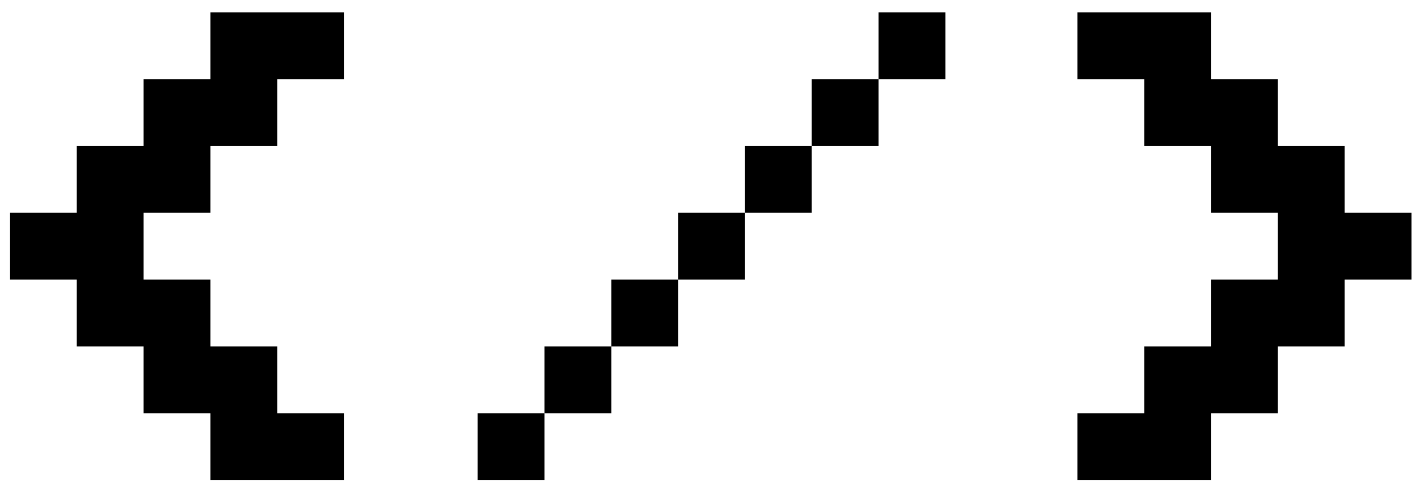


PicoBoy SDK Docs



Created by HalloSpaceBoy

[Discord](#)

[GitHub](#)

Table of Contents

1. Creating a Game

- The File Structure
- The IDE
- The Home Screen
- The Code
- Music
- Images and Sprites
- PLibraries
- A Step-by-Step Guide

1. Thonny

- Introduction
- Setup
- Connect Your PicoBoy
- Interacting With Your PicoBoy
- Managing Files on Your PicoBoy
- Running Code on Your PicoBoy
- Helpful Tips

Table of Contents

3. PicoBoySDK Toolkit

- Introduction
- The Functions of the Program

4. The Functions of the SDK

- The PicoBoySDK Object
- The PlayerObject Object
- The MusicBox Object

5. The Limitations of the Hardware




6. Final Notes

Creating a Game

The Project File Structure

The file structure of a PicoBoySDK game consists of the main game code file, the title image, and any dependencies the game needs such as image, sprite and music files within a folder of the same name as the main code file (The maximum length of your game's title is 18 characters). This folder is known as the project folder. There can only be one code file in the project folder. If there are more code files in the project folder, PicoBoy Communication Software will get confused while adding the game. Any image/sprite files must have the file extension .pbimg, .sprt, or .bin because of the way PicoBoy Communication Software handles loading files. The main code file must have the extension .py and the name of the main code file must be the same name as the project folder.

Here is an example of the file structure of a PicoBoySDK project folder:

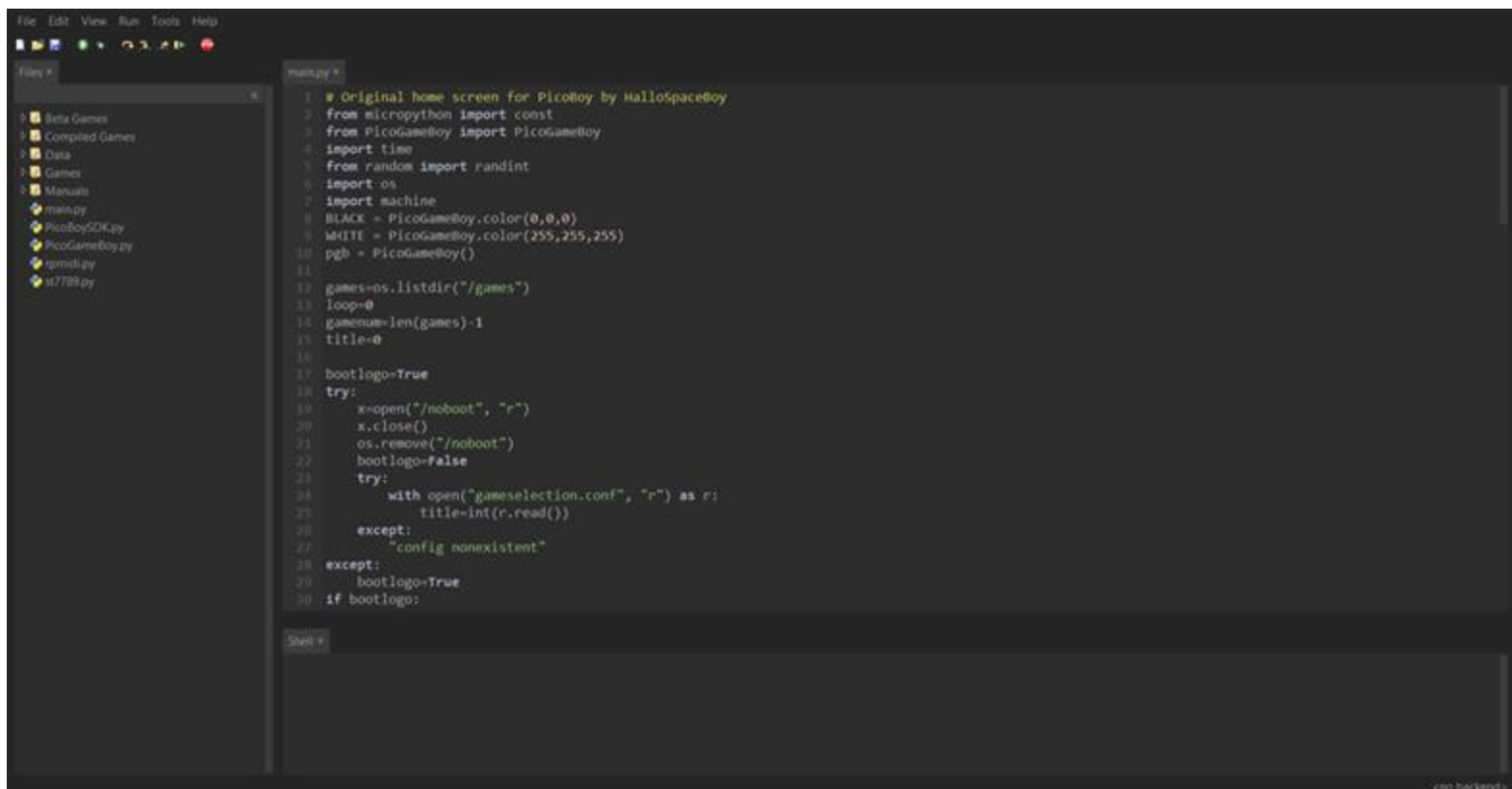
Minefield SDK Demo				
Search Minefield SDK Demo				
Name	Date modified	Type	Size	
 Minefield SDK Demo (Title Image).pbimg	10/1/2023 12:21 AM	PBIMG File	113 KB	
 Minefield SDK Demo.py	10/26/2023 8:24 PM	Python Source File	7 KB	
 sprite.sprt	9/30/2023 11:13 PM	SPRT File	1 KB	

Creating a Game

The IDE

To develop for the PicoBoy, the best IDE to use is [Thonny](#). Thonny is a basic IDE for Python development and is best for developing on the PicoBoy. It allows you to easily connect and modify the PicoBoy by directly accessing it's filesystem. Thonny also gives you a debugging console as well as the Python console built into the PicoBoy. It is very easy to use and is best not only for writing PicoBoy games, but for general Python development in addition.

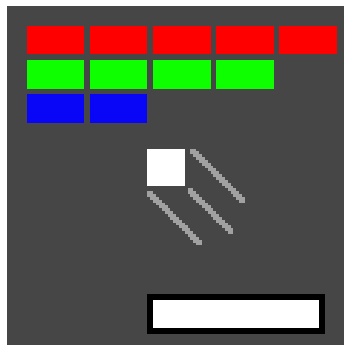
Here is an example of the Thonny IDE:



Creating a Game

The Home Screen

When the PicoBoy boots, it displays a home screen that can be used for game selection. For every game on the PicoBoy, it will display the title image contained in that game's files. If that title image does not exist, it will display the text "No Image". The image can be anything as long as its size is 120x120 pixels, no more, no less. The PicoBoy looks for the file with the name `projectfoldername+ " (Title Image).pbimg"`. There must be a space between the name of the folder and "(Title Image)" and capitalization matters. If you have a game with the name "Breakout", the title image would be called "Breakout (Title Image).pbimg". To create a usable image, refer to the section describing how to create and compile images. Use the "Compile Image" option, do not use the "Compile Sprite" option. Note that the create project function in PicoBoy SDK Toolkit will automatically convert and format your title image.



Example of the Breakout
title image

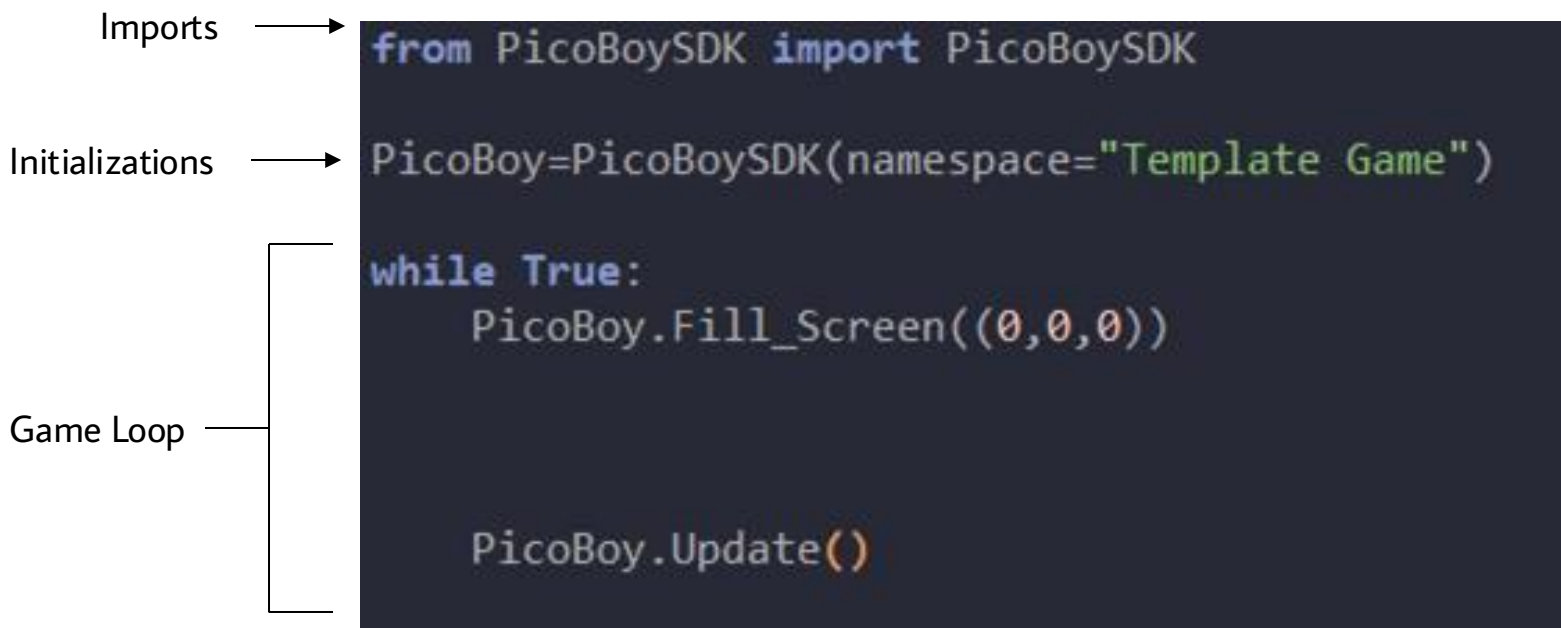


Example of title image naming scheme

Creating a Game

The Code

A game consists of multiple sections. These sections include imports, initializations, and the mainloop. In the imports, you import any libraries you may need in your game. In the initializations, you initialize any classes, sprites, etc for use in the game. In the mainloop, functions that happen every frame are called. For a template game that has all of the boilerplate code written, a template game is included in the SDK zip file. Note that the namespace initialization is automatically handled if you create your project with PicoBoy SDK Toolkit

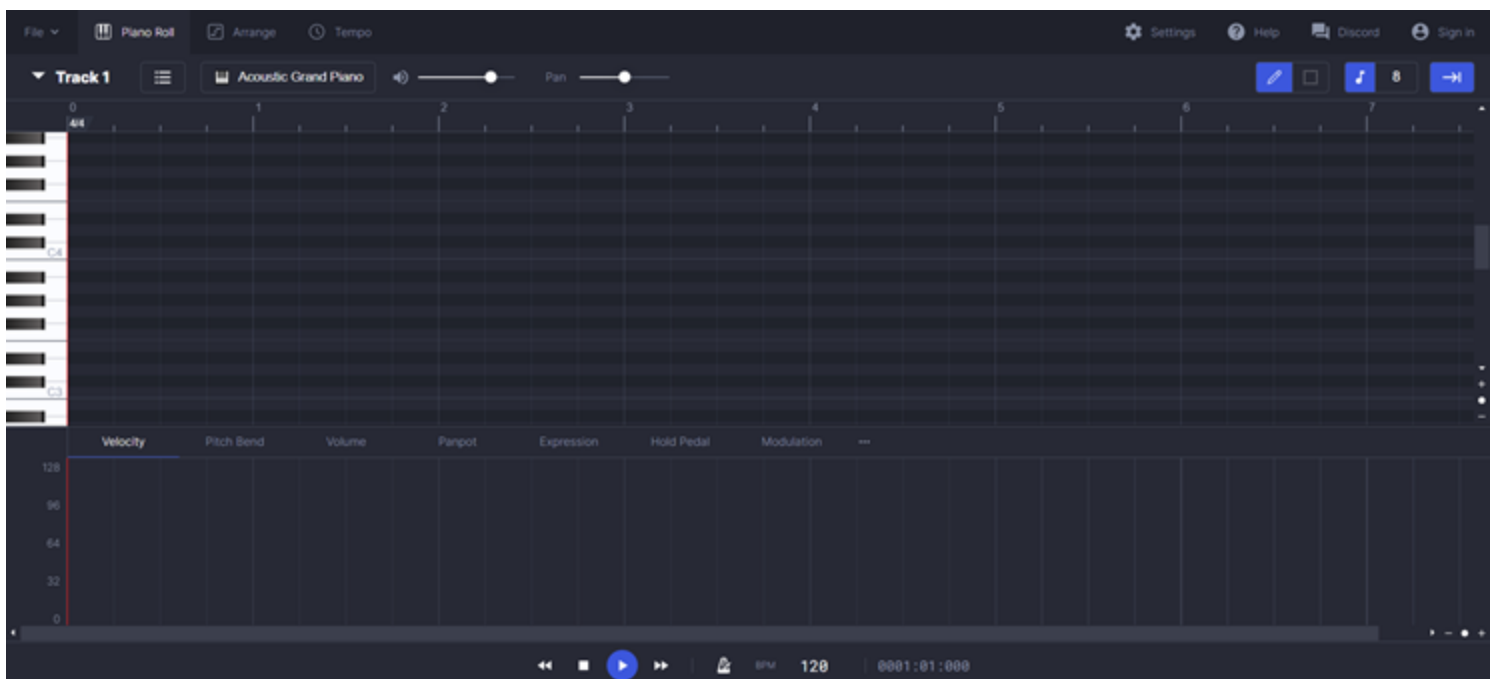


An example of the code structure from the Template Game

Creating a Game Music

Note: When making your music, make sure that the notes that are directly next to each other have space between them. The music compiler may encounter issues while compiling music if notes are directly next to each other.

In your game, you can play music using the MusicBox object. This object uses the function `Play_Song()` to play a song, this function uses a .pbs file. This kind of file contains the music data the PicoBoy uses to play a song. The way you can make your own .pbs file is by making a midi track in a midi editor, then compiling it to a .pbs file. A recommended editor is [Signal](#), it is completely free and online. Keep in mind while making your midi track, the PicoBoy can only play three notes at the same time. Any more will result in a track that will skip notes. When you are finished making your midi file, download it and open up the PicoBoySDK Toolkit. Compile the song and move the .pbs file into your project folder. You can now play the song with the MusicBox object.

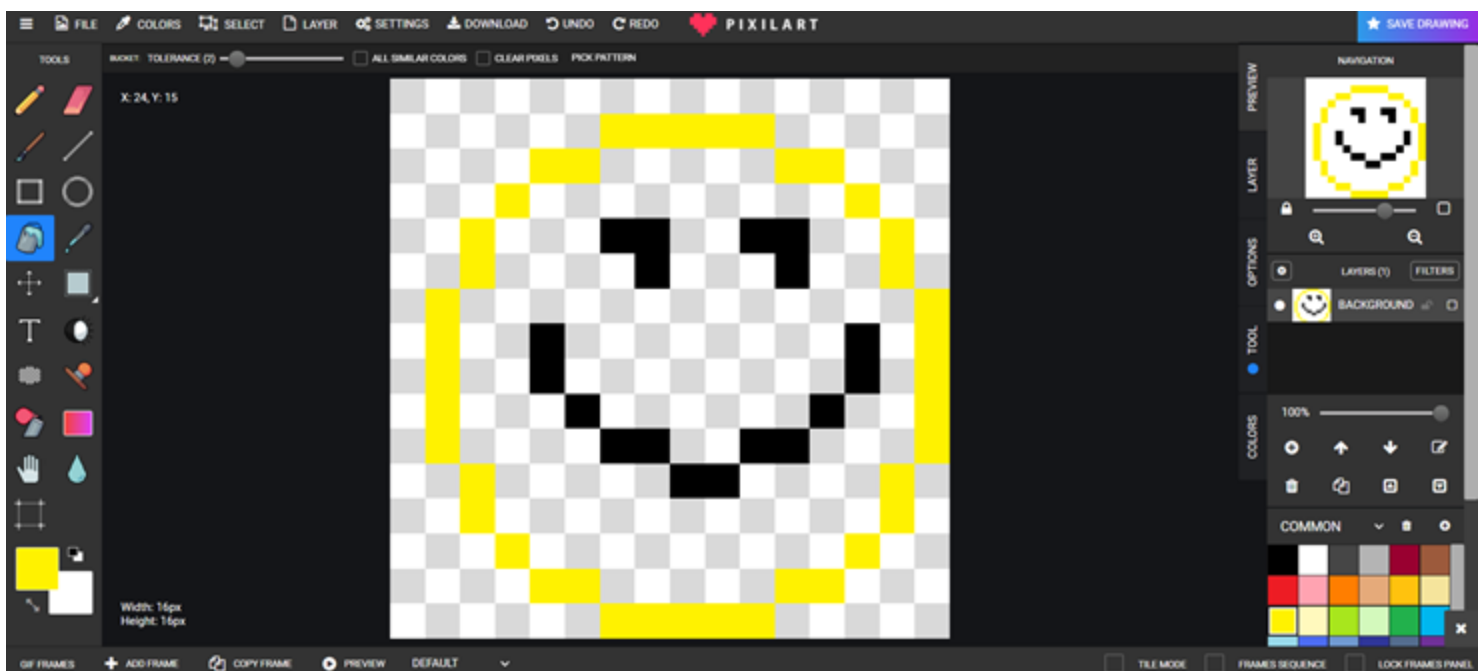


An image of the Signal editor

Creating a Game

Images and Sprites

When you are making a game, you can use images and sprites to have a graphically complex game. The way you can make these images and sprites is by using a pixel art editor. A recommended editor for making sprites and images is [Pixilart](#), it is completely free and online. In your pixel art editor choose the size of pixel art that you will make. If you are making an image, the size can be 1x1 pixels to 240x240 pixels. If you are making a sprite, the size of it can be from 1x1 pixels to 240x240 pixels. When you are finished with your art, export it as a PNG file. Open the PicoBoySDK Toolkit and choose either Compile Image or Compile Sprite depending on what you are making. You will get either a .sprt file or a .pbimg file. You can now put your image/sprite in your project folder and use it in your game.



An image of the Pixilart editor

Creating a Game

PLibraries

When you are making a game, sometimes the base functions and objects in the PicoBoySDK aren't enough to do what you want to do. This desire for custom features is remedied with the usage of PLibraries (PicoBoy Code Libraries), which allow you to import custom code from libraries written by you or other developers, allowing you to customize your development experience. PLibraries have a very specific structure that they need to abide by in order for the PicoBoySDK to detect and utilize them. They need to include only one Python file, containing objects that can be instantiated. This means that the files can only have declarations of objects to be utilized in another script as opposed to procedurally run code. All files included in the PLibrary must be contained in a folder of the same name of the Python file. Once imported, the objects from a PLibrary can be utilized just as an object declared in the main game code file. PLibraries can also include any other required files for the library to run such as text, image, sprite, music, and other file types. These files must be referenced from the path `"/libs/libraryname/filename"`, as the PicoBoySDK is designed to run from its namespace folder and does not have direct access to the files from within it. Using non-relative paths is paramount to avoid bugs.

```
from PicoBoySDK import PicoBoySDK

PicoBoy=PicoBoySDK(namespace="Template Game")
PicoBoy.Load_Library("Test_Plib", ["TestObj"])
test_obj=TestObj("This is a test object")
while True:
    PicoBoy.Fill_Screen((0,0,0))
    #Put the code you want here, anything graphical must come before PicoBoy.Update()
    test_obj.print()
    PicoBoy.Update()
```

Import the TestObj object from the Test_Plib PLibrary

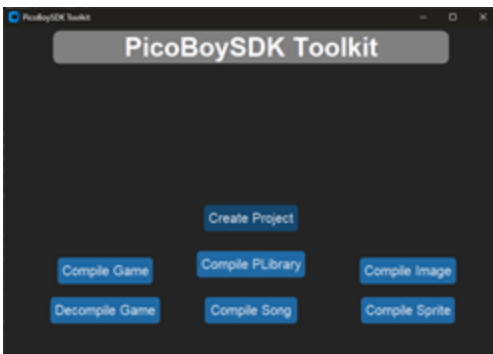
Initialize the TestObj object as if it were built in

Creating a Game

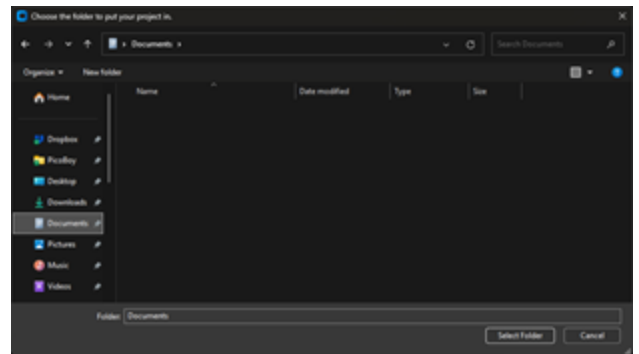
Starting Your Project

To begin your journey to making a game, you need a starting point. Luckily, PicoBoySDK Toolkit provides a good one to use. To start your project, you need a project folder, containing all of your code, images, sprites, music, and more. PicoBoySDK Toolkit provides an easy way to initialize this project folder, allowing you to choose a name, location, and even title image. The following demonstrates the creation of a project:

1. Click on the "Create Project" button.



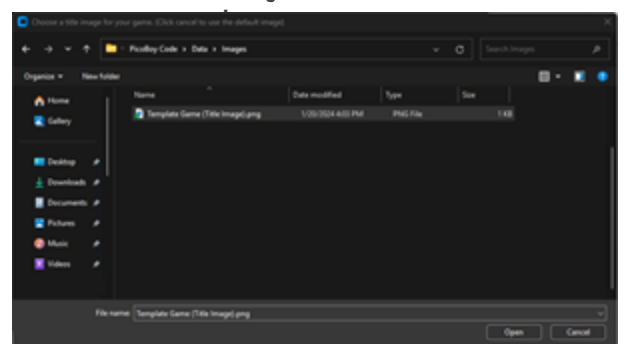
2. Choose the location of your project on your computer.



3. Choose the name of your project. Note that your name cannot contain any illegal characters and must be 18 characters or less.



4. Choose the title image of your project. If you don't choose an image, a default one will be used. You can choose PNG, JPG, or PBIMG images, they will be automatically formatted and



Creating a Game

A Step-by-Step Guide

1. Open Pixilart, create a 120x120 canvas, and draw a title image. This will be the image shown for your game on the home screen of your PicoBoy. Once finished, download your image.
2. Open PicoBoySDK Toolkit. (You can download it from the [GitHub page](#).)
3. Click on "Create Project". When prompted, choose the folder you want to create your project in.
4. After that, choose the name of your project
5. When prompted, find the image you created earlier and select it. It can be a PNG, JPG, or PBIMG image
6. You can now find your project folder for your game in the folder you chose earlier.
7. In the project folder, you will find your title image and the main game code file.
8. Open the game code file, and write your code. It should already be formatted with the name of your project.
9. After writing your game's code, you can compile your game.
10. Add your game to your PicoBoy using PicoBoy Communication Software

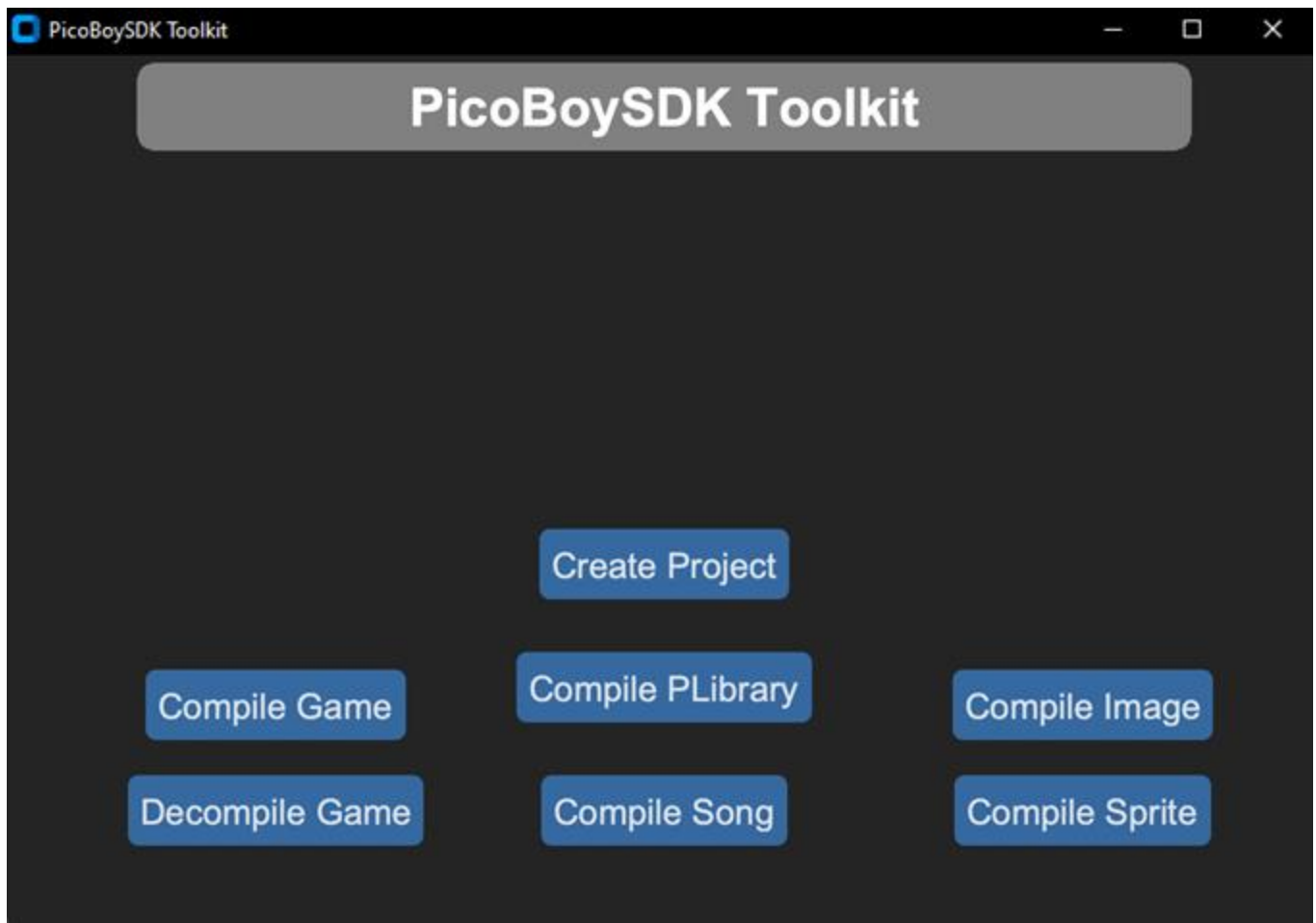
Note: If you want to use any of Thonny's filesystem and game running features, you need to compile and install your game onto your PicoBoy, no matter it's point of completion.

PicoBoySDK Toolkit

Introduction

The PicoBoySDK Toolkit is a program built to make making games for the PicoBoy easy. It has numerous functions that assist in game development such as image compilation, sprite compilation, music compilation, game compilation, and game decompilation. These different tools come in very handy when making games for the PicoBoy with the PicoBoySDK.

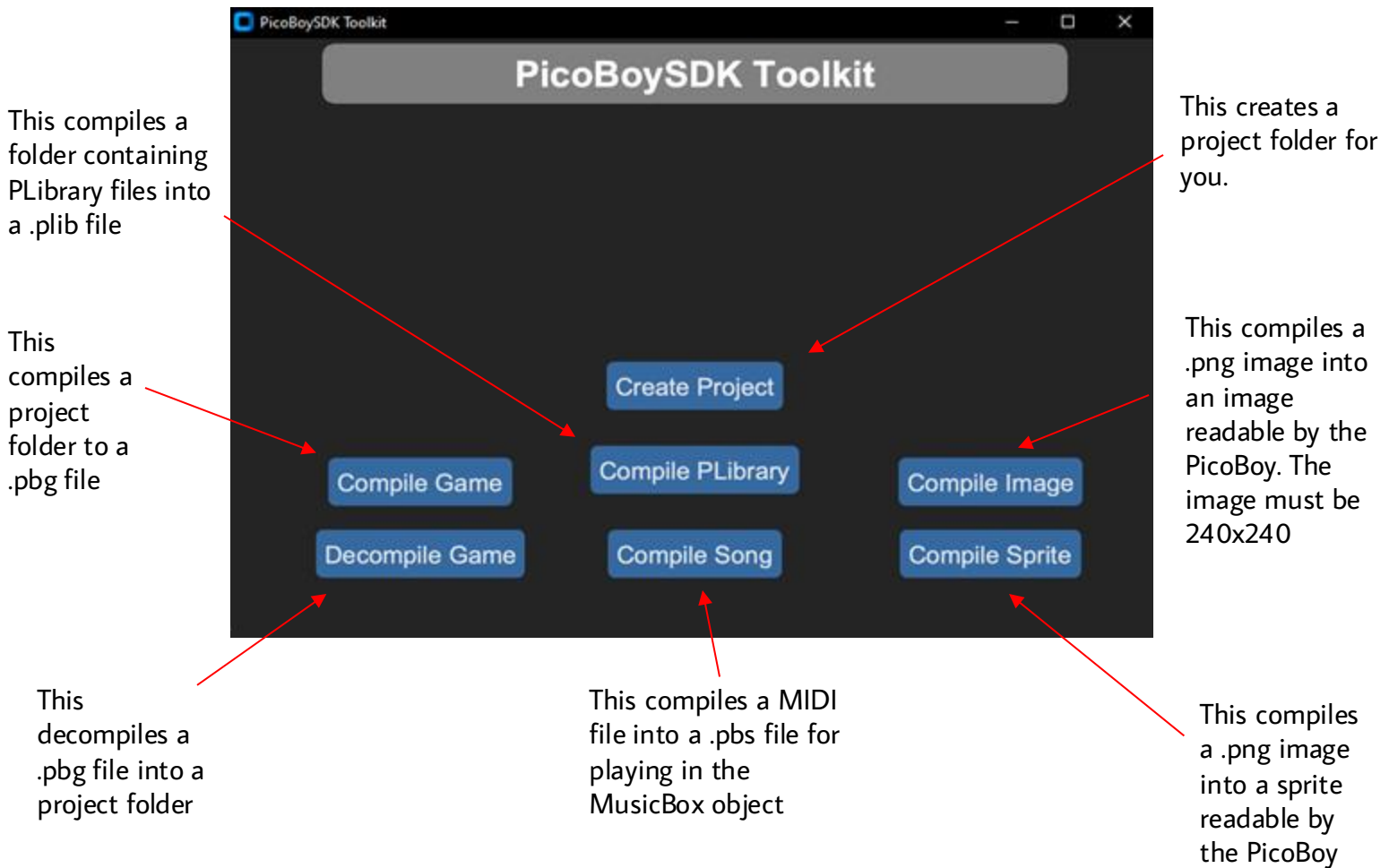
The PicoBoySDK Toolkit:



PicoBoySDK Toolkit

The Functions of the Program

The different functions of the PicoBoySDK Toolkit are documented here:



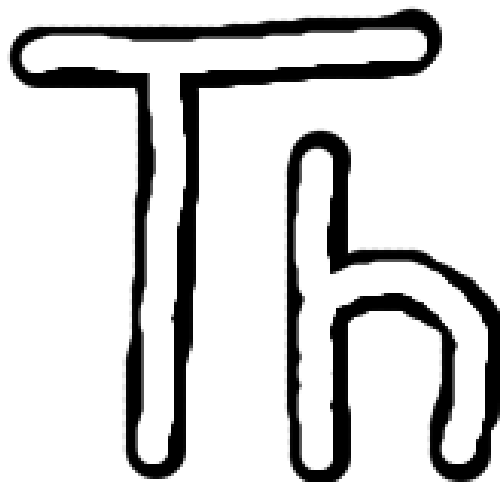
Thonny

Introduction

Thonny is a versatile and beginner-friendly integrated development environment (IDE) tailored for MicroPython, making it an ideal tool for working with your PicoBoy. It provides a streamlined interface that allows you to effortlessly debug your code, ensuring it runs as intended on the device. With Thonny, you can easily upload and download files to and from your PicoBoy, manage its file system, and interact with it directly using the built-in Python shell.

Additionally, Thonny enables you to run Python scripts stored on your computer directly on the PicoBoy, allowing for quick testing and development without permanently saving the code to the device. Whether you're debugging, managing files, or experimenting with new ideas, Thonny simplifies every aspect of the development process.

You can download and install Thonny from <https://thonny.org/>. It is open source and you can find the code on GitHub.com.

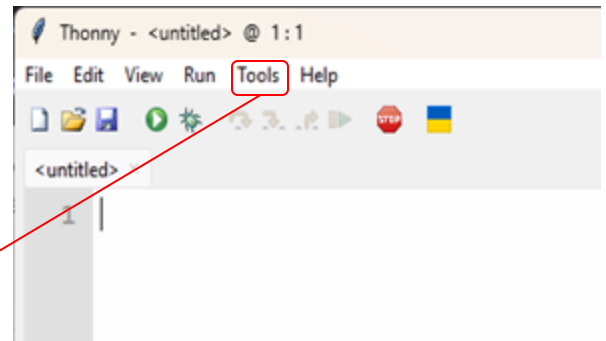


Thonny

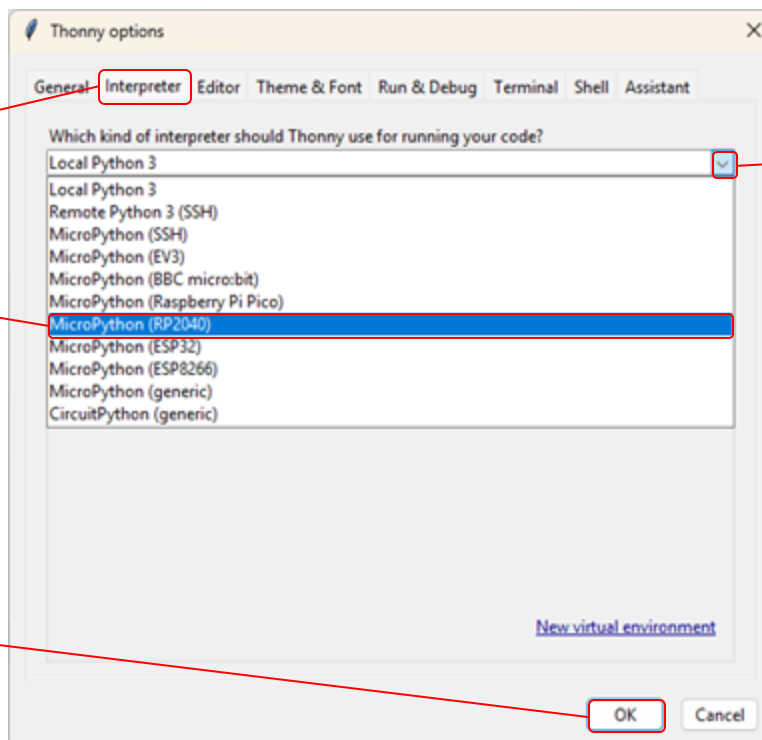
Setting Up Thonny For the PicoBoy

Before you can begin development with Thonny, you need to set it up to work with your PicoBoy:

1. In the application, click on “Tools”, then “Options”.
2. Click on “Interpreter” in the menu bar
3. Click on the dropdown menu labeled “Local Python 3”
4. Select “Micropython RP2040”
5. Click on “OK”



Step 1



Step 2

Step 3

Step 4

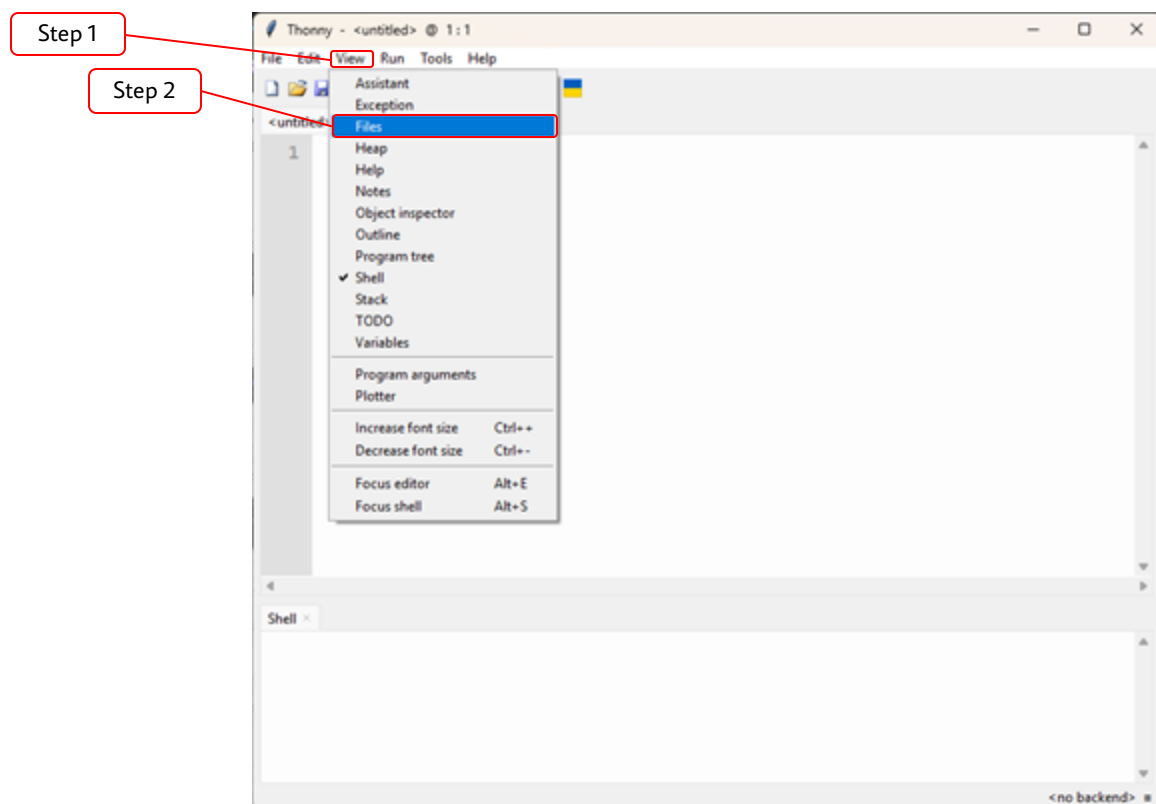
Step 5

Thonny

Setting Up Thonny For the PicoBoy

Now that Thonny is set up to find your PicoBoy, you need to enable file viewing. You can do this by following these steps:

1. Click on "View"
2. Then, click on "Files"



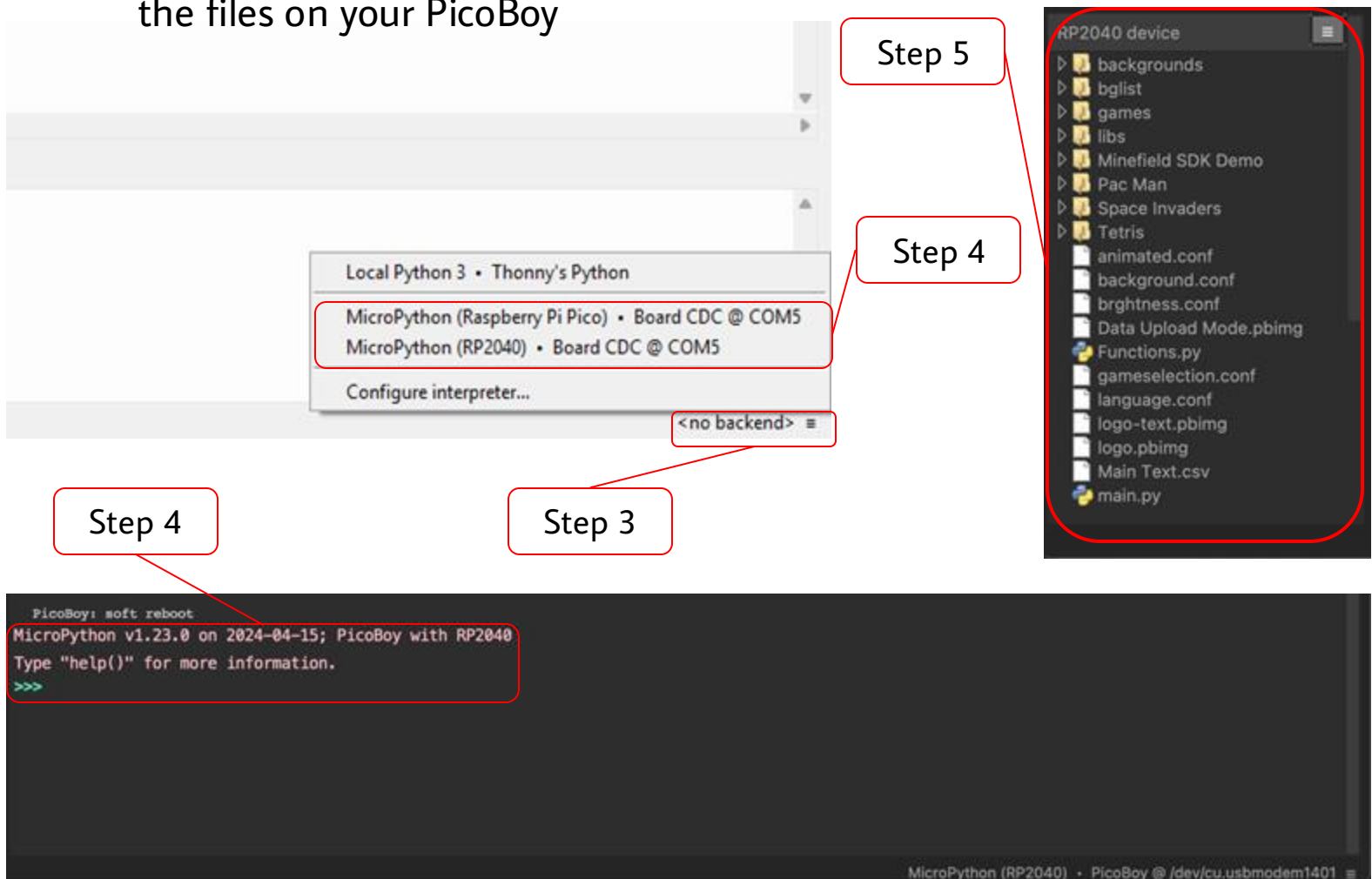
Thonny should now be set up to communicate with your PicoBoy. You should have enabled communication with RP2040 devices and should now see a files tab on the left side of your screen

Thonny

Connecting Your PicoBoy

Before you connect to your PicoBoy, make sure you have completed the previous steps to get Thonny set up for PicoBoy communication. Ensure that you see a files tab on the left side of your screen and the RP2040 Micropython interpreter is enabled.

1. Connect your PicoBoy to your computer
2. You can find it by clicking on "<no backend>"
3. You can select either of the two MicroPython devices to connect.
4. To confirm your PicoBoy is connected, you should see a message in the Thonny terminal
5. You should also see a tap pop up on the bottom left showing the files on your PicoBoy



Thonny

Interacting With Your PicoBoy

Thonny makes it easy to interact with your PicoBoy's Micropython interface. Similarly to how a shell works on a computer, Thonny allows you to issue Micropython commands to your PicoBoy, controlling it and allowing you to perform specific actions.

You can find more information on the Thonny shell on the [Raspberry Pi website](#)

```
PicoBoy: soft reboot
MicroPython v1.23.0 on 2024-04-15; PicoBoy with RP2040
Type "help()" for more information.
>>> variable="Hello World"
>>> print(variable)

Hello World
>>> |
```

Through the Thonny shell, you can take full control of your PicoBoy. You can change its clock speed, memory management, hardware settings, and GPIO output. This can assist you whilst programming, providing you further insight into the way your PicoBoy is running your code.

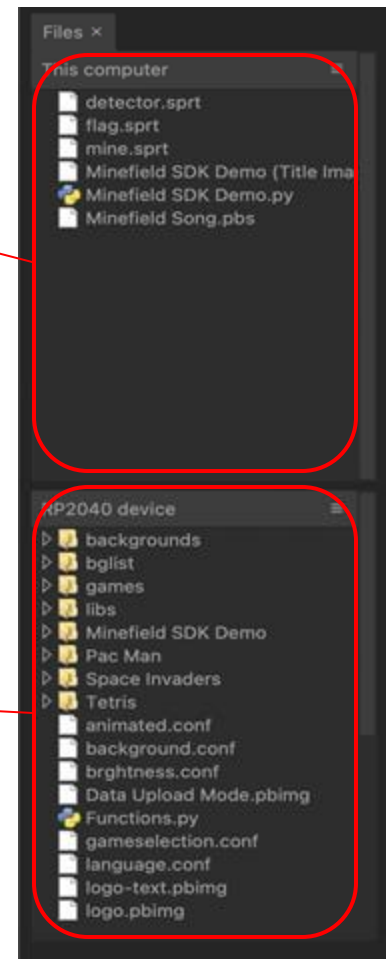
Thonny

Managing Files on Your PicoBoy

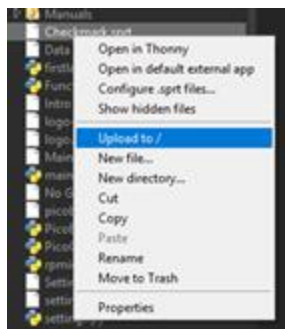
Thonny allows you to directly manage, upload, download, and delete files on your PicoBoy. You can find the files section of Thonny on the left side of the app, divided into files on your computer, and files on your PicoBoy.

This is the section that shows the files on your computer. You can navigate to specific folders. This is where you would open your project folder, so you have access to all the files your game requires.

This is the section dedicated to showing the files on your PicoBoy. You have access to the entire PBOS filesystem, including games, backgrounds, settings, and OS files.

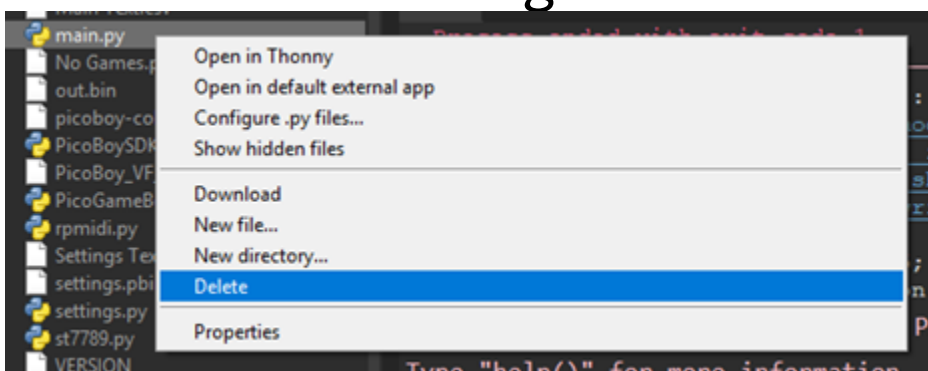


Uploading a File to Your PicoBoy



To upload a file to your PicoBoy, right click on the file you want to upload in the "This Computer" tab of the file manager, then click on "Upload to". To change which folder you are uploading to, navigate to it in the devices tab in the file manager.

Deleting a File on Your PicoBoy



To delete a file on your PicoBoy, right click on the file you want to delete in devices tab in the file manager, then click on "Delete"

Thonny

Running Code on Your PicoBoy

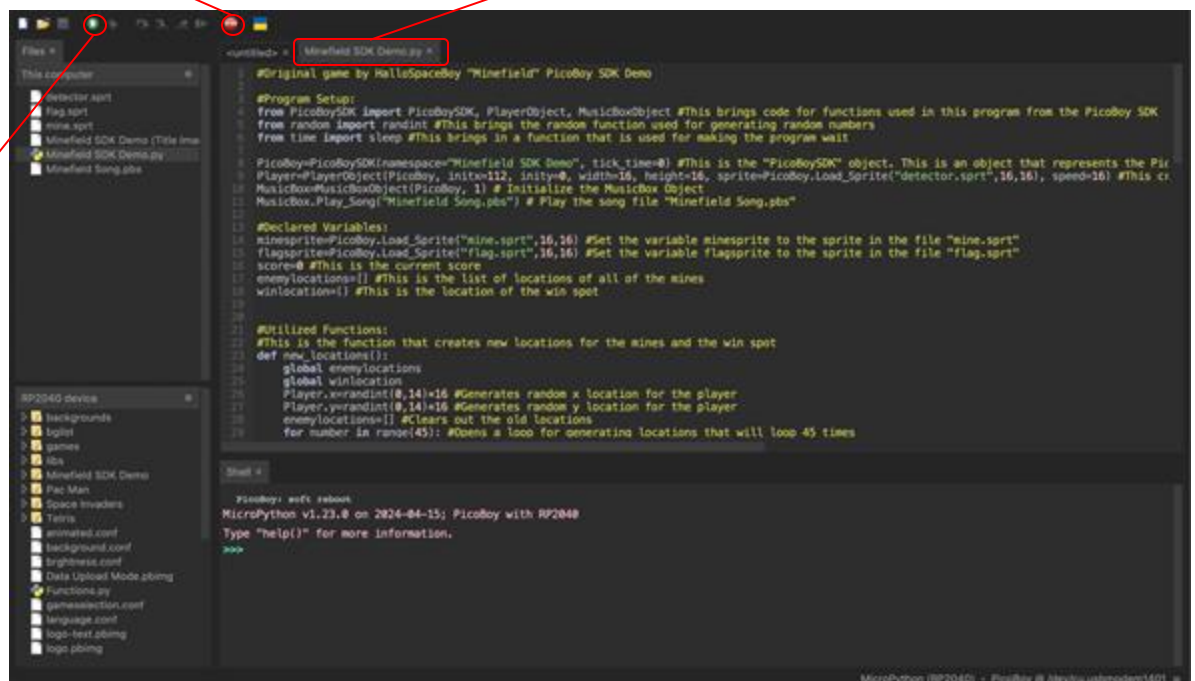
Thonny makes programming on the PicoBoy simple and efficient, allowing you to run code from your computer directly on the device without needing to save it permanently. By opening a code file in Thonny and clicking the play button, the currently open script is automatically loaded and executed on your PicoBoy. This feature is perfect for quickly testing and iterating on your code. Additionally, Thonny's built-in debugger and real-time Python shell allow you to troubleshoot errors, monitor outputs, and interact with your PicoBoy's hardware in a highly intuitive way. In addition to this, Thonny has a force stop button. By clicking the stop sign, you can forcefully stop a program running on your PicoBoy. This is incredibly useful if you happen to write a program that is prone to freezing.

Important: To run a file from your computer on your PicoBoy, the project must first be installed on your PicoBoy. Without installation, the game won't have a designated namespace to run in. After creating a project with the PicoBoySDK Toolkit, it is immediately ready to be compiled and installed, providing the required space for your code to execute. Once installed, you can freely upload and replace the template file from the Toolkit with your updated code, allowing you to iterate and test your game directly through Thonny.

The stop button immediately stops any currently running program.

This is the currently open file, the one that the play button runs.

The play button runs the currently open file



Thonny

Helpful Tips

- When writing a game, make sure you are in the project folder on your PicoBoy when uploading files. The PicoBoySDK looks for files specifically in the project folder, so make sure that is where you are uploading them.
- Sometimes, your PicoBoy can refuse the Thonny connection. To remedy this, simply unplug and replug your PicoBoy into your computer.
- If you recently disconnected and reconnected your PicoBoy, you can just click the stop button to connect your console.
- When you run your code from your computer, it does not save to your PicoBoy. To save your code to your PicoBoy, you can upload it to the project folder.

If you want to learn more about Thonny, you can use this [website from RealPython](#).

The Functions of the SDK

The PicoBoySDK Object

PicoBoySDK(str *namespace*, int *ticktime*)

This initializes the PicoBoySDK object.

Namespace - A string containing the name of the game code file (without the .py extension)

Ticktime - An integer representing the time in seconds to wait every Update()

PicoBoySDK.Load_Sprite(str *filepath*, int *width*, int *height*)

This function opens a sprite file and returns the sprite data.

Filepath - A string containing the file path of the .sprt sprite file

Width and height - Integers representing the width and height of the sprite

PicoBoySDK.Render_Sprite(sprite *sprite*, int *x*, int *y*)

This function renders a sprite at a given position.

Sprite - A variable containing the sprite data from Load_Sprite()

X and y - Integers representing the positional values to render the sprite on the x and y axes

PicoBoySDK.Update(int *savescore*, bool *noclear*)

This function updates the PicoBoy hardware and should be run every game loop. It Checks for the home button, brightness controls, shows what has been drawn to the screen, and waits for *ticktime* amount. After this, it clears the screen.

Savescore - An optional argument that will save the score *savescore* when the home button is pressed.

Noclear - An optional argument that is false by default. If true the screen will not clear out what was previously drawn on it.

The Functions of the SDK

The PicoBoySDK Object

PicoBoySDK.Button(str *button*)

This function checks for a button press on the specified button returning True for a button press and False if a button is not pressed.

Button - A string containing the button to check. The buttons are "A", "B", "Up", "Down", "Left", "Right", "Select", "Start", and "Any"

PicoBoySDK.Outline_Rect(int *x*, int *y*, int *width*, int *height*, tuple *color*)

This function draws a 1 pixel outline of a rectangle at position (*x*,*y*) with the dimensions (*width*,*height*) in the color *color*.

X and *y* - Integers representing the position to draw the rectangle on the x and y axes.

Width and *height* - Integers representing the width and height of the rectangle to draw

Color - A tuple formatted (red,green,blue) with a minimum of (0,0,0) and a maximum of (255,255,255).

PicoBoySDK.Fill_Rect(int *x*, int *y*, int *width*, int *height*, tuple *color*)

This function fills a rectangle with a color *color* at position (*x*,*y*) with the dimensions (*width*,*height*).

X and *y* - The position to draw the rectangle on the x and y axes.

Width and *height* - The width and height of the rectangle to draw

Color - a tuple formatted (red,green,blue) with a minimum of (0,0,0) and a maximum of (255,255,255).

PicoBoySDK.Fill_Screen(tuple *color*)

This function fills the screen with the color *color*.

Color - A tuple formatted (red,green,blue) with a minimum of (0,0,0) and a maximum of (255,255,255).

The Functions of the SDK

The PicoBoySDK Object

PicoBoySDK.Play_Sound(int *tone*, int *channel*)

This function plays a sound out of the PicoBoy's speaker at channel *channel* with the frequency *tone*.

Tone - An integer representing the frequency in hz that the speaker will play

Channel - An integer ranging from 1-4 representing the channel on which a sound will play.

PicoBoySDK.Stop_Sound(int *channel*)

This function stops sound playing on the channel *channel*.

Channel - An integer ranging from 1-4 representing the channel on which a sound will play.

PicoBoySDK.Create_Text(str *text*, int *x*, int *y*, tuple *color*, int *width*)

This function draws the text *text* at the position (*x*,*y*) in the color *color*. The text wraps to the inputted width.

Text - A string that contains the text to display.

X and *y* - Integers representing the positional values to draw the text on the x and y axes. If you set either x or y to be -1, the text will center on the axis of the variable assigned -1.

Color - A tuple formatted (red,green,blue) with a minimum of (0,0,0) and a maximum of (255,255,255) representing color

Width - An integer that represents the maximum width the text can extend to. Any text outside these bounds will be put on a new line.

PicoBoySDK.Load_Image(str *filepath*)

This function loads a 240x240 image file with the extension".pbimg" into the screen.

Filepath - A string containing the path to the image file.

The Functions of the SDK

The PicoBoySDK Object

PicoBoySDK.Load_Small_Image(str *filename*, int *x*, int *y*, int *w* int *h*)

This function draws a small .pbimg

X and *y* - Integers representing the x and y coordinates position of the image

W and *h* - Integers representing the width and height of the image.

PicoBoySDK.Get_Pixel_Color(int *x*, int *y*)

This function returns the color at the position (*x*,*y*).

X and *y* - Integers representing the position of the pixel.

PicoBoySDK.Pause_Screen()

This function pauses the current game loop and pulls up a pause screen. It will exit when the button "Start" is pressed.

PicoBoySDK.Save_Score(int *score*)

This function saves the score *score* to a scoreboard file readable by the PicoBoySDK and PicoBoy Communication Software. If there is no file, one will be created automatically.

Score - An integer representing the score to save

PicoBoySDK.Show_Scores()

This function loads a previously created score file from the function PicoBoySDK.Save_Score(). If there is no file, one will be created automatically.

The Functions of the SDK

The PicoBoySDK Object

PicoBoySDK.Line(int *x*, int *y*, int *x2*, int *y2*, tuple *color*)

This function draws a line between the points (*x*,*y*) and (*x2*,*y2*)

X and *y* - Integers representing the position of the first point

X2 and *y2* - Integers representing the position of the second point

Color - A tuple formatted (red,green,blue) with a minimum of (0,0,0) and a maximum of (255,255,255) representing color

PicoBoySDK.Hline(int *x*, int *y*, int *width*, tuple *color*)

This function draws a line from the origin point (*x*,*y*) to the right with the width *width*.

X and *y* - Integers representing the position of the first point

Width - Integer representing the width of the line

Color - A tuple formatted (red,green,blue) with a minimum of (0,0,0) and a maximum of (255,255,255) representing color

PicoBoySDK.Vline(int *x*, int *y*, int *height*, tuple *color*)

This function draws a line from the origin point (*x*,*y*) to downward with the height *height*.

X and *y* - Integers representing the position of the first point

Height - Integer representing the height of the line

Color - A tuple formatted (red,green,blue) with a minimum of (0,0,0) and a maximum of (255,255,255) representing color

The Functions of the SDK

The PicoBoySDK Object

PicoBoySDK.Load_Library(str *libraryname*, list (str) *objects*)

This function loads the list of objects *objects* from the Plibrary *libraryname*

libraryname - The name of the library you are importing from, make sure that you have the case of the letters correct or it won't find the library.

objects - The list of objects in the library that you are importing into your environment.

PicoBoySDK.Render_Popup(tuple *bgcolor*, tuple *textcolor*, int *x*, int *y*, int *width*, str *title*, list of strings *description*)

This function renders a popup of a width *width* at the location *x,y* on the screen. The color of the body of the popup is *bgcolor* and the color of the text is *textcolor*. The title of the popup is *title* and the lines of text in the popup are *description*. All text within the popup is wrapped to the specified width, height is calculated automatically.

bgcolor - The color of the background of the popup. Formatted in RGB888

textcolor - The color of the text in the popup. Formatted in RGB888

x, y - The X and Y coordinates where the popup will be rendered. Entering -1 will center the popup on that axis.

width - The width of the popup

title - The title at the top of the popup. It is automatically wrapped if it is too long

description - A list of strings, the text within the popup. Each entry in the list is separated from the others. The text is automatically wrapped to the width.

The Functions of the SDK

The PicoBoySDK Object

Collision

PicoBoySDK.Check_Collision

(int *x*, int *y*, int *width*, int *height*, int *x2*, int *y2*, int *width2*,
int *height2*, int *speed*, int *mode*)

This function takes in positional and dimensional data for 2 objects and checks if they have collided. Object 1 is the colliding object, the one that moves. Object 2 is the collidable object, the one that stays still. If Object 1 and Object 2 are both collidable and colliders (both moving) Certain modes of this function may return inaccurate results.

Function Arguments:

- *X* and *Y* are the variables that represent the position of object 1 (Integer)
- *Width* and *Height* represent the width and height of object 1 (Integer)
- *X2* and *Y2* are the variables that represent the position of object 2 (Integer)
- *Width2* and *Height2* represent the width and height of object 2 (Integer)
- *Speed* represents the speed at which object 1 is moving towards object 2. The object that moves is object 1 and the object that is still is object 2 (Integer)
- *Mode* represents the mode of collision that will be calculated. Mode 0 returns a set of coordinates with a corrected position that is not colliding. Mode 1 returns either True or False depending whether or not a collision has occurred. Mode 2 returns the x adjust and y adjust which can be used to transform an object's position. (Integer)

The Functions of the SDK

The PlayerObject Object

PlayerObject(object *parent*, int *initx*, int *inity*, int *width*, int *height*, sprite *sprite*, int *speed*)

This initializes the Player object. Parent is the PicoBoySDK object, if you do not plug in a PicoBoySDK object, you will get an error.

Initx and *inity* - Integers representing the initial positional coordinates for the object.

Width and *height* - Integers representing the width and height of the sprite that this object uses.

Sprite - The sprite that this object is represented as, it can be assigned by using PicoBoySDK.Load_Sprite().

Speed - An integer representing the speed in pixels that the character will move every update if a button is pressed.

PlayerObject.Update()

This function updates the player data, it checks for movement and renders its sprite. It performs all of the functions the player needs to function. This should be run every game loop.

PlayerObject.Goto(int *x*, int *y*)

This function moves the player to the coordinates *x* and *y*.

X and *y* - Integers that represent the positional values on the x and y axis.

PlayerObject.Change_Axis(bool *x*, bool *y*)

This function changes the axis that the player moves along. You can set any combo of true or false to change the axis of movement. If you set both to false it will default to movement on both axes.

X and *y* - Booleans that represent the axes to move along.

The Functions of the SDK

The PlayerObject Object

The Variables of the PlayerObject:

These variables are the different variables that the PlayerObject uses in its functions. They can be invoked and assigned by using `PlayerObject.variable` where *variable* represents the variable in the object.

Int *PlayerObject.x* - The integer positional value of the player on the x axis

Int *PlayerObject.y* - The integer positional value of the player on the y axis

Int *PlayerObject.initx* - The integer initial positional value of the player on the x axis

Int *PlayerObject.inity* - The integer initial positional value of the player on the y axis

Int *PlayerObject.width* - The integer width of the player

Int *PlayerObject.height* - The integer height of the player

Sprite *PlayerObject.sprite* - The sprite that represents the player

Here is an example of how to change the *x* and *y* positional values of the PlayerObject:

```
PlayerObject.x=100
```

```
PlayerObject.y=100
```

The Functions of the SDK

The MusicBoxObject Object

Note: This object takes up A LOT of memory. Make sure to factor this into your game before coding to make sure you don't run out of memory.

MusicBoxObject(object *parent*, int *mode*)

This function initializes the MusicBoxObject object.

Parent - The PicoBoySDK object. If this is not the PicoBoySDK object, an error will occur.

Mode - An integer between 1 and 0, mode 0 makes the song stop after one play and mode 1 makes the song loop.

MusicBoxObject.Play_Song(str *filepath*)

This function opens the song file *filepath* and begins playing it.

Filepath - A string containing the file path that leads to a song file represented as a string.

MusicBoxObject.Stop_Song()

This function stops the current song from playing.

MusicBoxObject.Change_Mode(int *mode*)

This function changes the current mode into *mode*.

Mode - An integer between 0 and 1, mode 0 makes the song stop after one play and mode 1 makes the song loop.

Limitations of the Hardware

The PicoBoy, while powerful, has some limitations to be aware of while developing a game for it.

- **Memory constraints:** The PicoBoy has about 264 kb of RAM. While that seems to be a lot, the amount of workable memory is about 90 kb while a program is loaded and running.
- **Processing speed:** The PicoBoy runs at 133mhz, this means that it is not all too powerful when it comes to complex tasks. If you are performing a lot of functions that use intense processing (EX: too many collisions), the PicoBoy will start to slow down.
- **4 channels of PWM sound:** The PicoBoy can sustain four simultaneous sounds being played at once, 3 for music, 1 for sfx. Of course, you can always use all four for either, however it is wise to use 3 for music and one for sfx. Be mindful of the frequencies the speakers play at, as too low or too high a volume can cause distortion on some devices.

Final Notes

- If you are working on an especially memory intensive project. You can use the deprecated PicoGameBoy library to save some memory. Just note that everything in this library has to be done manually, you need to perform system functions the SDK takes care of, and there is no documentation.
- If you are having any issues, you can join the [discord server](#) to ask any questions you have.
- If you are experiencing any bugs with the SDK or have a suggestion for the SDK, post about it on the discord server.