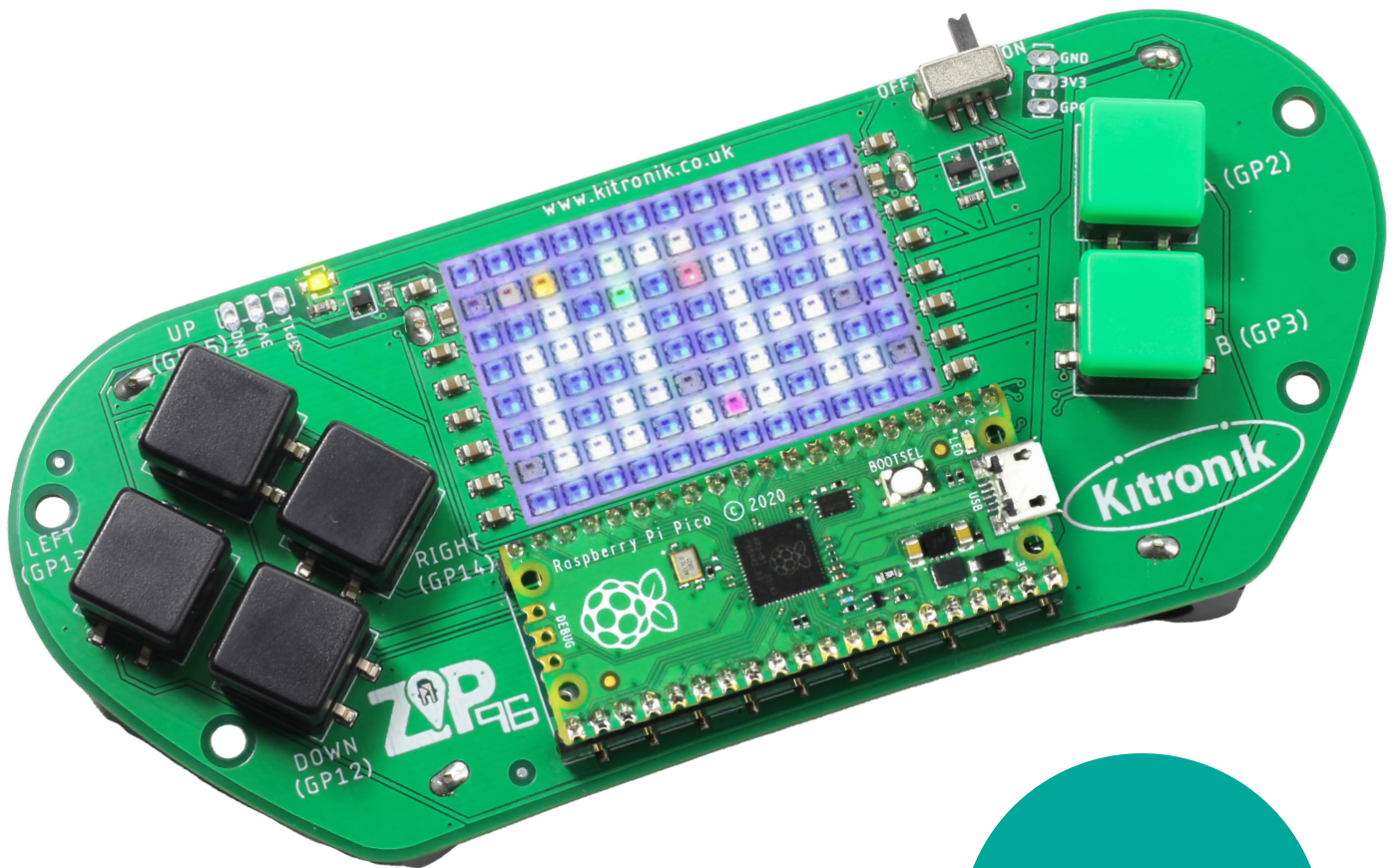


# LESSON GUIDE TO THE PICO ZIP96



## LESSON 2: THE A-MAZING GAME



This lesson includes curriculum mapping, practical exercises and a linked PowerPoint presentation.

14+

[www.kitronik.co.uk](http://www.kitronik.co.uk)

TEACH YOUR STUDENTS HOW TO CREATE GAMES WITH CODE!

## LESSON 2

# INTRODUCTION & SETUP



This is the second lesson in the 'A-mazing Game' series for Pico ZIP96. Having looked at the overall program design in the first lesson, this lesson starts the actual coding of the game, creating the Player class and testing the player movement on the screen.



## CLASSROOM SETUP



For this lesson, you will need:

- Pen & Paper
- A computer/laptop with a USB port and Internet access
- Raspberry Pi Pico H
- Kitronik Pico ZIP96
- 3 x AA batteries
- A micro USB cable
- A copy of the Kitronik ZIP96 library added in step 4 of the Thonny setup, or ZIP96Pico.py in Lessons Code folder

The teacher will be writing on the board as well as demonstrating code on a projected board (if available).



### Curriculum mapping

- Understanding tools for writing programs. Using sequence, variables, data types, inputs and outputs.
- Decompose problems into smaller components, and make use of subroutines to build up well-structured programs.

## KEYWORDS:

TRANSLATORS, INTEGRATED DEVELOPMENT ENVIRONMENTS (IDES), ERRORS, SEQUENCE, VARIABLES, DATA TYPES, INPUTS, OUTPUTS, SUBROUTINES, PROCEDURES, FUNCTIONS, MODULES, LIBRARIES, VARIABLE SCOPE, WELL-DESIGNED PROGRAMS, OBJECT-ORIENTED PROGRAMMING (OOP), CLASSES

## WHAT IS OUR A-MAZING GAME?



Our A-Mazing Game is a maze-based game where the player runs through a maze collecting gems. Once you have collected all the gems in the maze then you have won! But it won't be that easy, as there are enemies in the maze trying to catch you before you collect all of the gems. The three enemies each have their own level of difficulty. The first enemy moves randomly. The second enemy tries to move towards you, without trying to avoid the maze walls. The third and final enemy is smart and moves around the maze walls finding the best path to you. When an enemy catches you, you lose a life and everyone in the game is reset back to their starting positions. You have three lives to collect all the gems, and if you don't, then you lose.

## LESSON 2



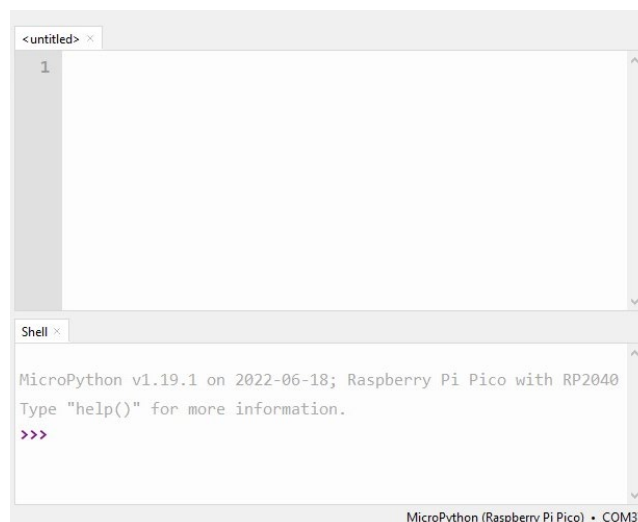
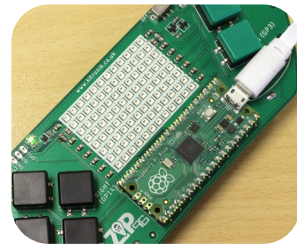
### Curriculum mapping

Understanding tools for writing programs.

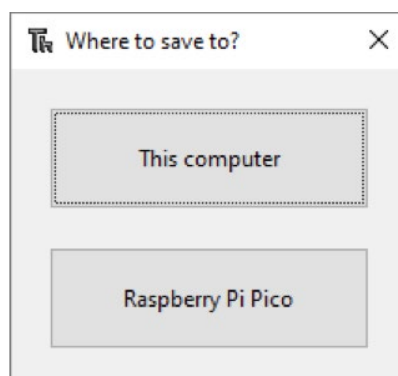
## SETUP

Students will need to work in pairs, having one device per pair.

- 1 Start by having the students setup the ZIP96 Pico by connecting it to a computer and opening up Thonny.
- 2 The Pico device will appear in the bottom right corner of Thonny.
  - If the device does not load automatically, try pressing the STOP icon at the top of the screen.
  - If the shell does not load automatically, turn it on by checking **View > Shell**.



- 3 Create a new file by clicking **File > New** and save this to your Pico as main.py by clicking **File > Save as** and selecting **Raspberry Pi Pico**.



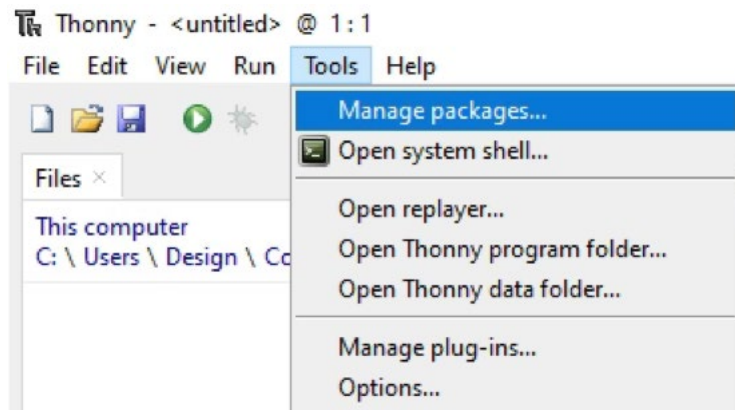
## LESSON 2

### SETUP CONTINUED

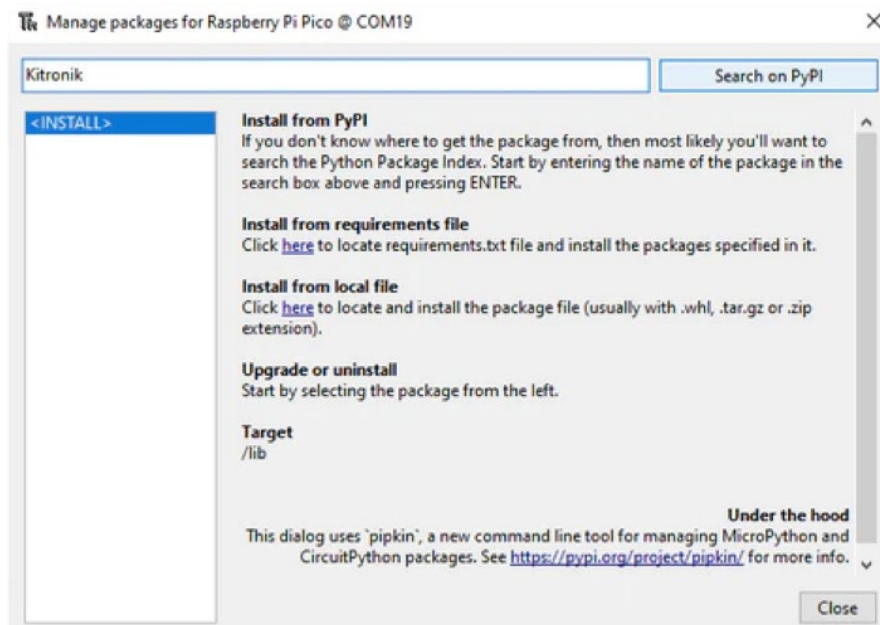
- 4 Now we can install the ZIP96Pico library onto our Pico.



- To do this we need to click **Tools > Manage Packages...** from the drop down menu.



- With the Manage packages window open we can now search for Kitronik in the text box at the top, and click the Search on PyPI button. Thonny will search the Python Package Index for all the Kitronik packages.



- Click on KitronikPicoZIP96 from the search results list. This will show us details about the package and from here we can click the Install button to add the package to our Pico. Thonny may ask you to confirm that you would like to install this package and we want to select Yes, we do want to install the package.

# MAIN LESSON

## WHAT IS A PROGRAMMING LIBRARY?



A library in programming is a file or group of files that implement in code some specific functionality. Libraries usually provide functionality that is commonly used in creating programs and are intended to make software development easier whilst also making code simpler. To use a library and have access to its functionality you need to include it in your code, which is done by including or importing the name of the library at the top of the file you are writing your code in.

Most of the libraries we'll use in our game will be from the standard MicroPython libraries created for the Raspberry Pi Pico. One extra library will be needed for the Pico ZIP96 and is available on the Kitronik Website or from the Lessons code folder. You should save the ZIP96Pico.py file onto your Raspberry Pi Pico so we can use the library later.

## GAME SETUP

To start creating the A-Mazing game we first need to include some libraries to make our code easier to write. We'll import two libraries into our main.py file. The first import includes all of the functions from the ZIP96Pico library which allows us to communicate with and control the different bits of hardware on our device. The second import includes the sleep function from the utime library to allow us to add delays into our game.

```
import ZIP96Pico
from utime import sleep
```



### Curriculum mapping

Using sequence, variables, data types, inputs and outputs.

Next, let's setup some variables that we can use for our game. We will create some variables to define our Pico ZIP96. The first two, screenWidth and screenHeight, define the width and height to be 12 and 8 respectively, as this is the number of pixel columns and rows we have on our Pico ZIP96. We then initialise our Pico ZIP96 Pico as gamer from the Pico ZIP96 library we imported earlier. Using the gamer object we can get the screen object to give us easy access to the LEDs that make up our Pico ZIP96 screen.

```
# Setup ZIP96Pico variables
screenWidth = 12
screenHeight = 8
gamer = ZIP96Pico.KitronikZIP96()
screen = gamer.Screen
```

## LESSON 2



### Curriculum mapping

Decompose problems into smaller components, and make use of subroutines to build up well-structured programs.

## CREATE GAME CLASS: *PLAYER*

To separate out the different sections of variables and functions in our game we are going to write some classes from which we can create objects to access the functionality. First, we will write a Player class to store everything related to the player of our game. A standard way to provide an object with information about itself is to write a constructor in the class.

In Python this is done by defining, using `def`, an `__init__` function to initialise an object with some values. It is important that for any function we create inside of a class we add the `self` keyword to its parameters as it allows us to access the variables and functions stored inside of the class. (Note that `self` doesn't need to be included in the inputs used when calling the function as it is included automatically by Python).



Ask the students to try to identify some of the information the Player class will need to store?



The Player will need to know:



- its x and y position on the screen,
- the colour used for the players LED,
- the screen object to draw the player on the Pico ZIP96,
- the screenWidth and screenHeight to detect when the player moves off the screen.

So, we should take these as input parameters in the Player constructor and set them as variables stored inside the object.

# Class to store our player functionality

**class** Player():

    # Player object constructor

**def** `__init__`(**self**, x, y, colour, screen, screenWidth, screenHeight):

**self**.x = x

**self**.y = y

**self**.colour = colour

**self**.screen = screen

**self**.screenWidth = screenWidth

**self**.screenHeight = screenHeight





## LESSON 2

### CREATE GAME FUNCTION: *PLAYER.DRAW*

To draw the player on the Pico ZIP96 we need to define a new function, draw, that doesn't require any input parameters from outside the class. To draw the player we want to use its x and y position and the colour used for the player's LED. We can use the self keyword to access the variables inside the class and supply these as inputs to the setLEDMatrix function on the screen which will update the LED colour for a given x and y position.

```
# Draw the current position on the screen
def draw(self):
    self.screen.setLEDMatrix(self.x, self.y, self.colour)
```



We can add a call to the draw function inside the constructor for our Player class underneath the line where we set the screenHeight value.

```
self.screenHeight = screenHeight
# Draw the starting position
self.draw()
```

### TASK: TEST *PLAYER* CONSTRUCTOR



Let's now test our Player class by creating an object player from it. We'll add this code at the bottom of our main.py file. To use the constructor for a class we use the class name as a function and supply it with the necessary inputs. After creating the player object we'll need to show any updates on the ZIP96 screen using the show function to update the LED values.

```
# Create an object for our player
player = Player(0, 1, gamer.Screen.YELLOW, screen, screenWidth, screenHeight)

# Show the created objects on the screen
screen.show()
```

We can run the code by clicking the green play button at the top of Thonny. Then try changing some values we give to the player constructor and see how this changes your LED screen.

## LESSON 2



Which of the Player class parameters need to be altered to add movement to the player?



### CREATE GAME FUNCTION: *PLAYER.MOVE*



To get the player to move around the maze we need to provide some way to change the x and y values stored in the Player class. Let's define a move function that takes an x and y value as its inputs. These values will determine how much our player object will move by. To update the object's position we need to use the self keyword followed by our variable name, then the += operator and finally the input value. This will add the input x value to the object's x value and store it back into the object's x value. The same happens with the y value. Next, we can update the ZIP96 screen with the new position the player has on the screen.

```
# Update the current position
def move(self, x, y):
    # Change the current position by the x and y parameters
    self.x += x
    self.y += y

    # Update the new position on the screen
    self.draw()
```

### TASK: TEST *PLAYER.MOVE* FUNCTION



Now try using the move function on your player object. Try using several different values as the input and see if you can figure out what is wrong with our move function. Also try calling move multiple times, as there is more than one problem we need to solve.



**Note:** Don't forget to call show on the screen after updating the player position.



**LESSON**  
**2**

# CONCLUSION

## LESSON 02 CONCLUSION



In this lesson, you have:

- Imported libraries
- Setup the Pico ZIP96 as the device variable
- Setup game variables of types integer, string
- Created Player class, constructor and methods

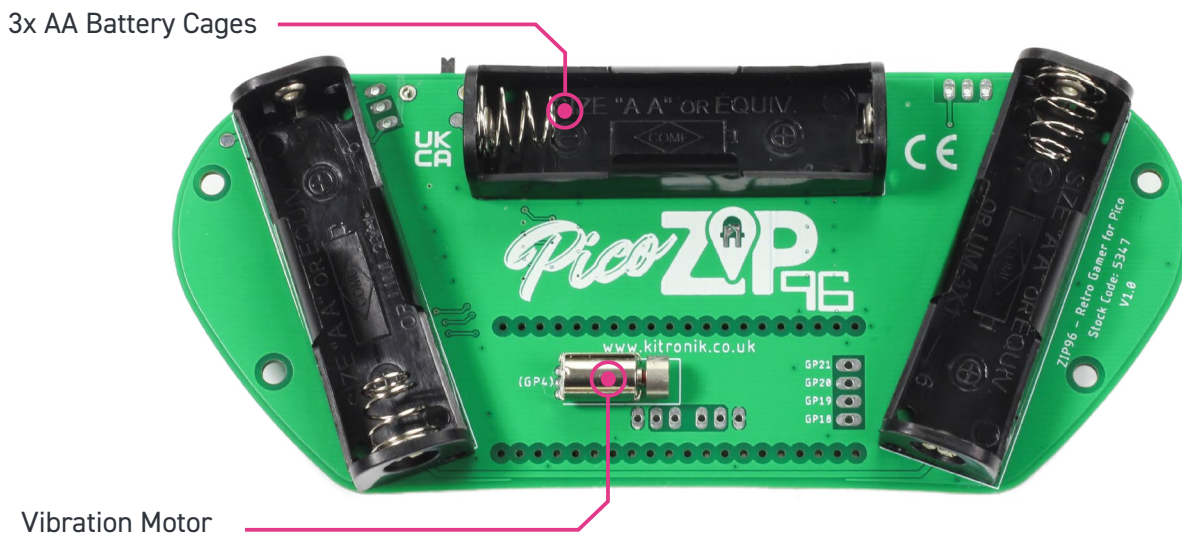
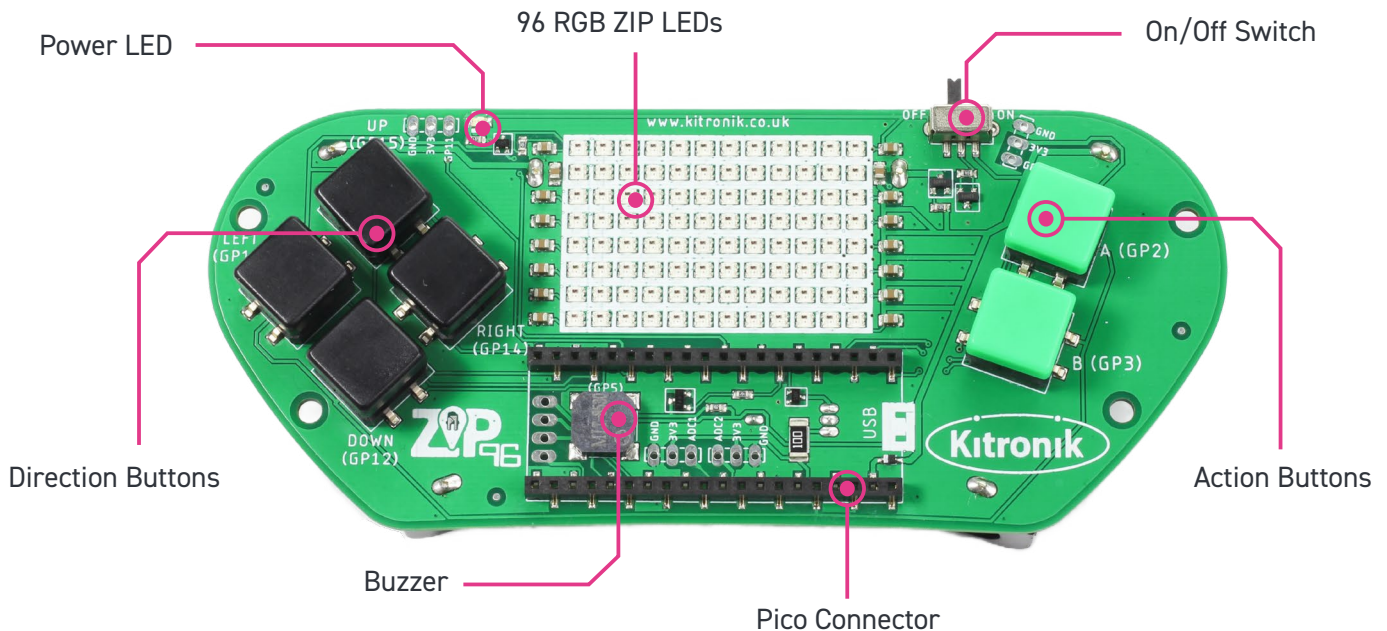
## FULL LESSON CODE

The full code for each lesson can be found in the Lessons Code folder.



## THE ZIP96 IS A PROGRAMMABLE RETRO GAMEPAD FOR THE RASPBERRY PI PICO.

It features 96 colour addressable LEDs arranged in a 12 x 8 display, a buzzer for audio feedback, a vibration motor for haptic feedback, and 6 input buttons. It also breaks out GP1, GP11, ADC1 and ADC2, along with a set of 3.3V and GND for each, to standard 0.1" footprints. GP18 to 21 are also broken out on a 0.1" footprint underneath the Pico. The Pico is connected via low profile 20-way pin sockets.



T: 0115 970 4243

W: [www.kitronik.co.uk](http://www.kitronik.co.uk)

E: [support@kitronik.co.uk](mailto:support@kitronik.co.uk)



[kitronik.co.uk/twitter](https://twitter.com/kitronik)



[kitronik.co.uk/facebook](https://www.facebook.com/kitronik)



[kitronik.co.uk/youtube](https://www.youtube.com/kitronik)



[kitronik.co.uk/instagram](https://www.instagram.com/kitronik)



Designed & manufactured  
in the UK by **Kitronik**



For more information on RoHs and CE please visit [kitronik.co.uk/rohs-ce](http://kitronik.co.uk/rohs-ce). Children assembling this product should be supervised by a competent adult. The product contains small parts so should be kept out of reach of children under 3 years old.