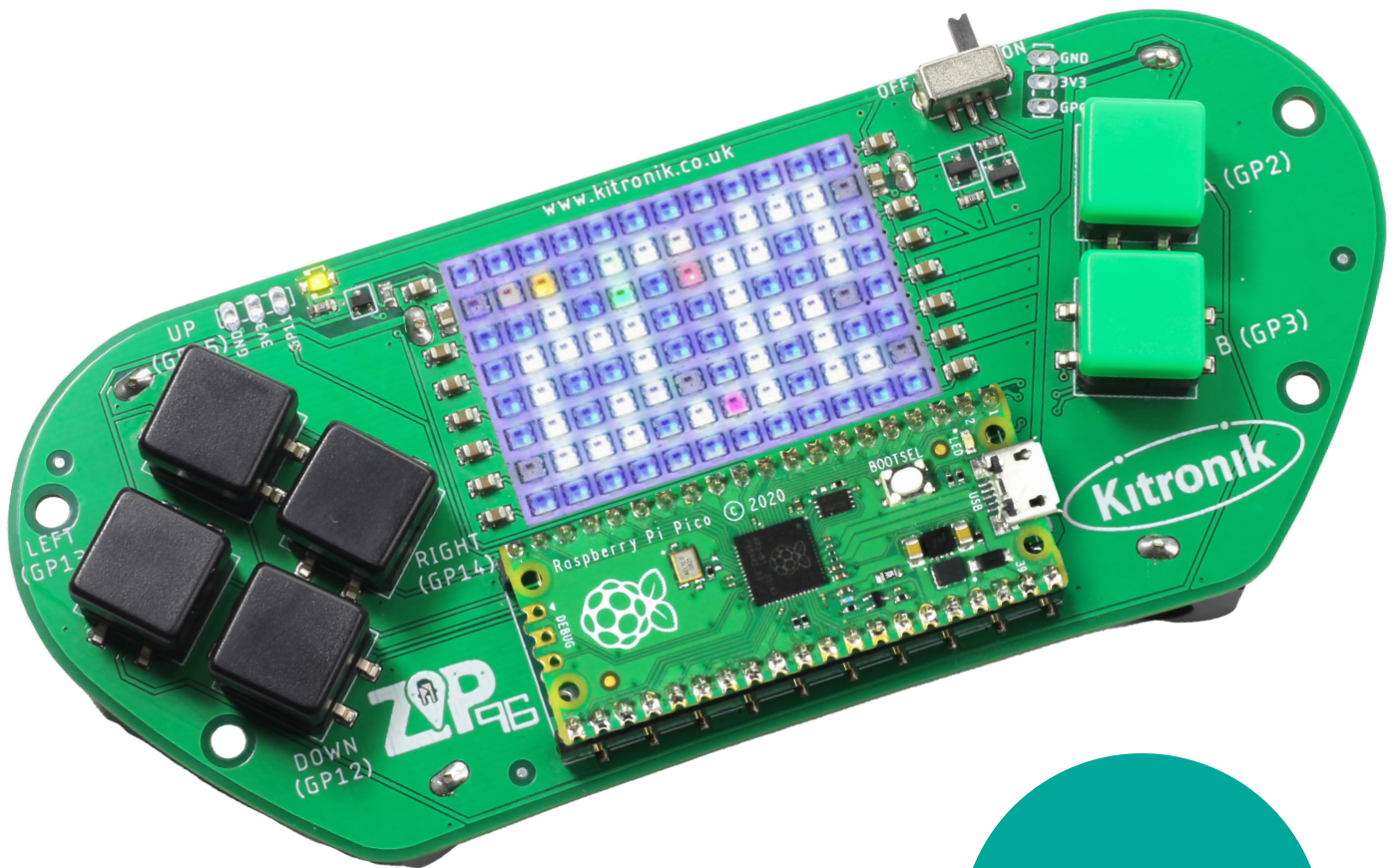


LESSON GUIDE TO THE PICO ZIP96



LESSON 7: THE A-MAZING GAME



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

14+

www.kitronik.co.uk

TEACH YOUR STUDENTS HOW TO CREATE GAMES WITH CODE!

LESSON 7

INTRODUCTION & SETUP



This is the seventh lesson in the 'A-mazing Game' series for Pico ZIP96. The game is now at a point where the player can move around the maze collecting all the gems, but there is no challenge involved – this lesson will see the first of the enemies introduced!



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 06.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.
- Describe the role of the CPU, and understand CPU components, the fetch-decode-execute cycle, and both primary and secondary storage.
- 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, COMPUTER SYSTEMS, CPU, MICROPROCESSORS, FETCH-DECODE-EXECUTE CYCLE, PRIMARY & SECONDARY MEMORY, INPUT & OUTPUT DEVICES, LOGIC CIRCUITS

LESSON 7

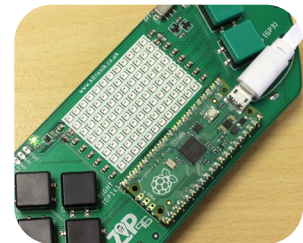
WHAT IS OUR A-MAZING GAME?



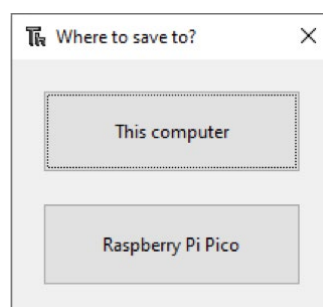
Our A-Mazing Game is a maze-based game where the player runs through a maze collecting gems. Once you have collected all the gems in the maze then you have won! But it won't be that easy, as there are enemies in the maze trying to catch you before you collect all of the gems. The three enemies each have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



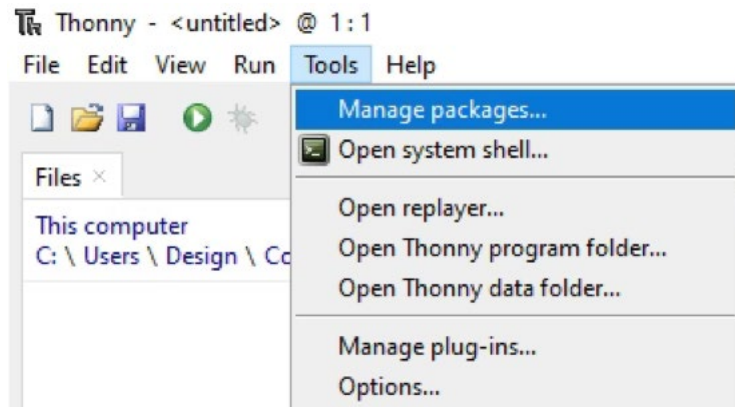
LESSON 7

SETUP CONTINUED

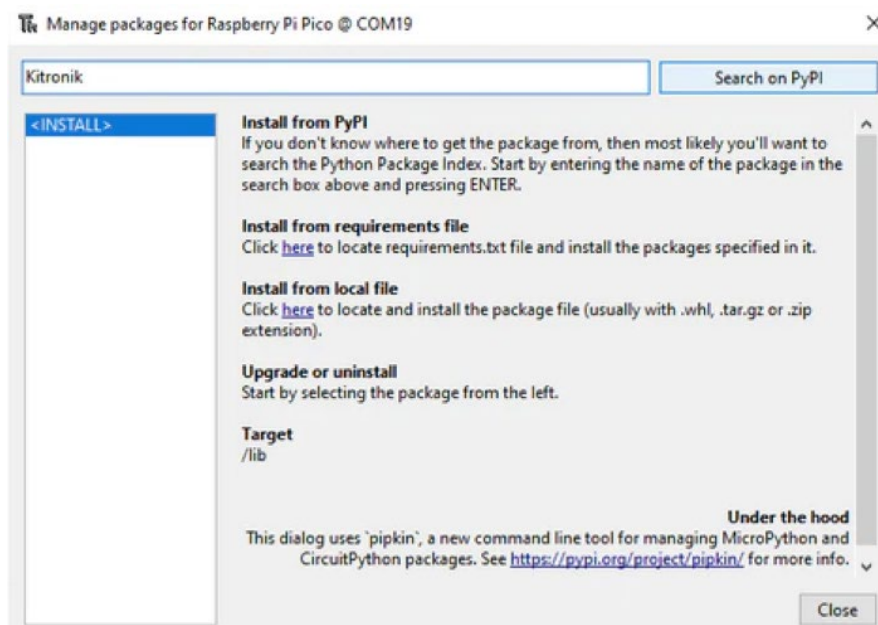
4 Now we can install the ZIP96Pico library onto our Pico.



- 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 7

MAIN LESSON



Curriculum mapping

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



CREATE GAME CLASS: *ENEMY*

With the core functionality of our maze game implemented, let's now add the enemies to chase the player around the maze. The first Enemy will move randomly and for this we'll use randint from the random library.

```
import _thread
from random import randint
```

In the Enemy class we need to define a constructor. The Enemy needs to know:

- its x and y position on the screen,
- the colour used for its LED,
- the walls used in the maze,
- the gems used in the maze,
- the screen object to draw the enemy on the ZIP96,
- the screenWidth and screenHeight to detect when the enemy moves off the screen.



So let's take these as input parameters in the Enemy constructor and set them as variables stored inside the object. We can again add a call to the draw function inside the constructor for our Enemy class which we will write the code for next.



The draw function is going to work in the same way as the Player.draw function does.

Class to store our Enemy functionality

```
class Enemy():
    # Enemy object constructor
    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.screen = screen
        self.screenWidth = screenWidth
        self.screenHeight = screenHeight
        # Draw the starting position
        self.draw()

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

LESSON 7



Curriculum mapping

Apply iteration in program designs using loops.

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



What needs to be different about how the enemy object interacts with the maze, when compared with how the player interacts?



Now the start of the drawEmpty function is the same as the Player.drawEmpty function, but we also need to handle the case where an Enemy object moves away from a Gem object. By default the Enemy.drawEmpty function would just remove the Gem object's LED from the screen. However, the Enemy objects do not collect the gems. To redraw a Gem object, we can loop through the gems, check for a collision and then call draw on the gem if we have collided with it. The Gem object will then only redraw itself to the screen if it hasn't been collected by the player.



Draw a blank space in the current position on the screen

```
def drawEmpty(self):
```

```
    self.screen.setLEDMatrix(self.x, self.y, self.screen.BLACK)
```

```
    # Check each Gem in our list
```

```
    for gem in self.gems:
```

```
        # If the Enemy is colliding with a Gem
```

```
        if gem.collision(self.x, self.y):
```

```
            # Redraw the Gem. Resets the LED to white if a gem is there
            gem.draw()
```

```
            break
```

To move the Enemy object around the screen we'll need to add a move function moveRandom. In this function, let's first reset the screen at the Enemy object's current position by calling drawEmpty on it. For our random move we can setup three variables x, y and random with random being set to a random integer between zero and three using the randint function we imported earlier. Depending on the value of random we'll move the Enemy object up, down, left or right.



Then update the objects position by adding the random x and y values to the object's x and y values before calling draw to update the ZIP96 screen with its new position.


```
# Update the current position randomly
def moveRandom(self):
    # Reset the current position on the screen
    self.drawEmpty()

    # Set the move parameters randomly
    x = 0
    y = 0
    random = randint(0, 3)
    if random == 0: x = 1
    elif random == 1: x = -1
    elif random == 2: x = 1
    else: y = -1

    # 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()
```



Curriculum mapping

Learn how to handle strings, and simplify solutions by making use of lists and arrays.

TASK: TEST ENEMY CLASS



Let's now test our Enemy class by creating an array of Enemy objects enemies underneath where we create the player object. We can set the start position for this Enemy to be in the middle of the screen where we have an empty space and its colour to red.

```
# List to store our Enemy objects
enemies = [Enemy(6, 2, gamer.Screen.RED, walls, gems, screen, screenWidth, screenHeight)]
```

To move the Enemy we need to add a call to moveRandom in our game loop. Let's create another loop to move the enemies above screen.show in the game loop. Inside this loop if the index i is equal to zero, call moveRandom on that Enemy object.

```
# Check each Enemy in our list
for i in range(len(enemies)):
    # First Enemy moves randomly
    if i == 0: enemies[i].moveRandom()

# Show the updates on the screen
screen.show()
```

LESSON 7



While testing, try to figure out what functionality is missing from the `Enemy.moveRandom` function.



UPDATE GAME FUNCTION: *ENEMY.MOVERANDOM*



As the Enemy moves around the screen, just like the Player, we need to perform checks on it to stop it moving off the screen or moving onto walls. This is why we included `screenWidth`, `screenHeight` and `walls` as parameters in the Enemy constructor.

To do this we'll add the exact same code we used in `Player.move` underneath where we update the object's position in `Enemy.moveRandom`.

```
self.y += y

# Check the new x position is valid (not off the edge of the screen)
if (self.x < 0): self.x = self.screenWidth - 1
if (self.x >= self.screenWidth): self.x = 0

# Check the new y position is valid (not off the edge of the screen)
if (self.y < 0): self.y = self.screenHeight - 1
if (self.y >= self.screenHeight): self.y = 0

# Check each Wall in our list
for wall in self.walls:
    # If the Enemy is colliding with a Wall
    if wall.collison(self.x, self.y):
        # Undo the new position using the x and y parameters
        self.x -= x
        self.y -= y
        # Cannot collide with more than one Wall so leave loop
        break
```

TASK: TEST UPDATED *ENEMY.MOVERANDOM*



With the collision checking added to the Enemy object, let's test it again to check if the enemies work without moving onto the walls or off the screen.

LESSON
7

CONCLUSION

LESSON 07 CONCLUSION



In this lesson, you have:

- Created Enemy class, constructor and methods
- Used for loop, if statement and Boolean logic to check for interaction between Enemy and Wall objects
- Used random library to include random behaviours in the maze game

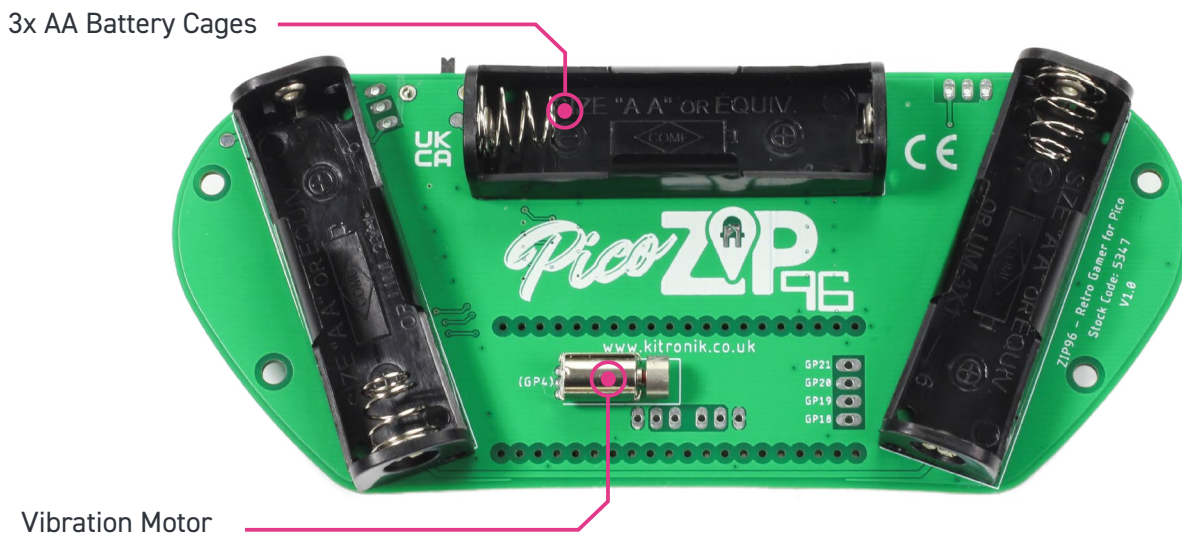
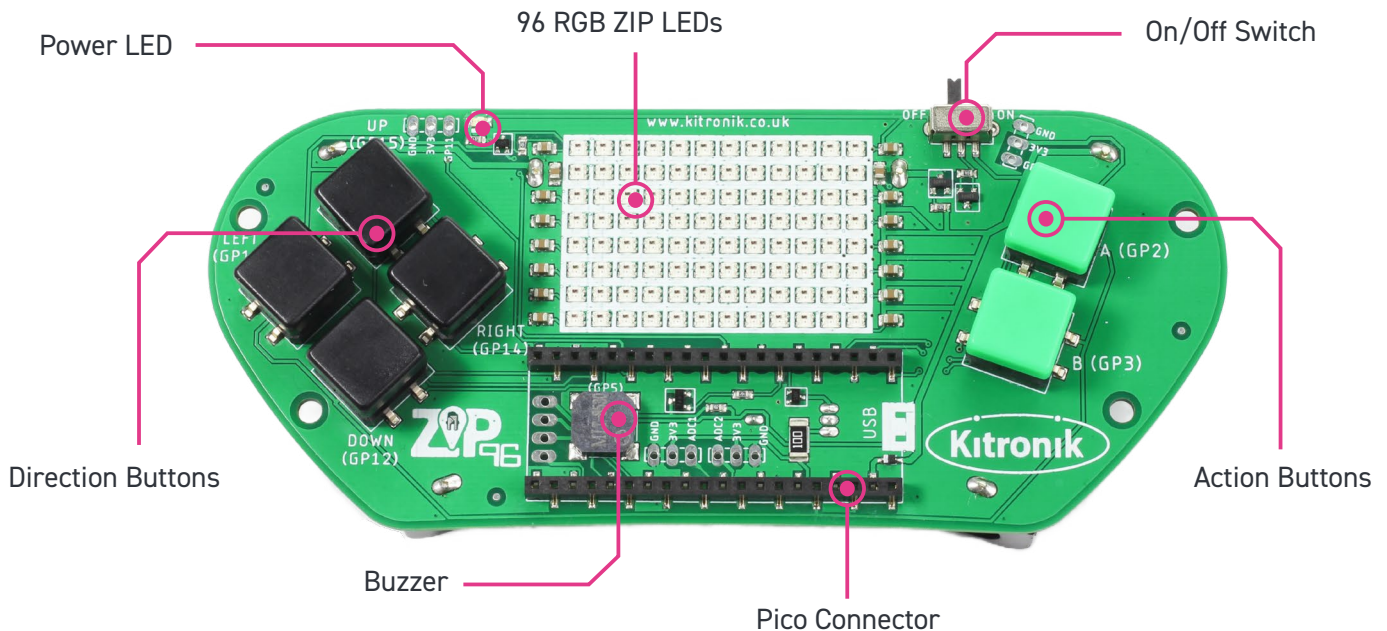
FULL LESSON CODE

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

E: 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. 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.