# 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

Once your game is complete and testing finished, you will probably want to share the code with the world. There are three ways to do this and I suggest you do them all!

Publishing the code to GitHub

Callout to “What is GitHub?”

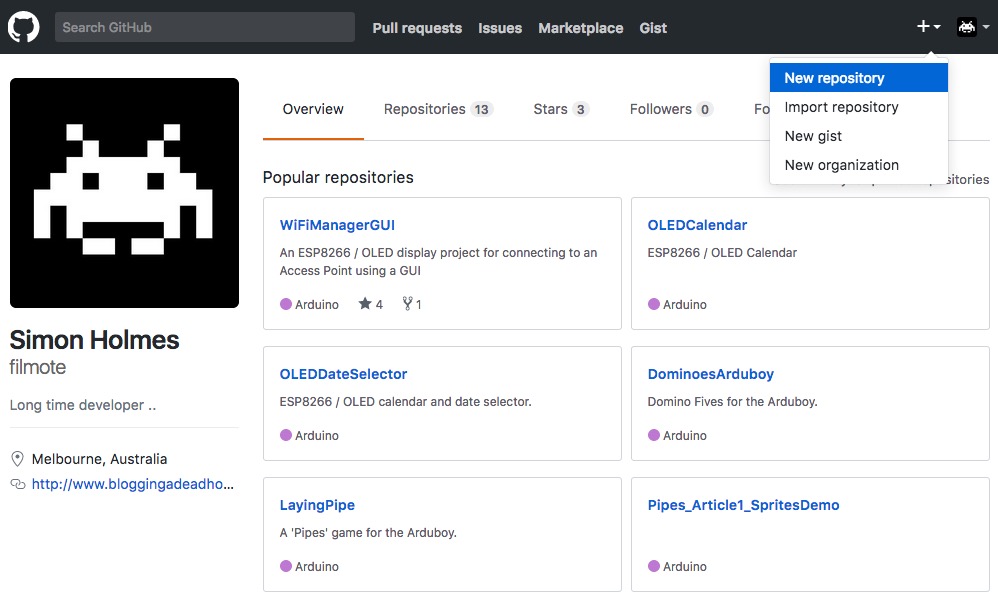
The Arduboy is an open-source project and is supported by a community who will offer assistance and guidance free of charge. As member of the community, you can contribute by publishing your source code for others to learn from.

GitHub is an open-source, web-based version control system that will allow people to view and comment on your code and – if you allow them to – suggest changes that you can incorporate back into the code set. Publishing an application is simple and I have documented it using the web interface only. Once you are familiar with the concepts, I would recommend you download the GitHub Desktop application. Among other things, it compares the files on your desktop against the repository to determine which ones to update thus minimizing the chance that a change will go unpublished or an incomplete change that spans multiple files is published.

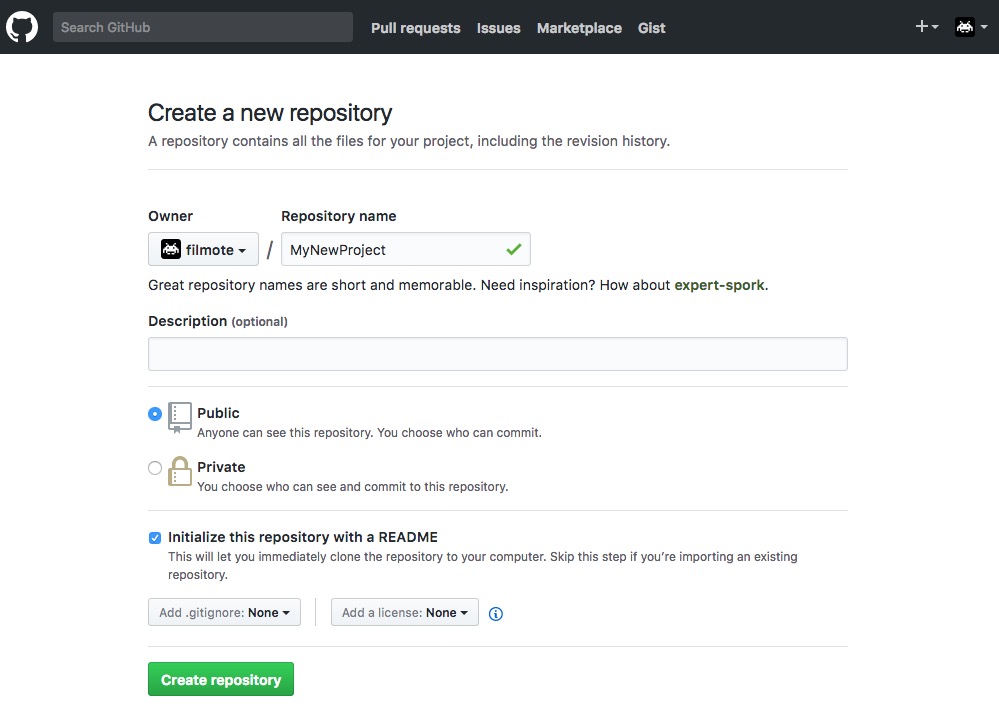
To publish an application:

**Step 1**: Create a user profile if you do not have one. Visit <https://github.com/> to get started.

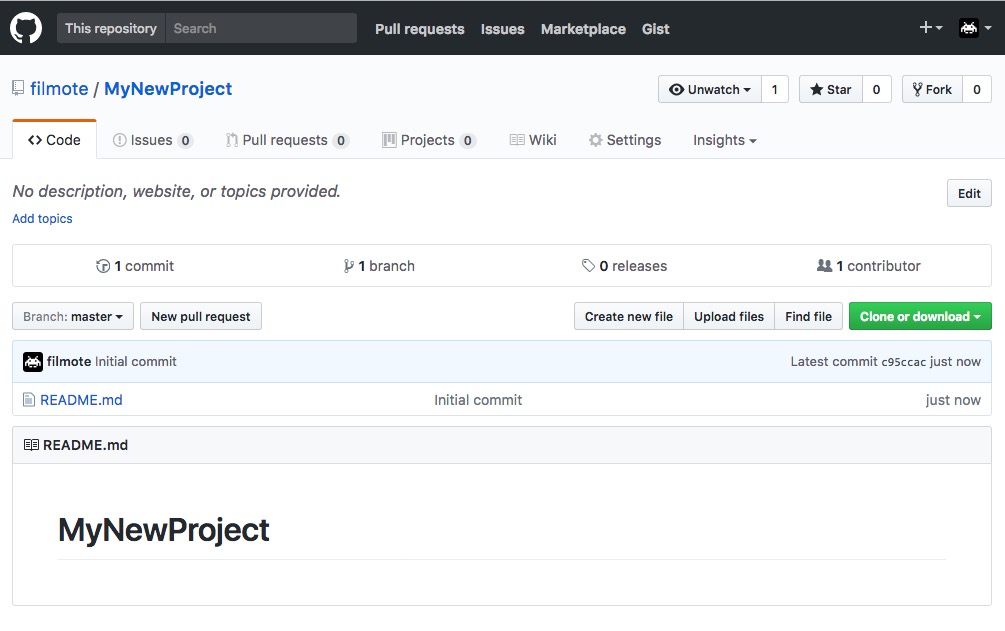
**Step 2**: Log in to GitHub using the credentials created in Step 1. Select *New Repository* from the menu in the top right hand corner of the screen, as shown below.



**Step 3**: Give the repository a name and, optionally, a description. Select the ‘Public’ repository option and check the *Initialize this repository with a README* checkbox. Click the *Create Repository* to complete the creation of the repository.

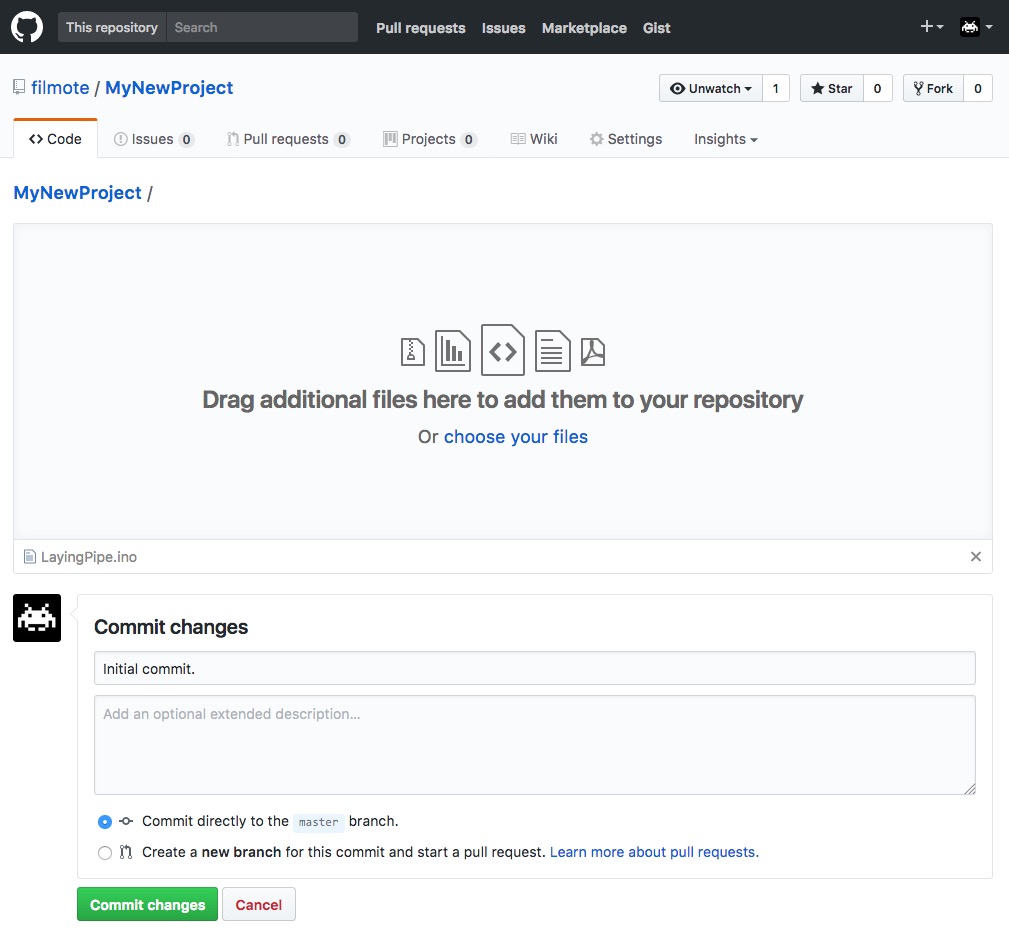


**Step 4**: Give the repository a name and, optionally, a description. Select the ‘Public’ repository option and check the *Initialize this repository with a README* checkbox. Click the *Create Repository* to complete the creation of the repository.

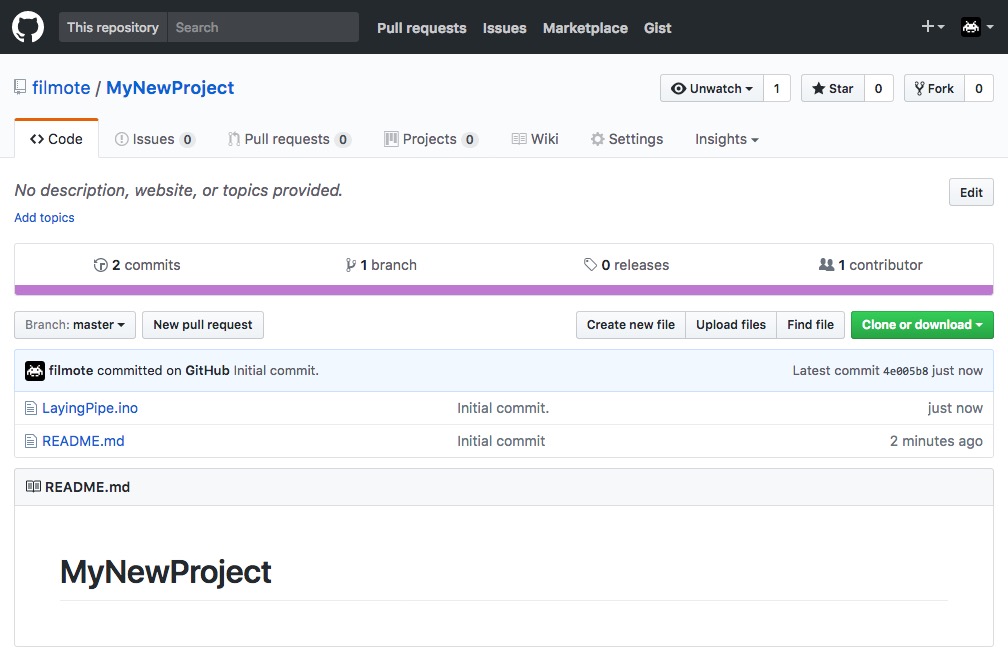


**Step 5**: Once the repository has been created, the actual source files can be added to the project. From the repository view, click *Upload Files* button to begin adding files.

Files can be dragged onto the window from Windows Explorer or OS X’s Finder. The image below shows how the files are accumulated under the drag-and-drop window as they are added. Provide a description of the files being committed and click the *Commit Changes* button.



**Step 6**: The repository is shown again with the files added. You can add new files as needed or update the existing ones using the *Upload Files* button. Other people can retrieve a copy of the code using the *Clone or Download* option.

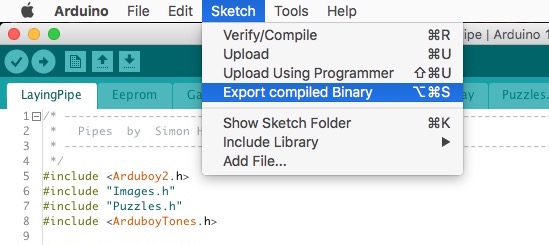


# Creating a Hex File

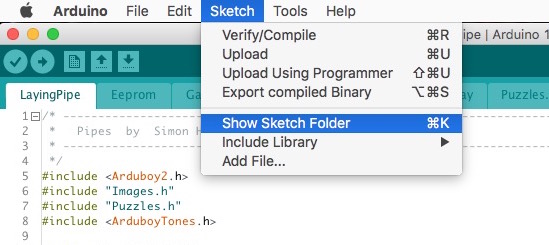
A .hex file is a compiled, binary version of your program that some game uploaders, such as @crait’s *Arduboy Manager,* @shdwwzrd’s *Arduboy Center* or @ereid’s *Arduboy Uploader* can directly load onto an Arduboy. A compiled binary saves the user the hassle of downloading your code and any prerequisite libraries and having to compile the code and upload it themselves.

Creating a binary file is simple and can be achieved following these steps.

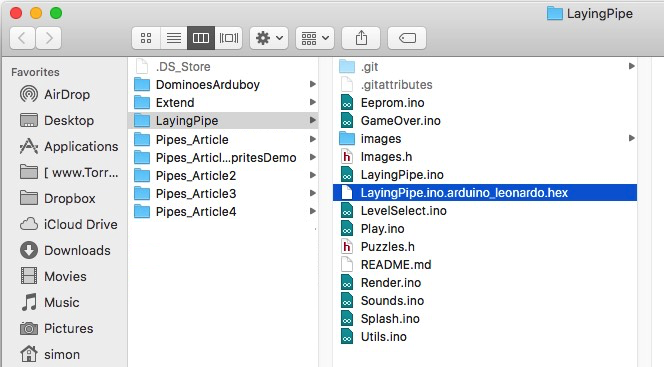
**Step 1**: Once the program is complete and compiles properly, create the .hex file by selecting the *Export compiled Binary* option from the *Sketch* menu as shown below. The images are from a Macintosh but the menu options are identical on the Windows version of the Arduino IDE.



**Step 2**: Once complete, select the *Show Sketch Folder* option from the *Sketch* menu. This will open Windows Explorer or OS X’s Finder at the folder in which the new binary file was created.



**Step 3**: The file is named after the primary INO file in your application and the platform or board selected when compiled. Rename this file and shorten it to *{your game name}.hex*, e.g. in the example below the file might get renamed to *Pipes.hex*.



This file can now be shared around to the various game sites.

# 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.

An .arduboy archive at a minimum must contain the .hex file along with a metadata file named *info.json*. The format of the info.json file can be found at the Team ARG site, here <http://www.team-arg.org/AGL-technical.html>

The file can be created by hand or you can use Jeremy Dorn’s web-based JSON editor which allows you to build one simply by filling in a form. The tool can be accessed via the TinyURL [http://tinyurl.com/jkw9vpf24](http://tinyurl.com/jkw9vpf)

Amongst the options you can fill in are two important ones for promoting your application. The first is a banner which is a 700 x 192 pixel, black and white image in PNG, BMP, JPG or GIF format. Team ARG have some nice tools for converting colour images to 1-bit black and white ones at <https://teamarg.github.io/arduboy-image-converter/>. I have made banner logos by finding appropriate artwork on the web and cropping it down to the correct size and adding a heading using the text tools in Paint / Paintshop. Once complete, I have then used Team ARG’s tool to convert it into the correct, 1 bit format.

The second option is for screenshots showing you game in action. These are again 1 bit, black and white images but are 128 x 64 pixels in size. You can use @crait’s *Arduboy Manager* to capture screenshots following the instructions found here <https://community.arduboy.com/t/screen-mirroring-guide-how-to-screenshot-your-game/2800>. In the latest version of the *info.json* schema, the screenshots can be named anything but by convention and as a prerequisite from earlier schema versions, they are named *screenshot00* and sequentially numbered. As with the banner, these images can be PNG, BMP, JPG or GIF format.

Once the JSON form is complete, the output of the file can be seen on the right-hand side of the screen. Once the form is completed, copy and paste the output into a text file named *info.json*. A sample *info.json* file can be seen below:

{

  "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",

  "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" }

  ]

}

The actual *.arduboy* file can be created by adding the *info.json* file, the .hex file and all of the image files into a single zip or compressed folder. Once done, rename your file to {your game name}.arduboy.

The Window’s command ‘Send to Compressed Folder’ will do the job perfectly. However, if you are using OS X’s ‘Compress’ option from Finder you will run into trouble as it includes a *.ds\_store* and *\_\_MACOSX* file automatically in the archive and these are unknown to the game loaders.

If you are using a Macintosh, follow the steps below to create a proper *.arduboy* file.

**Step 1**: Copy the *info.json* file, the .hex file and all of the image files into a new folder on the desktop or under the documents directory.

**Step 2**: Open the *Terminal* application. It is usually hidden in the *Utilities* folder under the *Applications* folder in the dock. If you cannot find it, search for *Terminal* using Spotlight.

**Step 3**: Navigate to the directory using the cd command. The file structure is hierarchical and you will probably start in a directory called /Users/{your user name}. You can confirm this by using the command pwd.

If your folder new folder is on the desktop, you can probably navigate to it using the command

cd Desktop/NewFolderName. If you created your new folder in the Documents directory, you can probably navigate to it using the command cd Documents/NewFolderName.

**Step 4**: Create a zip file using the command zip -r dir.zip . -x ".\*" -x "\_\_MACOSX" Note that the name of the file is specified as dir.zip but this can be changed to any appropriate name. Also note that there are two leading underscores on \_\_MACOSX.