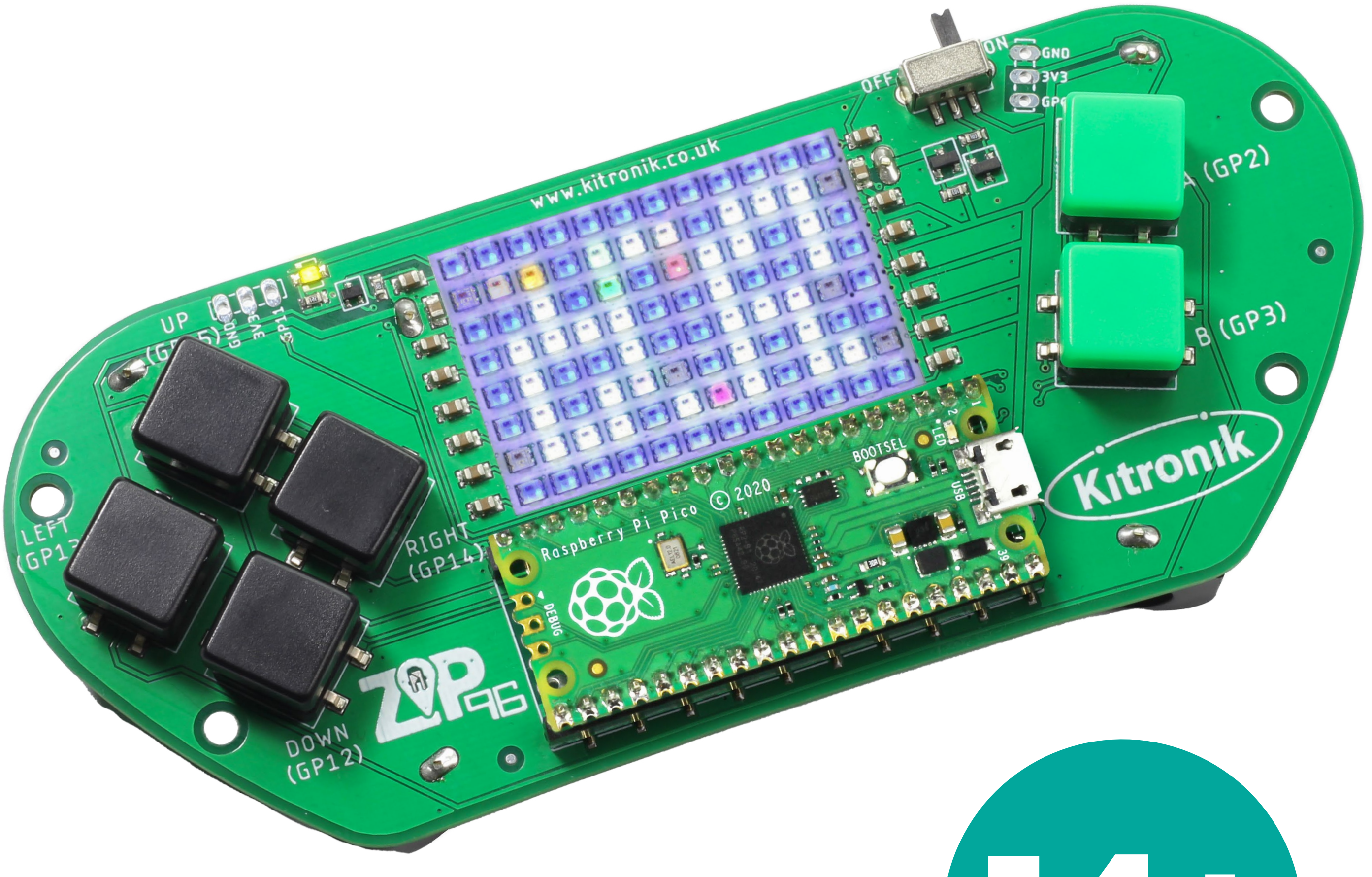
LESSON GUIDE TO THE PICO ZIP96



14+

LESSON 6:
THE A-MAZING GAME

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

www.kitronik.co.uk

TEACH YOUR STUDENTS HOW TO CREATE GAMES WITH CODE!

LESSON 6

INTRODUCTION & SETUP

This is the sixth lesson in the 'A-mazing Game' series for Pico ZIP96. To improve the running of the game and its responsiveness to user inputs, the main game code and the Pico ZIP96 input checks will be split into separate threads.

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 05.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

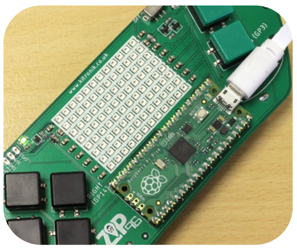
LESSON 6

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.

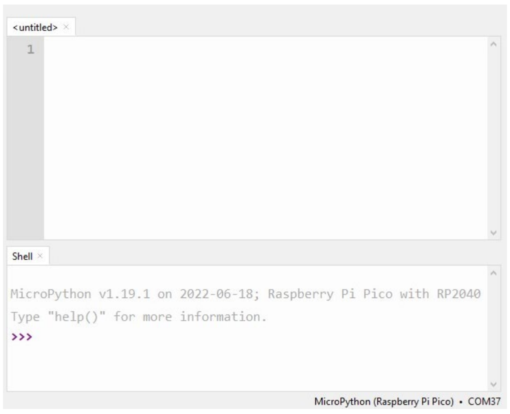
SETUP

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.

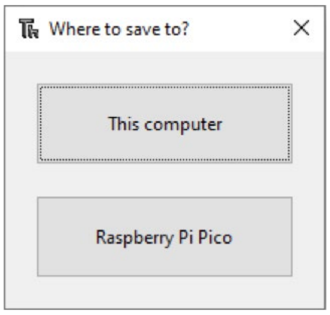
- If the device does not load automatically, try pressing the STOP icon at the top of the screen.
- If the shell does not load automatically, turn it on by checking **View** > **Shell**.



```
<untitled>
1

Shell
MicroPython v1.19.1 on 2022-06-18; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>>
                                MicroPython (Raspberry Pi Pico) • COM37
```

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**.



Where to save to?

This computer

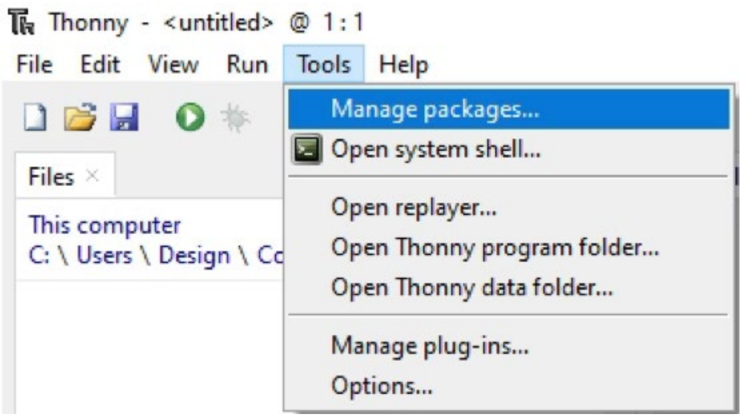
Raspberry Pi Pico

LESSON 6

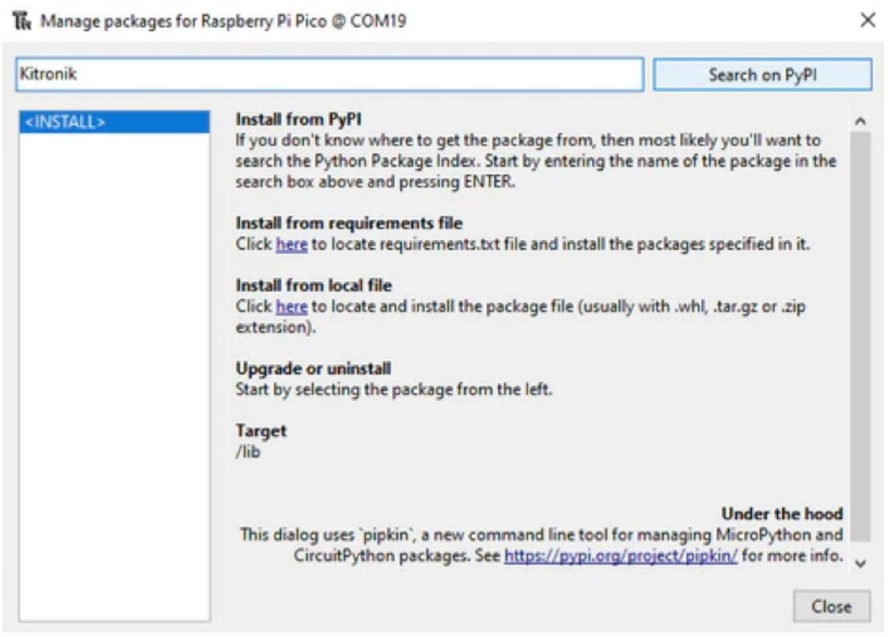
SETUP CONTINUED

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.



Install from PyPI
If you don't know where to get the package from, then most likely you'll want to search the Python Package Index. Start by entering the name of the package in the search box above and pressing ENTER.

Install from requirements file
Click here to locate requirements.txt file and install the packages specified in it.

Install from local file
Click here to locate and install the package file (usually with .whl, .tar.gz or .zip extension).

Upgrade or uninstall
Start by selecting the package from the left.

Target
/lib

Under the hood
This dialog uses `pipkin`, a new command line tool for managing MicroPython and CircuitPython packages. See https://pypi.org/project/pipkin/ for more info.

- 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.

Kitronik

LESSON 6

MAIN LESSON

Curriculum mapping
Describe the role of the CPU, and understand CPU components

SEPARATE ZIP96 INPUT AND GAME LOOP

Now that we have the player, walls and gems in our maze game we should separate the logic for our ZIP96 input and our game logic. This will allow our game to run smoother by increasing how responsive our ZIP96 inputs are, as up to now the buttons were only being checked for presses twice a second.

Students take note
To have our ZIP96 input code running separately from our game loop we need to use threads. A thread is basically just a process that runs on a computer processor, and because the Raspberry Pi Pico has a dual-core processor, this means two threads can run together. In our game so far, we have just been using one thread which has meant that each line of code is executed one after the other. However, when using two threads we can have different sections of code being executed at the same time.

To add the ability to use threads in our game let's import the _thread library underneath the line where we import sleep.

```
from utime import sleep
import _thread
```

Next, underneath the line where we setup the moveY variable let's create a lives variable to store how many lives the player has before they lose the game.

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

```
moveY = 0
lives = 3
```

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

10 To run the ZIP96 input in another thread we need to create a new function gamerInput above the game loop which contains all of our input code. Then we are going to use the ZIP96 gamer, moveX and moveY so we add them as global variables accessed in our function. Next, to make using the ZIP96 buzzer and vibration motor easier, let's create two new variables buzzer and haptic, setting them to the gamer.Buzzer and gamer.Vibrate.

```
# Get input from the ZIP96Pico Gamer in a seperate thread
def gamerInput(gamer, lives):
      global moveX, moveY

      buzzer = gamer.Buzzer
      haptic = gamer.Vibrate
```

Curriculum mapping
Apply iteration in program designs using loops.
Make decisions in programs, making use of arithmetic, logic and Boolean expressions and operators.

11 Inside of the gamerInput function, a loop needs to be used to run the input code.

? What type of loop should be used, and what condition should be checked to keep it running?
(**Hint:** It should make use of the new 'lives' variable.)

12 We need to add a while loop with the condition being that it continues while the player has more than zero lives. Inside of this loop we can then move the four if statements that check for when a ZIP96 button is pressed. Inside of each if statement let's add a noise and vibration that is played whenever a ZIP96 button is pressed. On the buzzer we can call playTone and set the frequency of the tone played. On the haptic motor we can call vibrate. At the end of each if statement then sleep for 100 milliseconds to give the buzzer and haptic motor some time to play.

```
# Start game loop
while lives > 0:
      # When up pressed, change player y position by -1
      if (gamer.Up.pressed()):
          buzzer.playTone(1000)
          haptic.vibrate()
          moveX = 0
          moveY = -1
          sleep(0.1)
```

```
# When down pressed, change player y position by 1
if (gamer.Down.pressed()):
    buzzer.playTone(1000)
    haptic.vibrate()
    moveX = 0
    moveY = 1
    sleep(0.1)

# When left pressed, change player x position by -1
if (gamer.Left.pressed()):
    buzzer.playTone(1000)
    haptic.vibrate()
    moveX = -1
    moveY = 0
    sleep(0.1)
```

13

Finally, after doing the same when the right button is pressed, set the buzzer and haptic to stop making any noise or vibrating. We can do this on every loop and not just inside the if statements because it reduces the code, and does not change our games functionality.

```
# When right pressed, change player x position by 1
if (gamer.Right.pressed()):
    buzzer.playTone(1000)
    haptic.vibrate()
    moveX = 1
    moveY = 0
    sleep(0.1)

buzzer.stopTone()
haptic.stop()
```

TASK: TEST ZIP96 INPUT THREAD

14

Now we can start the gamerInput function in a new thread just above the start of our game loop. Try running the updated maze game and check that your new thread is working.

```
# Start ZIP96Pico Gamer input logic in a seperate thread
_thread.start_new_thread(gamerInput, (gamer, lives))
```

UPDATE GAME LOOP

15

With our ZIP96 input handled in a new thread, let's update the game loop with some new functionality. First we should add the lives condition to the while loop like we did inside of gamerInput. We want to keep the moveDelay the same, so that the player isn't moving around the maze too fast. The call to Player.move will also stay the same but now with the moveX and moveY values changed by our gamerInput thread.

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

Since we added the Gem functionality in the last lesson, let's check when the player has found all of the gems. To do this we can compare the value of the Player.foundGems variable to the length of the gems array. When they are the same, then we know the player has collected every Gem object. When this happens, we'll just fill the screen with green LEDs and set lives to be zero so the game loops end.

```
# Start game loop
while lives > 0:

    # Wait for moveDelay before moving
    sleep(moveDelay)

    # Update the player position using the move values
    player.move(moveX, moveY)
    # Check if the player has found all the gems
    if player.foundGems == len(gems):
        # Set the screen to green
        screen.fill(screen.GREEN)
        screen.show()
        # Set lives to zero to end the game loops
        lives = 0
        break

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

TASK: TEST WINNING THE MAZE

With our game loop checking if the player has won the game, try collecting all of the gems inside the maze to see that you can win game!

LESSON 6

CONCLUSION

LESSON 06 CONCLUSION

In this lesson, you have:

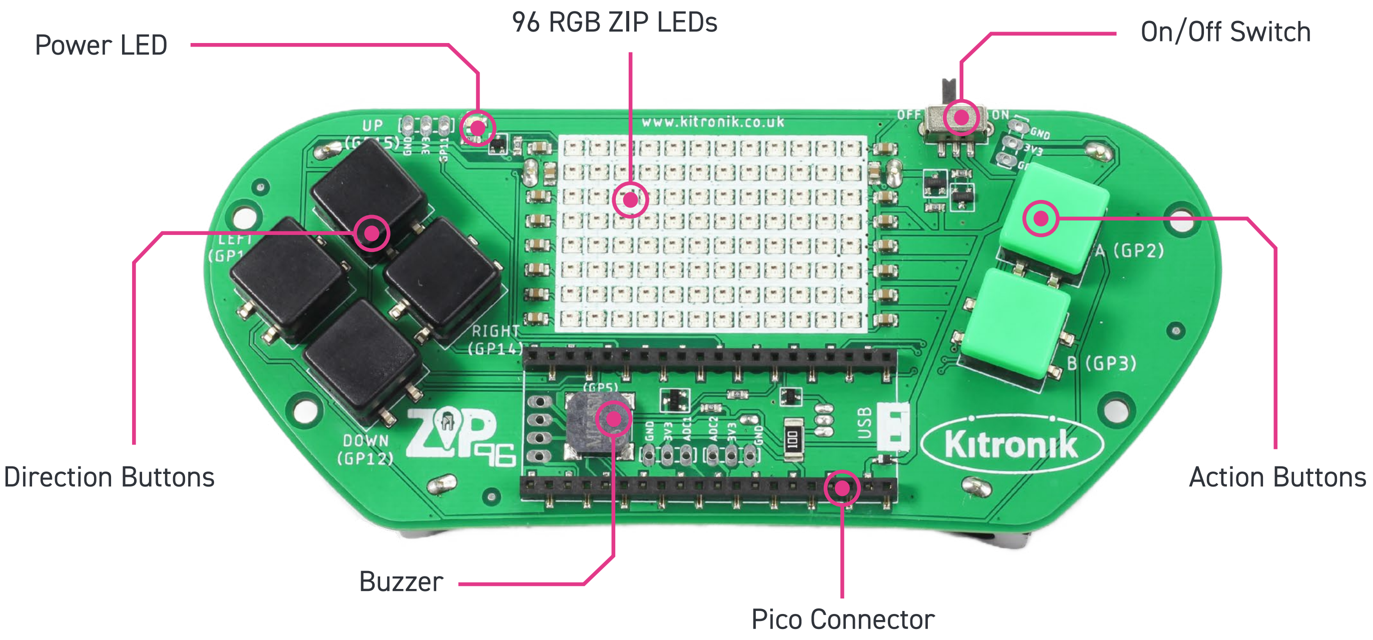
- Learnt about CPU cores
- Created a second thread
- Separated user input and game logic using parallelism
- Added win conditions to the game

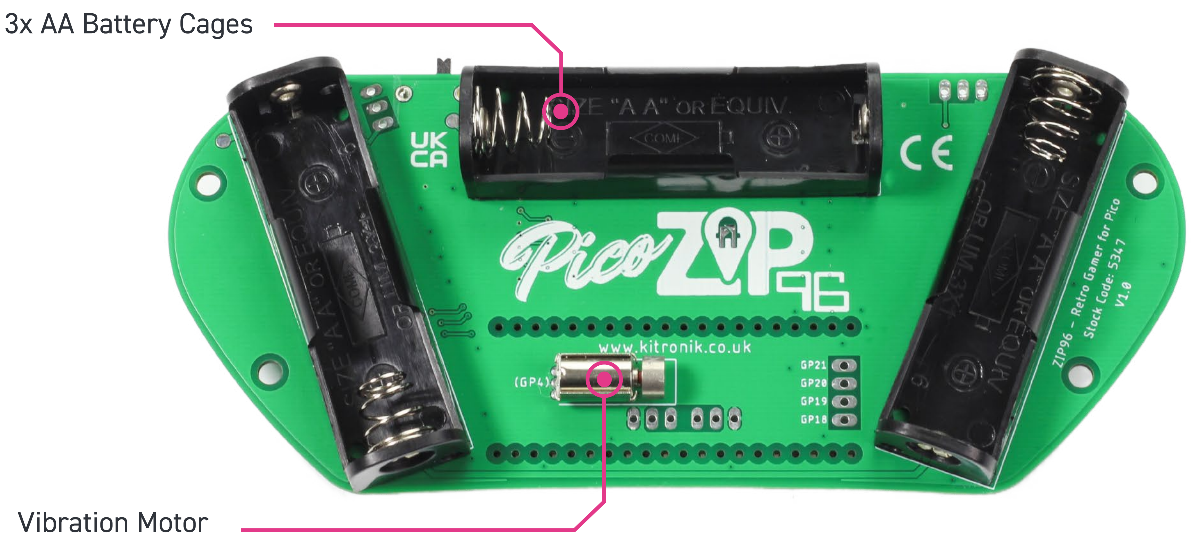
FULL LESSON CODE

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

kitronik.co.uk/youtube

kitronik.co.uk/facebook

kitronik.co.uk/instagram

Designed & manufactured in the UK by Kitronik

RoHS