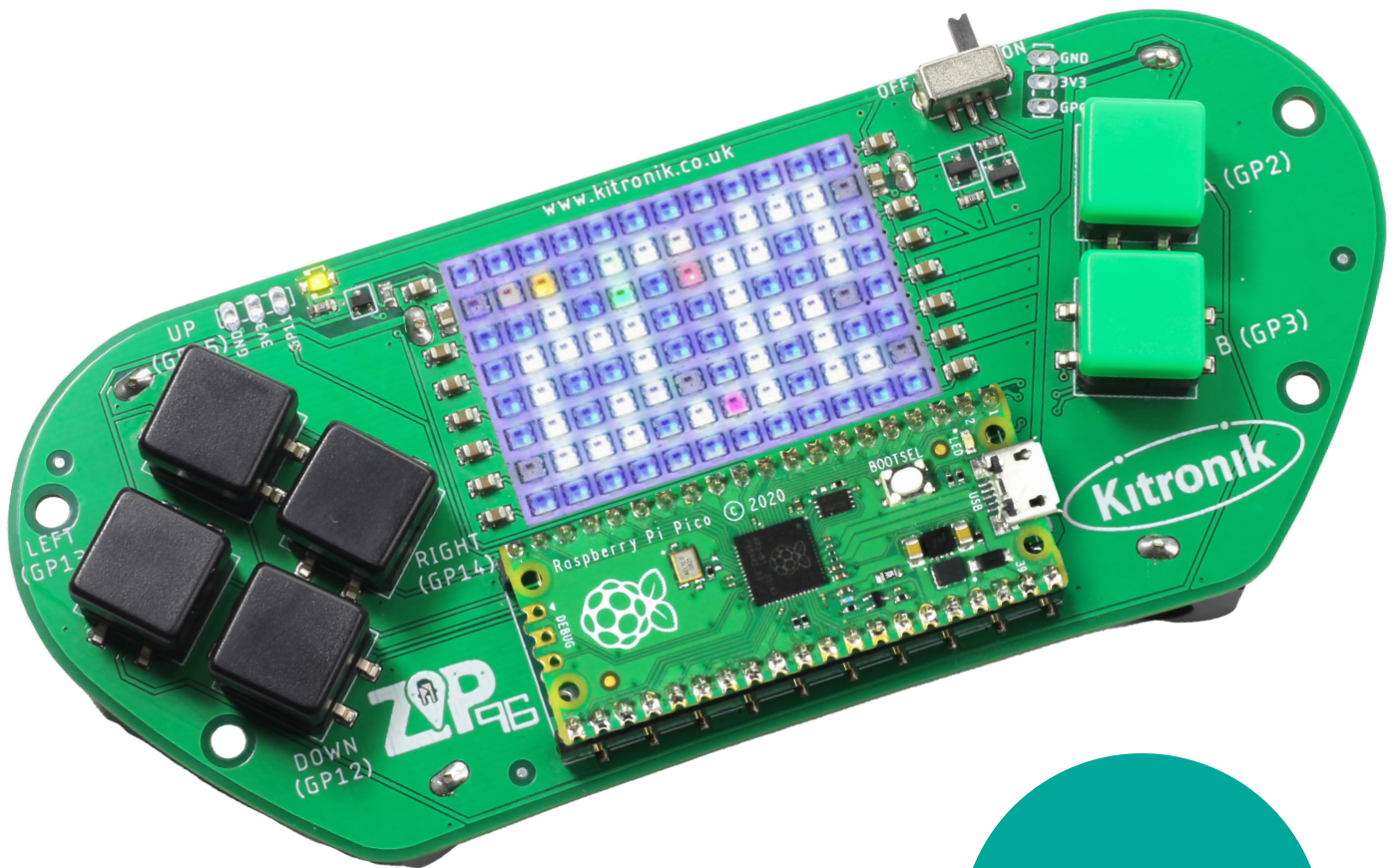


# LESSON GUIDE TO THE PICO ZIP96



## LESSON 5: 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 5

## INTRODUCTION & SETUP



This is the fifth lesson in the 'A-mazing Game' series for Pico ZIP96. Now that the player can move around the maze structure, gems are added in the lesson to provide a game objective.



### CLASSROOM SETUP



**Students will be working in pairs. They 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 (ZIP96Pico.py in Lessons Code folder)
- A copy of last lesson's code (ZIP96Pico - A-Mazing Game - Lesson 04.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.
- Learn how to handle strings, and simplify solutions by making use of lists and arrays.
- Apply iteration in program designs using loops.
- Make decisions in programs, making use of arithmetic, logic and Boolean expressions and operators. Created nested selection statements.

### KEYWORDS:

TRANSLATORS, IDES, ERRORS, SEQUENCE, VARIABLES, DATA TYPES, INPUTS, OUTPUTS, ITERATION, WHILE LOOPS, FOR LOOPS, NESTED STATEMENTS, DESIGN PROGRAMS, SELECTION, CONTROL STRUCTURES, LOGIC, BOOLEAN, NESTED STATEMENTS, SUBROUTINES, PROCEDURES, FUNCTIONS, MODULES, LIBRARIES, VARIABLE SCOPE, WELL-DESIGNED PROGRAMS, STRINGS, LISTS, STRING HANDLING, ARRAYS (1D, 2D), MANIPULATION, ITERATION

# LESSON 5

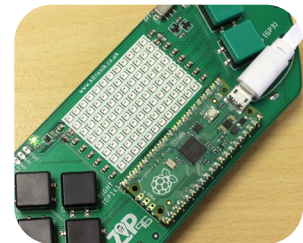
## WHAT IS OUR A-MAZING GAME?



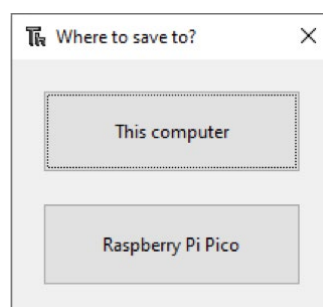
Our A-Mazing Game is a maze-based game where the player runs through a maze collecting gems. Once you have collected all the gems in the maze then you have won! But it won't be that easy, as there are enemies in the maze trying to catch you before you collect all of the gems. The three enemies each have 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.

## SETUP

- 1 Start by having the student's 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**.



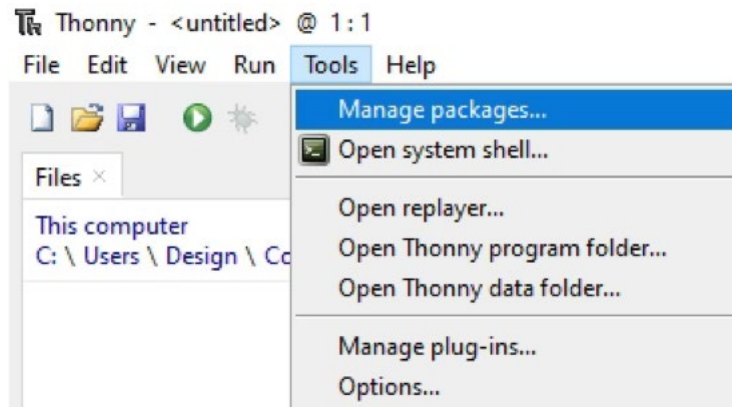
# LESSON 5

## SETUP CONTINUED

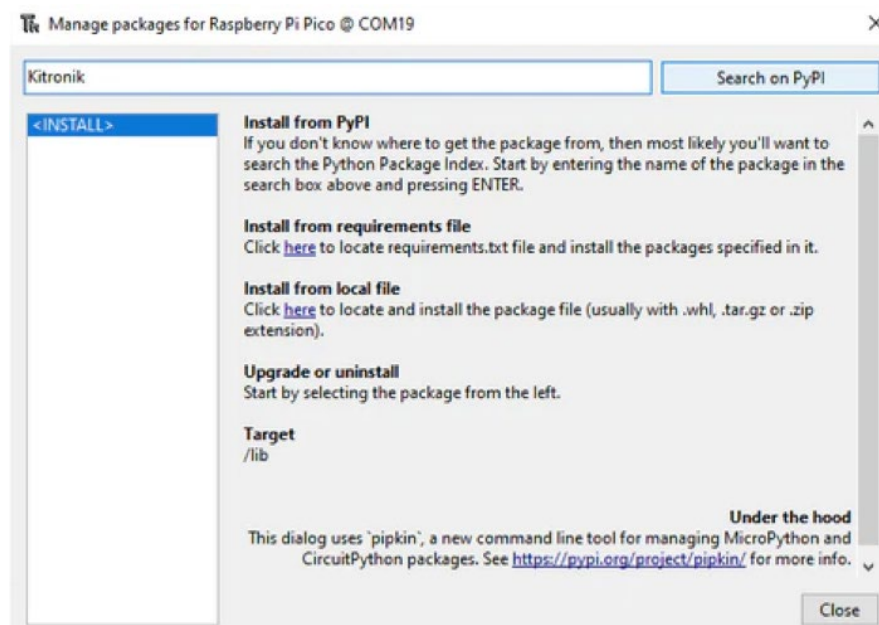
**4** Now we can install the ZIP96Pico library onto our Pico.



- 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.

# LESSON 5

## MAIN LESSON



### Curriculum mapping

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



### CREATE GAME CLASS: *GEM*

Now that we have a working Wall class, let's add some gems to our maze. In the Gem class we need to define a constructor. The Gem needs to know:

- its x and y position on the screen,
- the colour used for its LED,
- the screen object to draw the player on the ZIP96.



The gems are in the maze to be collected by the player. How will the program know whether a gem is collected or not? What method could be used, and what difference should a gem being collected or not make to the game setup?



So let's take these as input parameters in the Gem constructor and set them as variables stored inside the object. An additional variable the Gem class will need is collected so it knows whether the player has collected the gem, which we can set to False to start off with. We can again add a call to the draw function inside the constructor for our Gem class which we will write the code for next.



```
# Class to store our Gem functionality
class Gem():
    # Gem object constructor
    def __init__(self, x, y, colour, screen):
        self.x = x
        self.y = y
        self.colour = colour
        self.screen = screen
        self.collected = False
        # Draw the starting position
        self.draw()
```



### Curriculum mapping

Make decisions in programs, making use of arithmetic, logic and Boolean expressions and operators.

## LESSON 5

The draw function we are going to write for the Gem class is going to be like a combination of the Player.draw and Player.drawEmpty functions. When our Gem.collected variable is True we want to make the gem disappear, like in the drawEmpty function. However, when our Gem.collected variable is False we want to make the gem visible, like in the draw function. Let's use an if statement inside the Gem.draw function to show the gem's colour when it hasn't been collected but hide the gem when it has.



```
# Draw the current position on the screen
def draw(self):
    # Only draw as a Gem if it hasn't been collected
    if self.collected:
        self.screen.setLEDMatrix(self.x, self.y, screen.BLACK)
    else:
        self.screen.setLEDMatrix(self.x, self.y, self.colour)
```

In the same way we wanted to check when the player moved onto a Wall object, we want to check when the player moves onto a Gem object. We'll create the Gem.collison function in the exact same way, accepting an x and y value as inputs and checking them against the Gem coordinates to return if they match.

```
# Check if the given x and y are the same as our current position
def collision(self, x, y):
    return self.x == x and self.y == y
```



### Curriculum mapping

Learn how to handle strings, and simplify solutions by making use of lists and arrays. Apply iteration in program designs using loops.



# LESSON 5

## CREATE GAME GEM OBJECTS



To setup the gems for our maze we'll copy what we did for the walls. First make an array of where we want them to go. For the gems we want to use almost all the coordinates that aren't used by the walls. The exceptions to this are:

- the gaps in the left and right walls on the edge of the screen where the player can wrap from one side of the maze to the other,
- the three coordinates where the Enemy objects will start.

Above where we create our player object, let's create an array gemsXY of the x and y coordinates for each Gem object in our maze.

```
# List to store the coordinates of our gems
gemsXY = [(1, 1), (2, 1), (4, 1), (5, 1), (6, 1), (8, 1), (9, 1), (10, 1),
          (2, 2), (4, 2), (7, 2), (8, 2), (10, 2),
          (1, 3), (2, 3), (3, 3), (4, 3), (7, 3), (10, 3),
          (1, 4), (4, 4), (7, 4), (8, 4), (9, 4),
          (1, 5), (3, 5), (4, 5), (7, 5), (9, 5),
          (1, 6), (2, 6), (3, 6), (5, 6), (6, 6), (7, 6), (9, 6), (10, 6)]
```

With the coordinates for the gems setup we can now create an empty array gems to store all of the Gem objects. We will fill the gems array by looping through each x and y pair stored in gemsXY, using the coordinates to create a Gem object and append it to the end of the gems array.



```
# List to store our Gem objects
gems = []
# Loop to create Gem objects using the gemsXY coordinates
for gemXY in gemsXY:
    # Add the Gem objects to our gems list
    gems.append(Gem(gemXY[0], gemXY[1], gamer.Screen.WHITE, screen))
```

## UPDATE GAME FUNCTION: *PLAYER.MOVE*



Next, we again need access to the gems inside the Player class, so let's add the gems array as an input parameter in the Player constructor. On top of this the Player class needs to keep track of how many gems have been collected so we can create a foundGems variable inside the Player class that stores the number of gems found.

```
def __init__(self, x, y, colour, walls, gems, screen, screenWidth, screenHeight):
    self.x = x
    self.y = y
    self.colour = colour
    self.walls = walls
    self.gems = gems
    self.foundGems = 0
```

## LESSON 5

To check when we have found one of the gems, we'll add a loop to the Player.move function about the call to Player.draw, similar to the one we used when checking collisions with the walls. We'll use a for each loop to loop through the array of Gem objects we added to the Player constructor.



### Curriculum mapping

Created nested selection statements.

Inside the loop use gem to access the Gem object and check if there is a collision with the gem from the player object's new position. When there is a collision we should then check if the gem has already been collected by the player. If it hasn't been collected we should tell it the player has now collected it by setting the gem variable to True. Then increase the foundGems variable to show another Gem object has been found by the player.

```
# Check each Gem in our list
for gem in self.gems:
    # If the Player is colliding with a Gem
    if gem.collison(self.x, self.y):
        # If the Gem hasn't been collected
        if not gem.collected:
            # Collect the Gem
            gem.collected = True
            self.foundGems += 1
        # Cannot collide with more than one Gem so leave loop
        break

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

### TASK: TEST GEM CLASS

With our Gem class written we can now test our maze with the gems added.

**Note:** Don't forget to update the player object's inputs to include the gems array.

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



What happens when all of the Gem objects have been collected?





**LESSON**  
**5**

# CONCLUSION

## LESSON 05 CONCLUSION



In this lesson, you have:

- Created Gem class, constructor and methods
- Used for loop, if statement and Boolean logic to check for interaction between Player and Gem objects

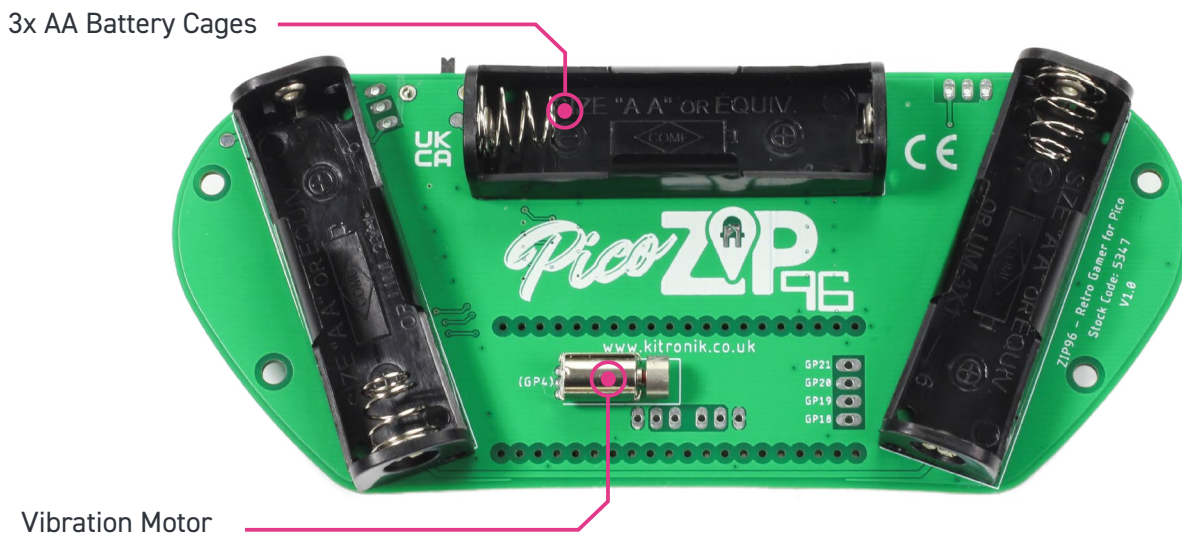
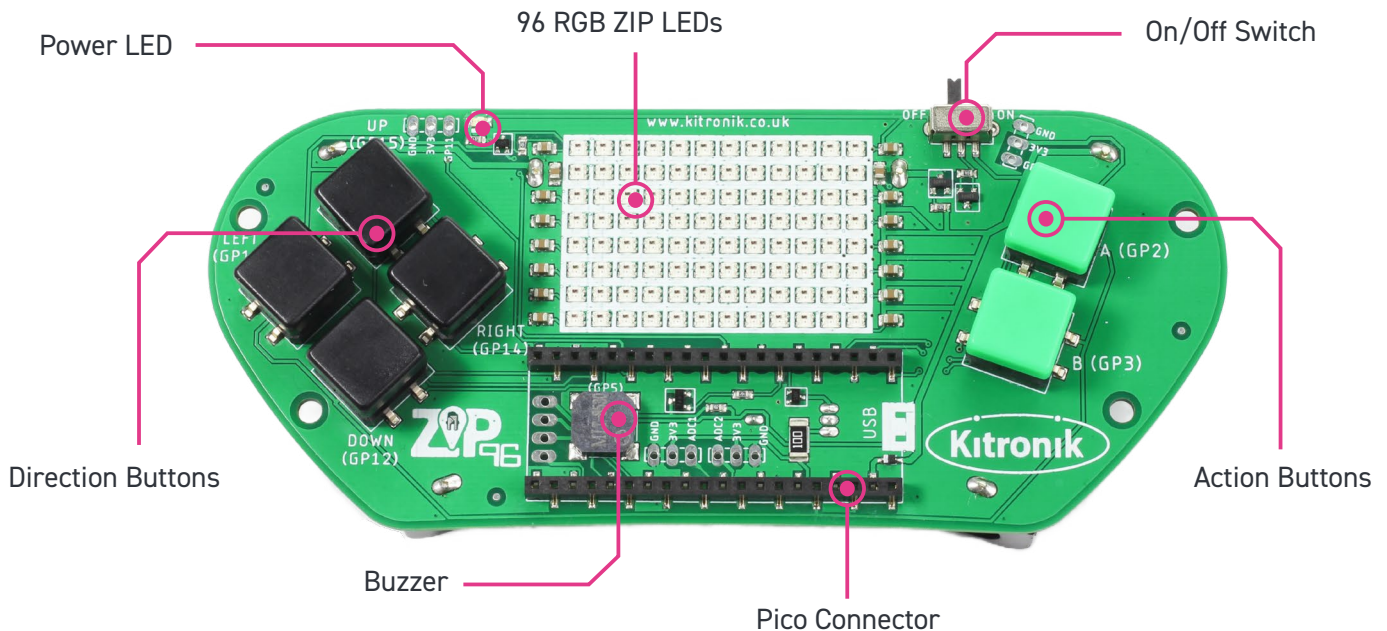
## FULL LESSON CODE

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.