFrame); // X, Y, sprite name. and sprite index
53.   arduboy.display();
54.   XPos += 1; // move every one pixel
55.   if (XPos > 127) // move sprite all the way across the screen
56.     XPos = 0; // reset back to start of screen
57.   }
```

Explanation (line 52)

Line 52: **Sprites::drawOverwrite(XPos, 20, InvaderMiddleGfx, currentFrame);.**
Sprites::drawOverwrite is a class used for drawing animated sprites to the OLED screen buffer, from program memory, contained in an array. **Sprites::drawOverwrite** has 4 arguments (parameters), X position on the screen, which we use **XPos** variable to enable the graphic to move across the screen. The next is Y position, which we have as 20 pixels. The parameter is the name of the sprite and the final parameter is the currentFrame.

Overview: Line 52 draws each frame of the graphic (sprite), in order, and in a different position as XPos is changing. When the frames are complete they return to frame 0 and repeat, the XPos reaches 127, it goes back to zero and then repeats. While this is happening we are writing text, in white on the screen as well, using code on line 35.

Final Code

```
1.  /* www.pixilart.com for sprite generation
2.    http://javl.github.io/image2cpp/ for image conversion
3.    This code displays Space Invaders and moves a sprite from left to right.
4.  */
5.  //libraries for graphics/display
6.  #include <Arduboy2.h>
7.  Arduboy2 arduboy;
8.
9.  constexpr uint8_t frameCount = 2;     // The number of frames in the image
10. constexpr uint8_t firstFrame = 0;     // The first frame index (always 0)
11. constexpr uint8_t lastFrame = (frameCount - 1);     // The last frame index (one less
    than the number of frames)
12.
13. uint8_t currentFrame = firstFrame;      // The current frame being drawn
14.
15. // Sprites
16. static const unsigned char PROGMEM InvaderMiddleGfx [] = {
17.   11, 8,                                 // width, height,pixel size of image
18.   0x1e, 0xb8, 0x7d, 0x36, 0x3c, 0x3c, 0x3c, 0x36, 0x7d, 0xb8, 0x1e,// Frame 1
19.
20.   0x78, 0x18, 0x7d, 0xb6, 0xbc, 0x3c, 0xbc, 0xb6, 0x7d, 0x18, 0x78,// Frame 2
21. };
22.
23. // move sprite left to right
24. int XPos = 0; // integer for the initial position of the sprite
25.
26. void setup() { // put your setup code here, to run once:
27.   arduboy.setFrameRate(1);
28.   arduboy.begin();
29. }
30.
31. void loop() { // put your main code here, to run repeatedly:
  arduboy.clear();
  arduboy.setTextColor(WHITE);
  arduboy.setCursor (20, 0);
  arduboy.print("Space Invaders");
if (!arduboy.nextFrame())           // Limit the frame rate (default 60fps)
    return;

  arduboy.pollButtons(); // Update the button state;

  if (arduboy.everyXFrames(1)) // If 15 frames have passed (at 60fps that's around 1/4 of a second)
  {
    if (currentFrame < lastFrame) // If the current frame is less than the last frame

    {
      ++currentFrame;       // Increment the current frame
    }
    else  // Otherwise
    {
      currentFrame = firstFrame;  // Cycle back to the start
    }
}
   Sprites::drawOverwrite(XPos, 20, InvaderMiddleGfx, currentFrame); // X, Y, sprite name. and sprite index
  arduboy.display();
  XPos += 1; // move every one pixel
  if (XPos > 127) // move sprite all the way across the screen
    XPos = 0; // reset back to start of screen
  }
```

Lesson Number: Game Development 1c

Lesson Title: Plotting Graphics and 2 Frame Movement + Sound
Code: Full Code for Lesson
System: Arduboy + Project ABE

Prerequisites to completing this tutorial

1. Game Development 1a,b

Step 1 ACCESS SOUND LIBRARY FILES

Type the following code into line 8-9

```
6.  #include <Arduboy2.h>
7.  Arduboy2 arduboy;
8.  #include <ArduboyPlaytune.h> // Arduboy sound library
9.  ArduboyPlaytune tunes(arduboy.audio.enabled);
10. bool sound_enabled=true;
```

Explanation (line 8-9)

Line 8: **#include arduboyPlaytune.h** is a header file which links to a library file which has to be downloaded and added to the IDE (unless you are using Project ABE, because the files are already built in). Like previously stated, this will give use access to additional code using simple object orientated commands to create sound for our game.
ArduboyPlaytune tunes(arduboy.audio.enabled) This function within the library enables the sound (unmutes).
More information: https://github.com/Arduboy/ArduboyPlaytune

Step 2 TUNES.TONE MAKING SOUND

Type the following code into line 48,54

```
44.   if (arduboy.everyXFrames(1)) // If 15 frames have passed (at 60fps that's around 1/4 of a second)
45.   {
46.     if (currentFrame < lastFrame) // If the current frame is less than the last frame
47.     {
48. tunes.tone(3300,5);// Frequency and duration (lowest 31hz highest 65535hz)
49.       ++currentFrame;        // Increment the current frame
50.     }
51.     else  // Otherwise
52.     {
53.       currentFrame = firstFrame;  // Cycle back to the start
54. tunes.tone(3000,5);// Frequency and duration (lowest 31hz highest 65535hz)
55.     }
56. }
```

Explanation (line 48, 54)

Tunes and tone are functions within the library which enable sound to be generated. tunes.tone(3300,5) has 2 parameters, first is the frequency of the sound and the second is the sound duration in milliseconds. As shown in the comments next to the code, frequency can go as high as 65535hz and as low as 31hz.

Final code

```
1.  /* www.pixilart.com for sprite generation
2.    http://javl.github.io/image2cpp/ for image conversion
3.    This code displays Space Invaders and moves a sprite from left to right.
```

```
*/
//libraries for graphics/display
#include <Arduboy2.h>
Arduboy2 arduboy;
#include <ArduboyPlaytune.h> // Arduboy sound library
ArduboyPlaytune tunes(arduboy.audio.enabled);
bool sound_enabled=true;

constexpr uint8_t frameCount = 2;      // The number of frames in the image
constexpr uint8_t firstFrame = 0;      // The first frame index (always 0)
constexpr uint8_t lastFrame = (frameCount - 1);      // The last frame index (one less than the number of frames)

uint8_t currentFrame = firstFrame;       // The current frame being drawn

// Sprites
static const unsigned char PROGMEM InvaderMiddleGfx [] = {
  11, 8,                                    // width, height,pixel size of image
  0x1e, 0xb8, 0x7d, 0x36, 0x3c, 0x3c, 0x3c, 0x36, 0x7d, 0xb8, 0x1e,// Frame 1

  0x78, 0x18, 0x7d, 0xb6, 0xbc, 0x3c, 0xbc, 0xb6, 0x7d, 0x18, 0x78,// Frame 2
};

// move sprite left to right
int XPos = 0; // integer for the initial position of the sprite

void setup() { // put your setup code here, to run once:
  arduboy.setFrameRate(1);
  arduboy.begin();
}

void loop() { // put your main code here, to run repeatedly:
  arduboy.clear();
  arduboy.setTextColor(WHITE);
  arduboy.setCursor (20, 0);
  arduboy.print("Space Invaders");
if (!arduboy.nextFrame())            // Limit the frame rate (default 60fps)
    return;

  arduboy.pollButtons(); // Update the button state;

  if (arduboy.everyXFrames(1)) // If 15 frames have passed (at 60fps that's around 1/4 of a second)
  {
    if (currentFrame < lastFrame) // If the current frame is less than the last frame

    {
tunes.tone(3300,5);// Frequency and duration (lowest 31hz highest 65535hz)
      ++currentFrame;       // Increment the current frame
    }
    else  // Otherwise
    {
      currentFrame = firstFrame;  // Cycle back to the start
tunes.tone(3000,5);// Frequency and duration (lowest 31hz highest 65535hz)
    }
}
   Sprites::drawOverwrite(XPos, 20, InvaderMiddleGfx, currentFrame); // X, Y, sprite name. and sprite index
  arduboy.display();
  XPos += 1; // move every one pixel
  if (XPos > 127) // move sprite all the way across the screen
    XPos = 0; // reset back to start of screen
  }
```