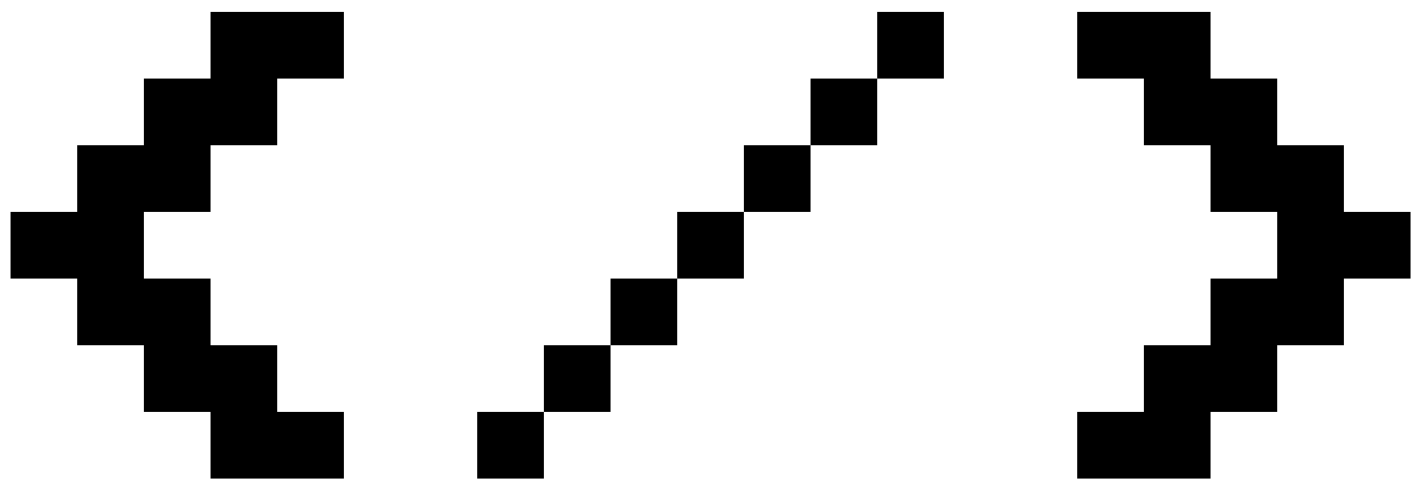


PicoBoy SDK Docs



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
- A Step-by-Step Guide

2. PicoBoySDK Toolkit

- Introduction
- The Functions of the Program

3. The Functions of the SDK

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

4. The Limitations of the Hardware




5. Final Notes

Creating a Game

The Project File Structure

The file structure of a PicoBoySDK game consists of the main game code file 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. 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	

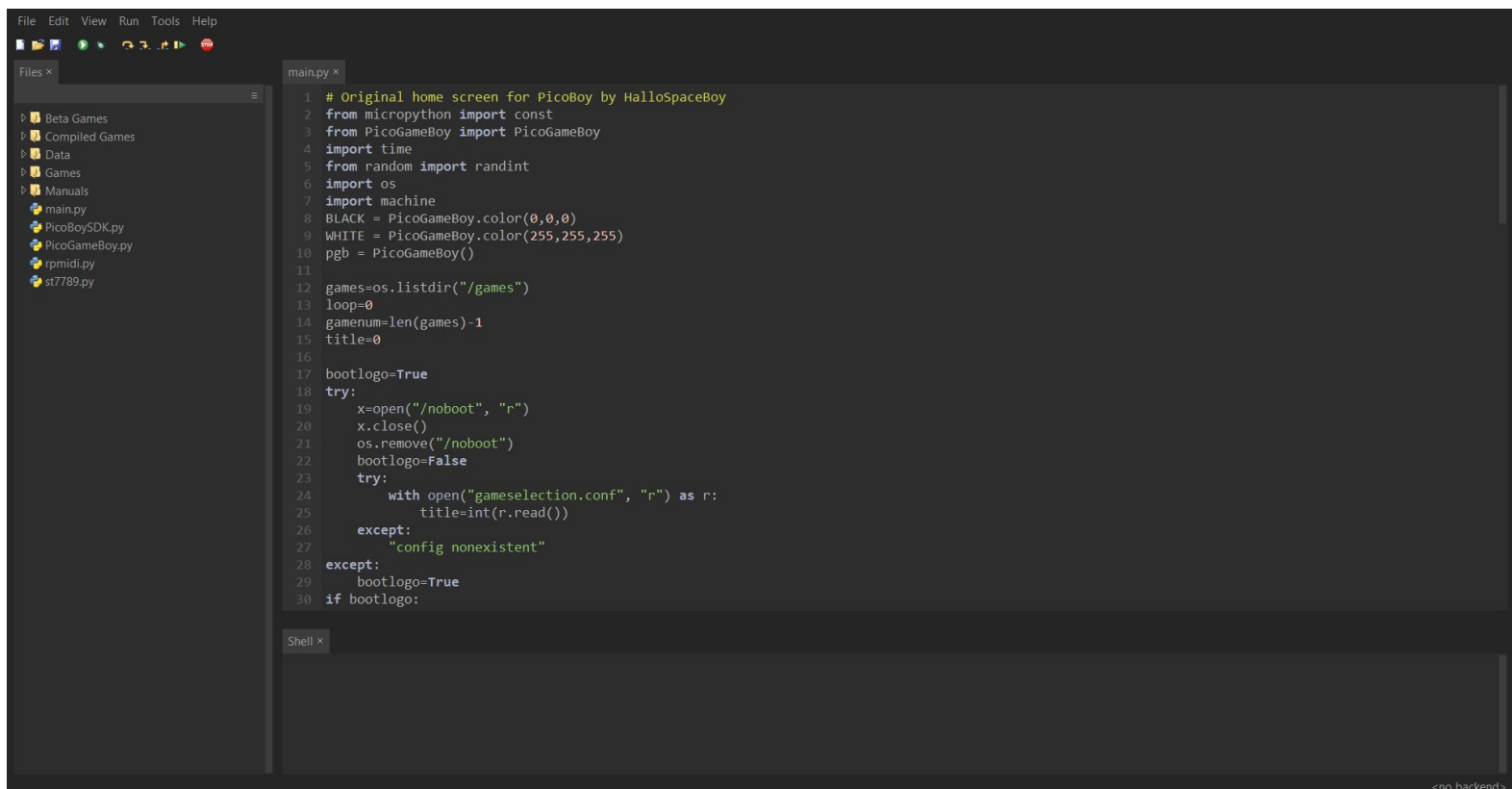
Creating a Game

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.

You can find a guide for setting up Thonny [here](#). (Do not follow the part on installing libraries/packages)

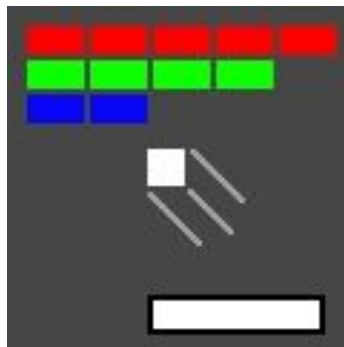
Here is an example of the Thonny IDE:



Creating a Game

The Home Screen

When the PicoBoy boots, it displays a game selection “home” screen. 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 name of the game. 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 foldername+“(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.



Example of the Breakout
title image

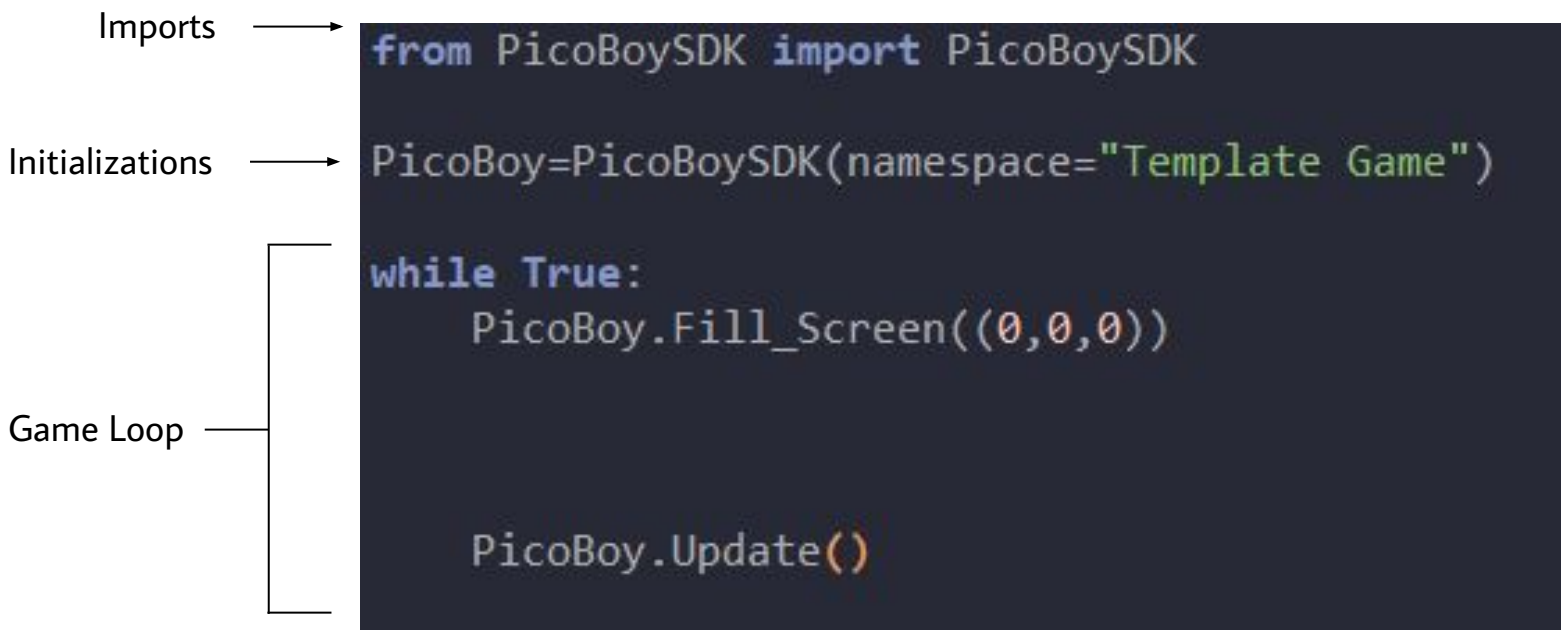


Example of title image naming scheme

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.

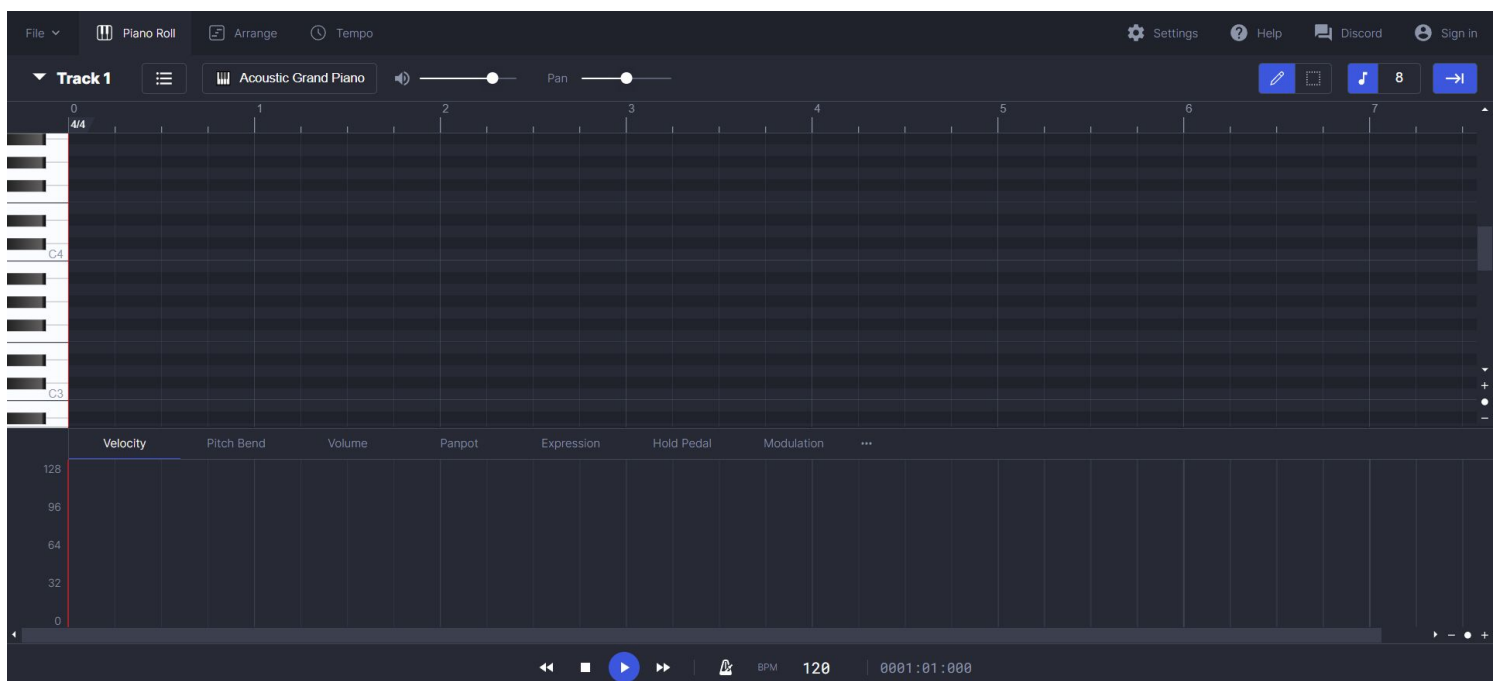


An example of the code structure from the Template Game

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.

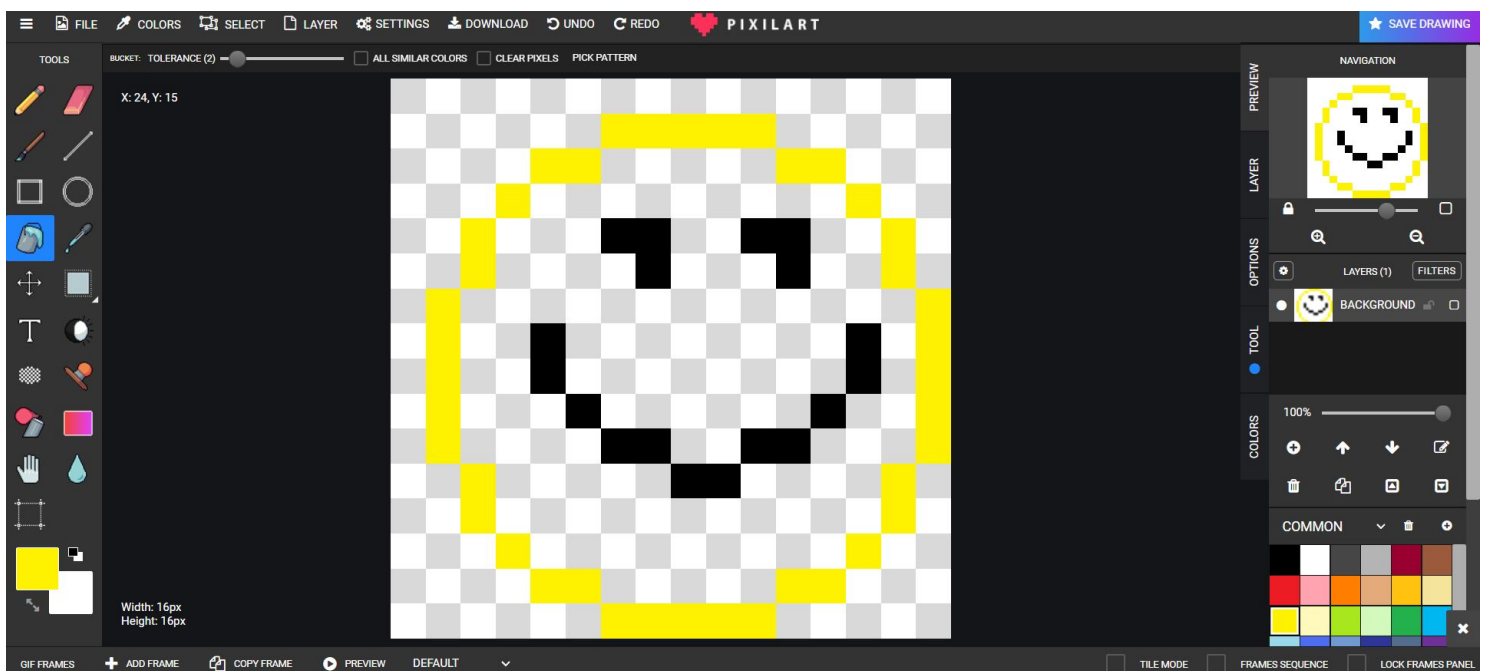


An image of the Signal editor

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 must be 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

A Step-by-Step Guide

1. Install the template game included in this package. Make sure that you have no more than 3 other games on your PicoBoy to account for space to be used in the future.
2. Open the template project folder in Thonny and connect to your PicoBoy.
3. Open the template game code file and run it, it should navigate you to it's game folder on your PicoBoy.
4. Begin modifying the template game code to perform the instructions you want it to perform.
5. When you want to test your code, you can press the play button or F5 to run your code.
6. Once your game behaves how you want it to, you can change its name and compile it. This can be achieved in three steps. 1, rename the project folder to be the name you want. 2, rename the game code .py file to be *name.py* where name is the name of the game. 3, change the *namespace* variable in the PicoBoySDK object to be the name of the game code file. 4, rename the .pbimg file to be "*name* (Title Image).pbimg" where *name* is the name of your game code file.
7. Open the PicoBoySDK Toolkit and compile the project folder into a .pbg file.
8. Delete the template game off of your PicoBoy and replace it with your newly made .pbg file to continue making the game under its new name.

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.

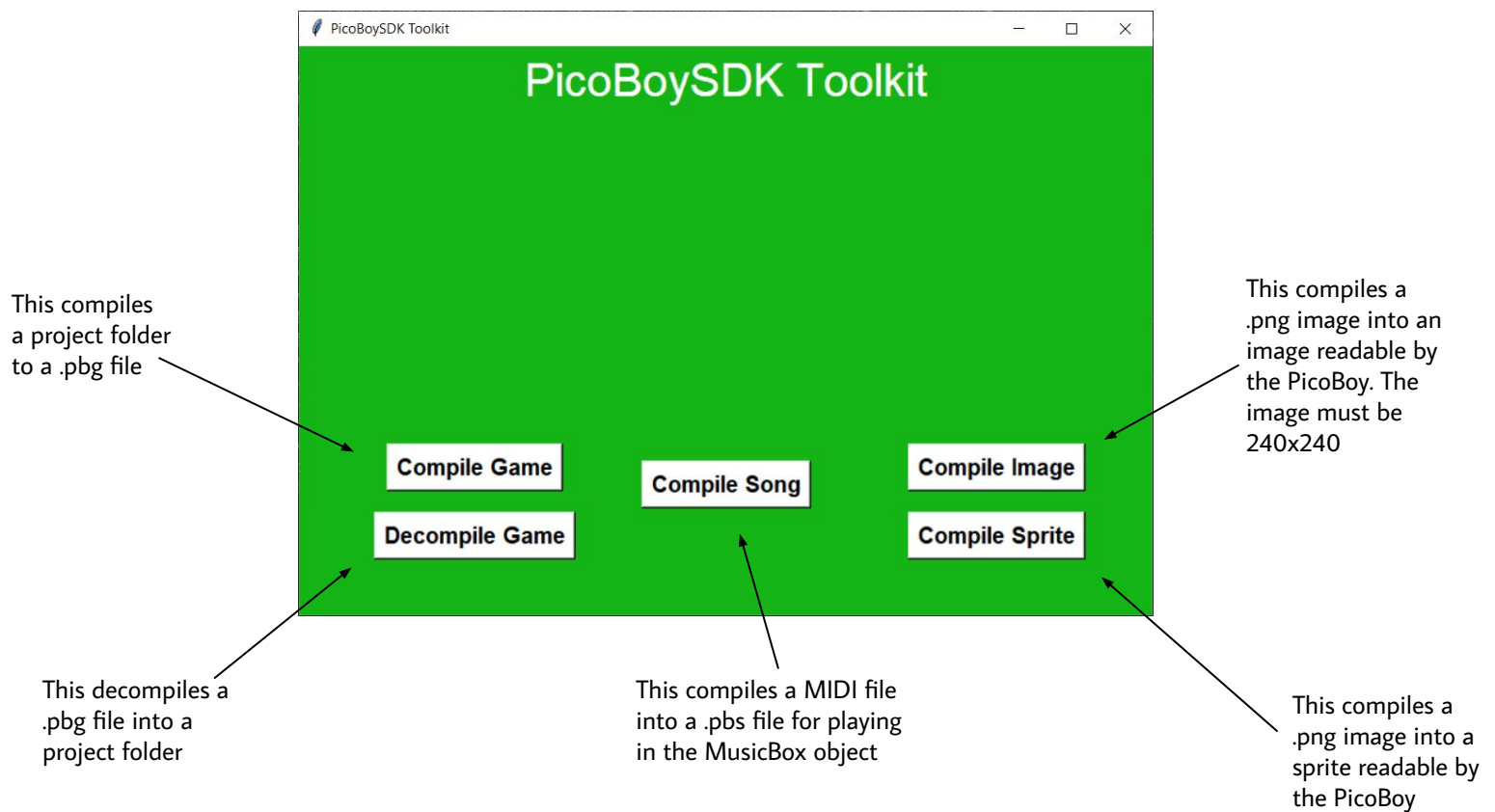
The PicoBoySDK Toolkit:



PicoBoySDK Toolkit

The Functions of the Program

The different functions of the PicoBoySDK Toolkit are documented here:



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()

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.

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*)

This function draws the text *text* at the position (x,y) in the color *color*.

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

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.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

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.

Player.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.

Player.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.

Player.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.

MusicBox(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.

MusicBox.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.

MusicBox.Stop_Song()

This function stops the current song from playing.

MusicBox.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 and 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: 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. The usage of PWM sound means that volume cannot be digitally controlled.

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.