# Recap

In the last article we enabled the multi-level game play and the game is almost finished – all it needs is some polish to make it look a little professional. In this article, we will add some animation to the splash screen and some sound effects before packaging the application ready for publication.

# Splash Screen Animation

When I originally designed the artwork for the splash screen, I made individual sprites for each of the pipe pieces rather than one large graphic. This would allow me to ‘animate’ the pipes being laid.

Currently the drawSplash() routine clears the screen and renders all of the pipes in one action.

arduboy.clear();

sprites.drawOverwrite(112, 0, logo\_straight\_TB\_noflange, frame);

sprites.drawOverwrite(112, 16, logo\_elbow\_TL, frame);

sprites.drawOverwrite(96,  16, logo\_elbow\_TR, frame);

sprites.drawOverwrite(96,  0, logo\_elbow\_LB, frame);

...

arduboy.display();

The code below shows the modified version of the drawSplash() routine. It is similar to that original except it uses a helper function, splashAnimation(), to render the pipes individually. As you will see in a moment, the splashAnimation() routine takes the original parameters as the original sprites.drawOverwrite() function it replaces but adds an additional that allows the functionality to be skipped. After seeing the animation once or twice, most players will be happy to skip straight to the game play.

bool skipSplash = false;

arduboy.clear();

arduboy.display();

skipSplash = splashAnimation(112, 0, logo\_straight\_TB\_noflange, frame, skipSplash);

skipSplash = splashAnimation(112, 16, logo\_elbow\_TL, frame, skipSplash);

skipSplash = splashAnimation(96,  16, logo\_elbow\_TR, frame, skipSplash);

skipSplash = splashAnimation(96,  0, logo\_elbow\_LB, frame, skipSplash);

...

splashWaitForever();

The splashAnimation() function is shown below. As mentioned, it accepts a new parameter, skip, and if detected to be true causes the function to immediately return. Otherwise, the requested sprite is rendered to the screen and the routine loops 20 times, pausing for 15 milliseconds at the end of each loop. If the player has pressed the ‘A’ button, the function returns true and all subsequent calls to the function from the drawSplash() routine effectively skip the animation.

#define ANIMATION\_DELAY\_SHORT     20

bool splashAnimation(byte x, byte y, const uint8\_t \*bitmap, uint8\_t frame, bool skip) {

  if (skip) return true;

  int i = ANIMATION\_DELAY\_SHORT;

  sprites.drawOverwrite(x, y, bitmap, frame);

  arduboy.display();

  while (i >= 0) {

  arduboy.pollButtons();

    if (arduboy.justPressed(A\_BUTTON)) { return true; }

    arduboy.delayShort(15);

    i--;

  }

  return false;

}

We will revisit these two functions in a moment to add some sound effects and additional functionality that will allow the player to turn the sound effects on or off.

# Adding Sound Effects

The Arduboy has a number of sound options including the basic tone() function of the Arduboy2 library and the more functional ArduboyTones library maintained by @MLXXXp. The ArduboyTones library provides methods to play single notes or sequences that can be stored in PROGMEM. It also plays these tunes asynchronously, allowing you code to continue executing.

The Arduboy2 library also contains code that other libraries, such as ArduboyTones, can utilize to save and retrieve audio settings providing a consistent model across the different libraries. The sample code below shows how the ArduboyTones library and the standard Arduboy2 libraries combine.

#include <ArduboyTones.h>

ArduboyTones sound(arduboy.audio.enabled);

sound.tone(NOTE\_C1,50, NOTE\_D1,50, NOTE\_C1,50);

After importing the library, an instance of the sound class is instantiated. The constructor of the ArduboyTones class has a parameter to indicate whether sounds should be played (true) or not (false). Here we have used a function in the base Arduboy2 class to retrieve the audio setting from the EEPROM setting while instantiating the class. Other functions of the Arduboy2 class allow the setting to be changed and saved to the EEPROM allowing a player to mute the game and have that setting honored when they next turn the device on.

Callout to “Constructors”

The tone() method comes in three variants, as shown below, that play one, two and three notes respectively.

static void tone(uint16\_t freq, uint16\_t dur = 0);

static void tone(uint16\_t freq1, uint16\_t dur1,

uint16\_t freq2, uint16\_t dur2);

static void tone(uint16\_t freq1, uint16\_t dur1,

uint16\_t freq2, uint16\_t dur2,

uint16\_t freq3, uint16\_t dur3);

Callout to “#defines versus enumerations”

The function takes ‘pairs’ of parameters which represent the frequency of the note followed by a duration. The notes are enumerated as #defines in the class itself and span nine octaves, ranging from a very low C to a very high B. The naming convention for these is NOTE\_{tone, A - G}{octave, 0 – 8}. The duration is specified in 1/1024th of a second – close enough to milliseconds. Omitting the duration or specifying a duration of 0 will result in the tone being played forever.

#define NOTE\_C0  16

#define NOTE\_CS0 17

#define NOTE\_D0  18

#define NOTE\_DS0 19

…

#define NOTE\_A9  14080

#define NOTE\_AS9 14917

#define NOTE\_B9  15804

The library also defines a second set of notes that are the equivalent to the previous ones but they are played at a higher volume. These have the same naming convention as the normal notes but are suffixed with an ‘H’.

#define NOTE\_C0H  (NOTE\_C0 + TONE\_HIGH\_VOLUME)

#define NOTE\_CS0H (NOTE\_CS0 + TONE\_HIGH\_VOLUME)

#define NOTE\_D0H  (NOTE\_D08 + TONE\_HIGH\_VOLUME)

#define NOTE\_DS0H (NOTE\_DS0 + TONE\_HIGH\_VOLUME)

…

#define NOTE\_A9H  (NOTE\_A9 + TONE\_HIGH\_VOLUME)

#define NOTE\_AS9H (NOTE\_AS9 + TONE\_HIGH\_VOLUME)

#define NOTE\_B9H  (NOTE\_B9 + TONE\_HIGH\_VOLUME)

The three tone() variants are great for short sound effects and I have created four small functions that make appropriate sounds while pipes are being laid in the splash screen or the play selects and matches nodes while playing the game. I will leave it to you to search for these within the game play code itself.

void playSplashTune() { sound.tone(NOTE\_C1,50, NOTE\_D1,50, NOTE\_C1,50); }

void playClearSelectionTune() { sound.tone(NOTE\_C2,50, NOTE\_D2,50, NOTE\_E2,50); }

void playSelectNodeTune() { sound.tone(NOTE\_C4, 50); }

void playMatchNodeTune() { sound.tone(NOTE\_C4,50, NOTE\_D4,50, NOTE\_E4,50); }

The ArduboyTones library also allows sequences of tones to be stored in an array in PROGMEM in much the same way sprites are defined. This caters for longer sequences than the tone() method can handle. An example of the declaration and use is shown below. Two things to note are the use of the TONES\_END constant is used to signify the end of the sequence and the use of the tones() - with an ‘s’ - method.

const uint16\_t puzzleSolved[] PROGMEM = {

  NOTE\_C4, 50, NOTE\_D4, 50, NOTE\_E4, 50,

   NOTE\_C4, 50, NOTE\_D4, 50, NOTE\_E4, 50,

   NOTE\_C4, 50, NOTE\_D4, 50, NOTE\_E4, 50,

   TONES\_END

};

void playPuzzleSolved() {

  sound.tones(puzzleSolved);

}

# Toggling Sound on and Off

I think it is a good design practice for all games to allow players to mute the sound as soon as the game starts up and to have that setting be honoured next time the Arduboy is turned on. This prevents those awkward situations where you are at, say, a funeral and decide to play a quick game while the eulogy rambles on and on. The high pitched sounds of the Arduboy can be hard to muffle as you desperately try to turn the machine off.

Obviously the authors of the Arduboy2 library thought so as well and they have included a standardized way to save and retrieve sound settings that can be utilised across games. The audio state – on or off – is saved in the EEPROM at position 2 and its value can be retrieved using the arduboy.audio.enabled() function.

When rendering the splash screen, I have included a sprite in the bottom right corner to indicate the current sound status. The modified drawSplash() function below shows the code in action.

void drawSplash() {

arduboy.clear();

sprites.drawOverwrite(120, 56, (arduboy.audio.enabled() ? sound\_icon : no\_sound\_icon), frame);

arduboy.display();

…

}

The splashAnimation() function has also been updated to detect if the user has pressed the ‘B’ button. If pressed, the value in the EEPROM is overwritten using a wrapper function that coordinates the calls to the base Arduboy2 functions.

bool splashAnimation(byte x, byte y, const uint8\_t \*bitmap, uint8\_t frame, bool skip) {

  ...

  if (arduboy.justPressed(B\_BUTTON)) {

    toggleSoundSettings();

    sprites.drawOverwrite(120, 56, (arduboy.audio.enabled() ? sound\_icon : no\_sound\_icon), frame);

    arduboy.display();

  }

  ...

}

void toggleSoundSettings() {

  if (arduboy.audio.enabled()) {

    arduboy.audio.off();

    arduboy.audio.saveOnOff();

  }

  else {

    arduboy.audio.on();

    arduboy.audio.saveOnOff();

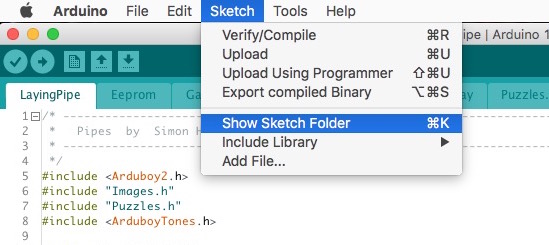
  }

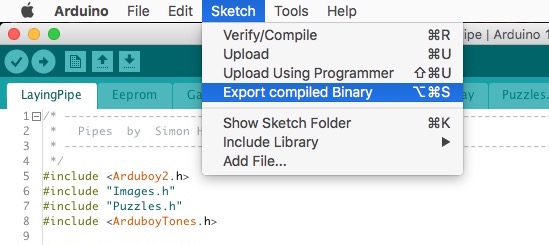
}

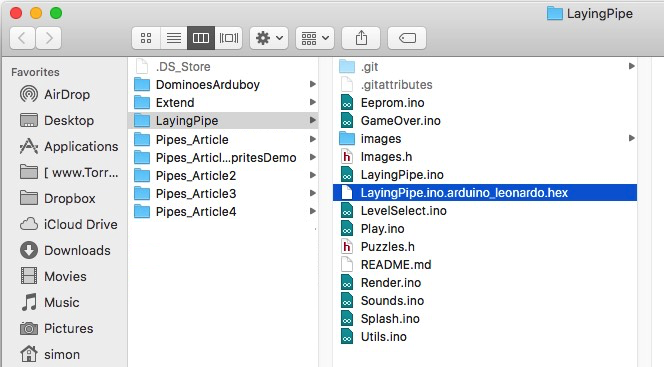
At this point, our application has a funky splash screen, sound effects and an ability for the player to turn the sounds on or off. The complete code is included in my repository at <https://github.com/filmote/LayingPipe> and I encourage you to download it and look at some of the other little additions I have made to make the level and puzzle selection and game over banners nicer. This complete version has 30 puzzles per level for you to try out.

# Packaging a game for Distribution

Creating a Hex File







Creating an Arduboy file.

An .arduboy file is an archive file (zip file) that contains the HEX file created in the previous step along with a metadata file that describes the program in a standard format. Optionally, the archive can contain a banner image and one or more screenshots of the game in action. The .arduboy format was developed by the community to allow game loaders, such as @crait’s *Arduboy Manager*, TeamARG’s *Game Loader* or @ereid’s *Arduboy Uploader*, to be able to catalogue and display a library of games in a nice format.

<http://www.team-arg.org/AGL-technical.html>

Using [Jeremy Dorn4](http://jeremydorn.com/) his JSON editor, we are providing a form where you can fill out all fields and it will create a correct json schema, you then can copy paste in your own info.json file. Because of the nature of the link, I had to shrink the link with TinyURL.

[http://tinyurl.com/jkw9vpf24](http://tinyurl.com/jkw9vpf)

{

  "schemaVersion": 2,

  "title" : "Pipes",

  "description" : "The classic pipe laying game for the Arduboy.",

  "version" : "1.0.0",

  "device" : "Arduboy",

  "author" : "Filmote",

  "url" : "<https://github.com/filmote/LayingPipe>",

  "sourceUrl" : "<https://github.com/filmote/LayingPipe>",

  "genre" : "Puzzle",

  "date" : "2017-06-28",

  "idea" : "Filmote",

  "code" : "Filmote",

  "art" : "Filmote",

  "sound" : "Filmote",

  "publisher" : "Filmote",

  "banner": "banner.png",

  "screenshots": [

    { "title": "Title Screen",    "filename": "screenshot00.png" },

    { "title": "Level Select 1",  "filename": "screenshot01.png" },

    { "title": "Level Select 2",  "filename": "screenshot02.png" },

    { "title": "Simple Game",     "filename": "screenshot03.png" },

    { "title": "Complex Game 1",  "filename": "screenshot04.png" },

    { "title": "Complex Game 2",  "filename": "screenshot05.png" }

  ],

  "buttons": [

    { "control": "Down",    "action": "Down" },

    { "control": "Up",      "action": "Up" },

    { "control": "Left",    "action": "Left" },

    { "control": "Right",   "action": "Right" },

    { "control": "A",       "action": "Select" },

    { "control": "B",       "action": "Back" }

  ]

}

You can use zip command in Terminal to zip the files without the *.DS\_Store*, *\_\_MACOSX* and other .\* files.

Open Terminal (search for terminal in spotlight)

Navigate to the folder you want to zip using the cd command

Paste this zip -r dir.zip . -x ".\*" -x "\_\_MACOSX"

Example: Let's say you have a folder on your desktop called Folder with stuff to zip.

Open terminal and write following commands:

cd Desktop/Folder

zip -r dir.zip . -x ".\*" -x "\_\_MACOSX"

Now you have a file called dir.zip without *\_\_MACOSX* and .\* files in the folder Folder on your desktop.