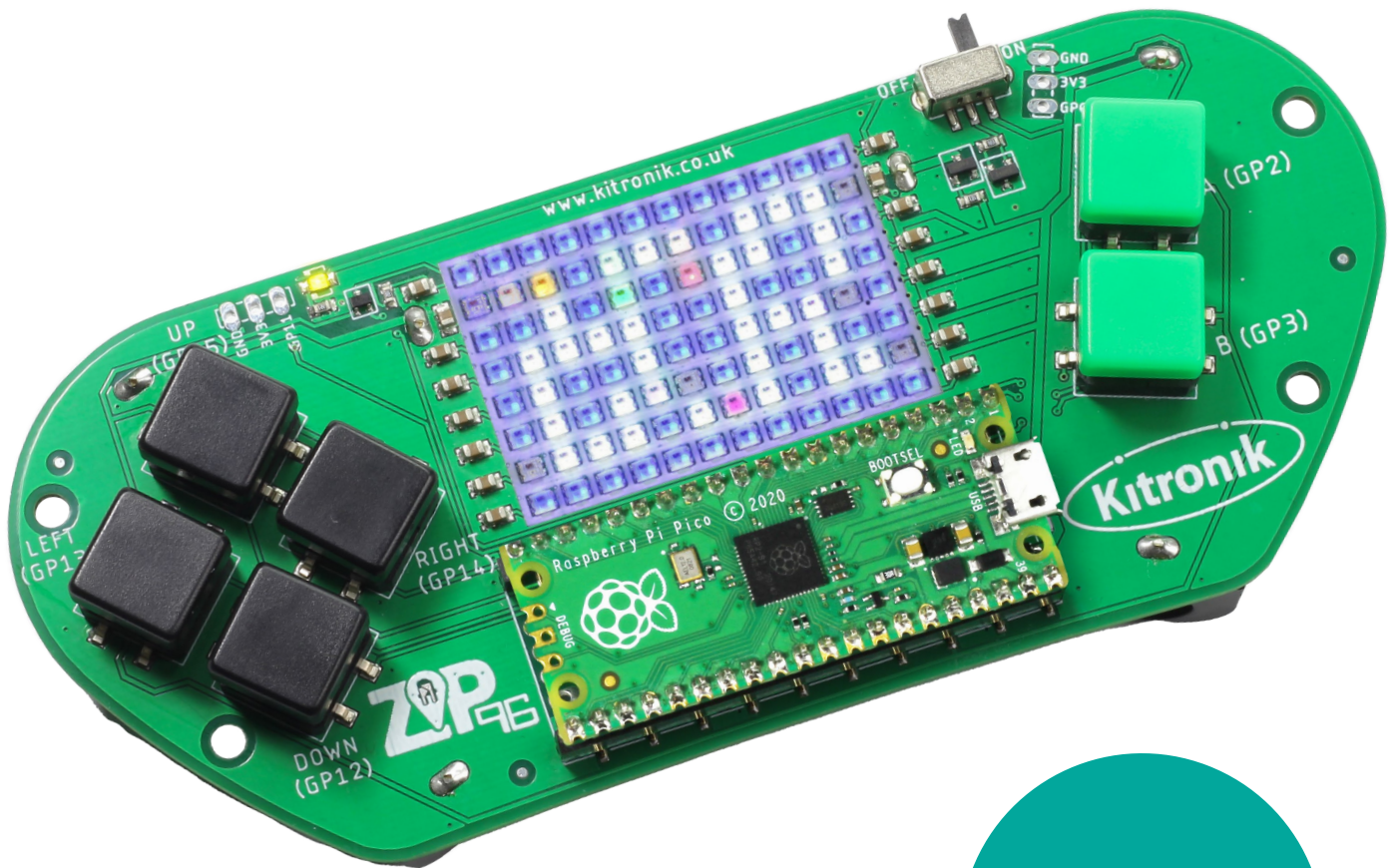


LESSON GUIDE TO THE PICO ZIP96



LESSON 8: 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 8

INTRODUCTION & SETUP



This is the eighth lesson in the 'A-mazing Game' series for Pico ZIP96. The previous lesson added an enemy to the maze, this lesson the code is written to give enemies the ability to take lives away from the player.



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 07.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 8

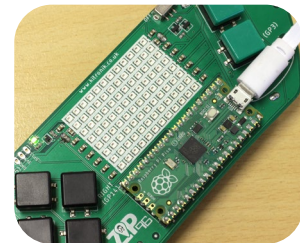
WHAT IS OUR A-MAZING GAME?



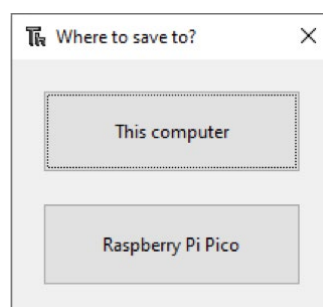
Our A-Mazing Game is a maze-based game where the player runs through a maze collecting gems. Once you have collected all the gems in the maze then you have won! But it won't be that easy, as there are enemies in the maze trying to catch you before you collect all of the gems. The three enemies each have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



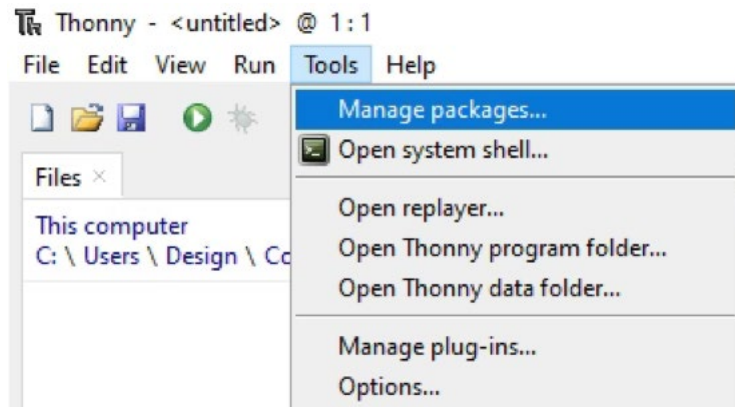
LESSON 8

SETUP CONTINUED

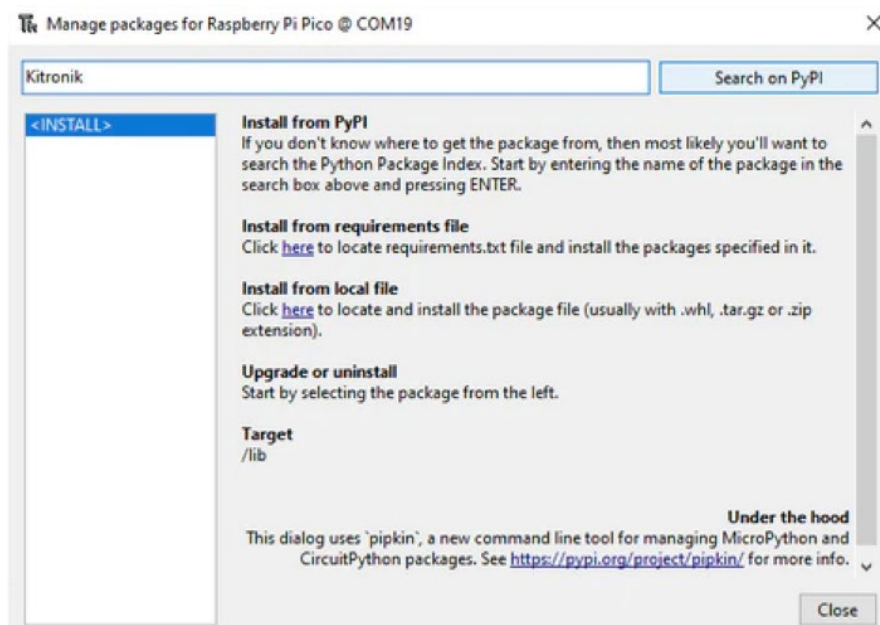
4 Now we can install the ZIP96Pico library onto our Pico.



- 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



Curriculum mapping

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



CREATE GAME FUNCTION: *PLAYER.COLLISION*

With the basic functions for our Enemy done we now want the Enemy to remove a life whenever it catches the player. In the same way we wanted to check when the player moved onto a Wall object, we want to check when an Enemy moves onto the player. We'll create the Player.collison function in the exact same way, accepting an x and y value as inputs and checking them against the Player 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
```

UPDATE GAME FUNCTION: *ENEMY.MOVERANDOM*



To check when an Enemy object moves onto the player we need have access to the player from inside the Enemy class. We can do this again by adding it as a parameter in the Enemy constructor and saving the player inside the object. We also want an extra variable hitPlayer that we set to True when an Enemy object collides with the player.

```
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.player = player
    self.hitPlayer = False

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

At the top of the Enemy.moveRandom function let's set hitPlayer to False. We should set it to False every time the Enemy moves because if we don't reset it, and we had collided with the player on the last move, then it would still be True telling us the Enemy has collided with the player again even when it hasn't.



```
self.drawEmpty()
# Reset hitPlayer
self.hitPlayer = False
```

**Curriculum mapping**

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

As we have done before, to check for a collision between the player and Enemy object we'll call the collision function. Let's add this at the end of moveRandom just above the call to draw. When there is a collision we want to then set hitPlayer to be True.

```
# If the Enemy is colliding with the Player
if self.player.collision(self.x, self.y):
    # Set hitPlayer
    self.hitPlayer = True

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

CREATE GAME FUNCTIONS: *PLAYER.RESET* AND *ENEMY.RESET*

When we detect that the player has been caught by an Enemy object we are going to reset their positions to where they started. To do this we can add the same reset function to both the Player and Enemy classes. The reset function will take an x and y value as input parameters. Inside the function, first call drawEmpty to clear the screen of their current position. Then set the position to the x and y input values. Finally, redraw the object in its new position using draw.

```
# Reset the Player position and redraw
def reset(self, x, y):
    self.drawEmpty()
    self.x = x
    self.y = y
    self.draw()

# Reset the Enemy position and redraw
def reset(self, x, y):
    self.drawEmpty()
    self.x = x
    self.y = y
    self.draw()
```


CREATE GAME FUNCTION: *LIVESUPDATE*



For resetting the game when the player is caught, let's create a new function `livesUpdate`. Inside `livesUpdate` we need some of the global variables used in the maze which we'll add here. The start of `livesUpdate` should turn the screen red, using `fill`, for a few second to show the player has been caught and lost a life.

```
# Update the user and reset the position
def livesUpdate():
    global lives, screen, player, enemies, walls, gems

    screen.fill(screen.RED)
    screen.show()
    sleep(5)
```



Curriculum mapping

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

Then, when the player still has lives left, clear the screen again using `fill` with the colour set to black. Next we need to reset the positions of the player and enemies back to where they started in the maze. Finally, redraw the walls and gems onto the screen and show the changes on the ZIP96.



```
# If the game isn't over
if lives > 0:
    # Reset the screen
    screen.fill(screen.BLACK)

    # Reset the Player position
    player.reset(0, 1)
    # Reset the Enemy position
    enemies[0].reset(6, 2)

    # Redraw the walls
    for wall in walls:
        wall.draw()

    # Redraw the gems
    for gem in gems:
        gem.draw()

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

TASK: TEST *LIVESUPDATE* GAME FUNCTION



Inside of the game loop where we move the enemies let's add an if statement to check if we have caught the player by checking if `Enemy.hitPlayer` is `True`. When an Enemy has caught the player we need to remove one life from the `lives` variable. Then we can call the `livesUpdate` function to reset the maze.

Try testing this function by repeatedly crossing paths with the `Enemy` object and see if the game stops when your three lives run out.

```
if i == 0: enemies[i].moveRandom()
# Check if the Enemy has collided with our Player
if enemies[i].hitPlayer:
    # Remove a life
    lives -= 1
    # Update the user and reset the positions
    livesUpdate()
    break
```


LESSON
8

CONCLUSION

LESSON 08 CONCLUSION



In this lesson, you have:

- Used for loop, if statement and Boolean logic to check for interaction between Player and Enemy objects
- Updated the user on their progress using the Pico ZIP96 screen

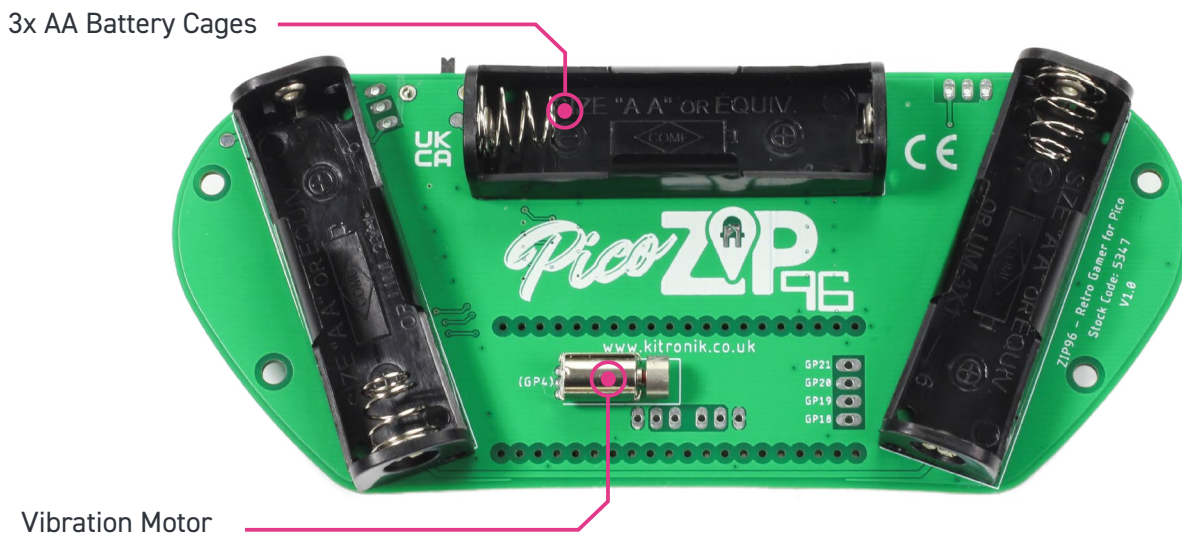
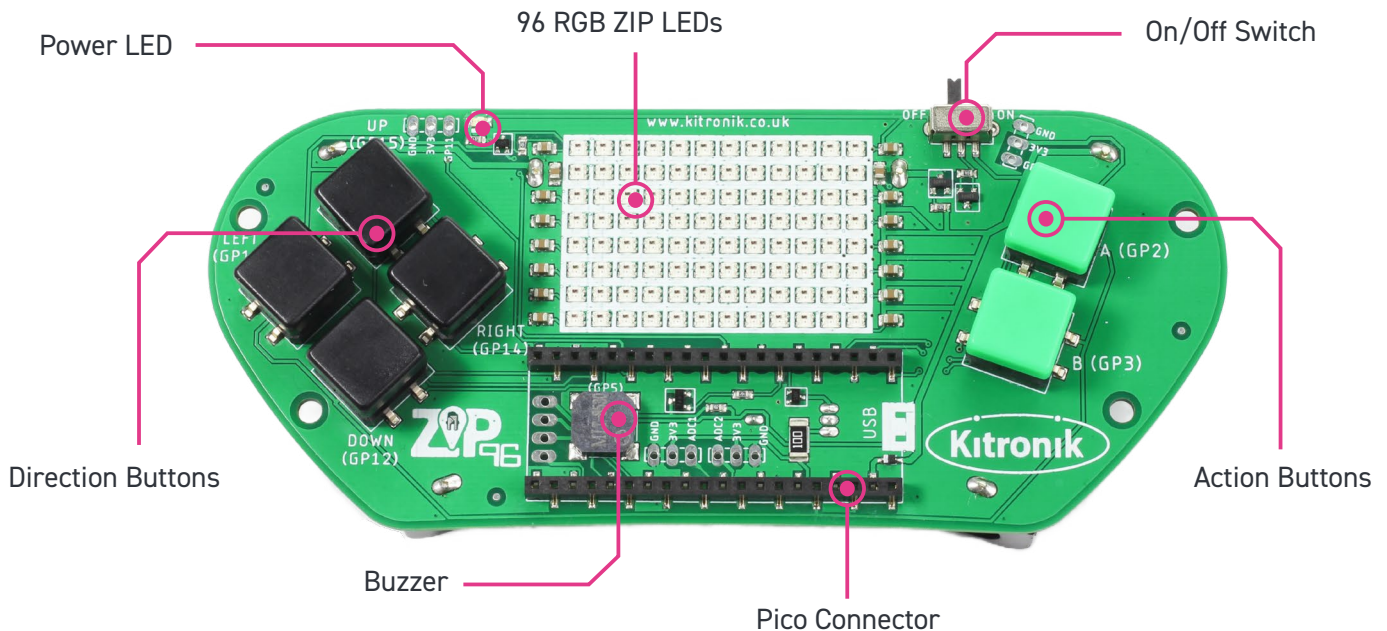
FULL LESSON CODE

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.