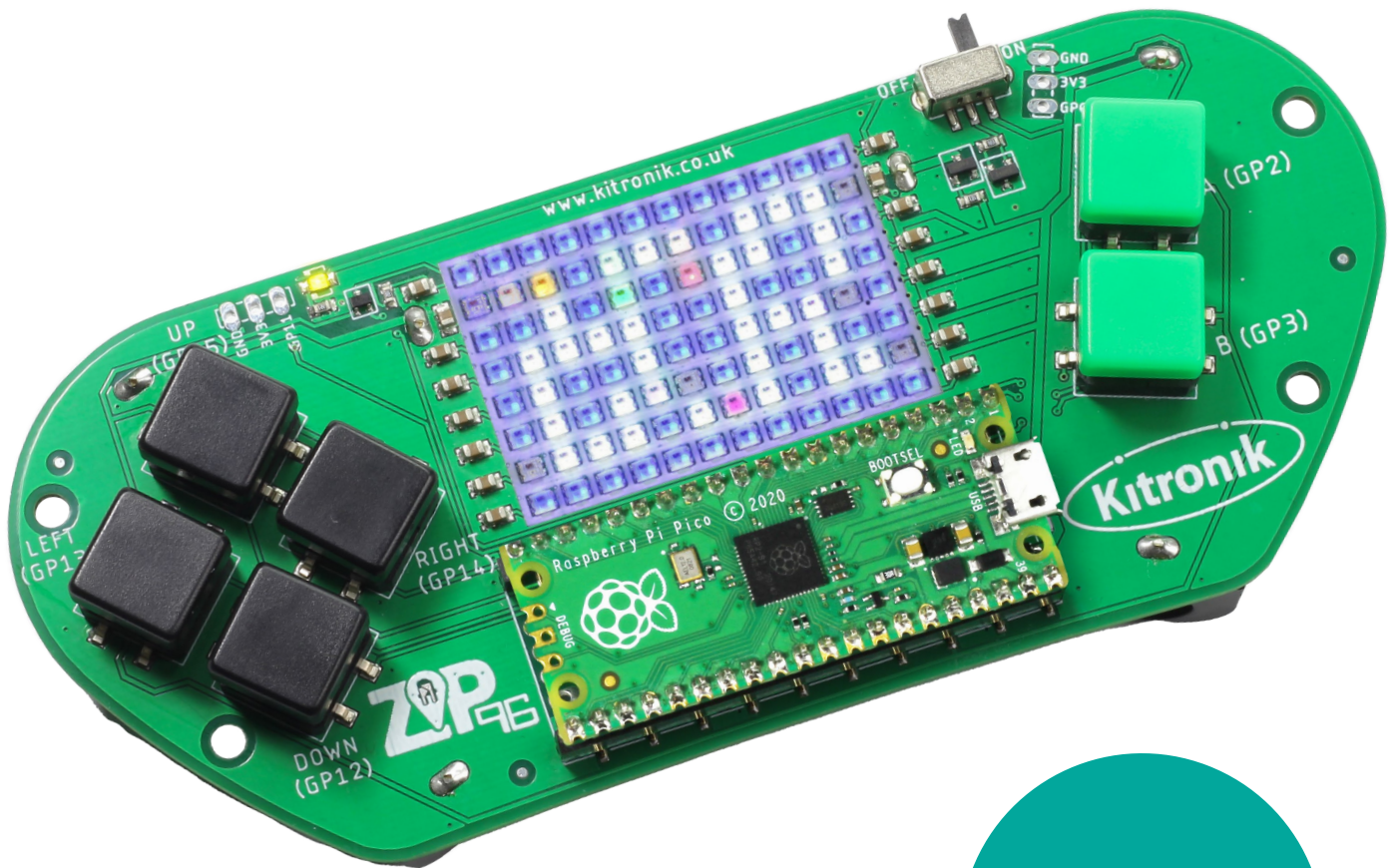


LESSON GUIDE TO THE PICO ZIP96



LESSON 9: 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!

INTRODUCTION & SETUP



This is the ninth lesson in the 'A-mazing Game' series for Pico ZIP96. With the first, randomly moving enemy now providing some extra challenge for the player, this lesson adds a second enemy with the ability to move towards 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 08.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 and solve them using algorithms. Explore different searching and sorting algorithms.
- 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. Create 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, ALGORITHMS, DECOMPOSITION, ABSTRACTION, DESIGN METHODS, TRACE TABLES, SEARCHING AND SORTING ALGORITHMS

LESSON 9

WHAT IS OUR A-MAZING GAME?

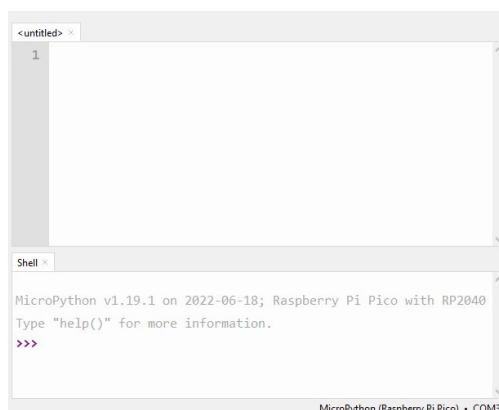


Our A-Mazing Game is a maze-based game where the player runs through a maze collecting gems. Once you have collected all the gems in the maze then you have won! But it won't be that easy, as there are enemies in the maze trying to catch you before you collect all of the gems. The three enemies each have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

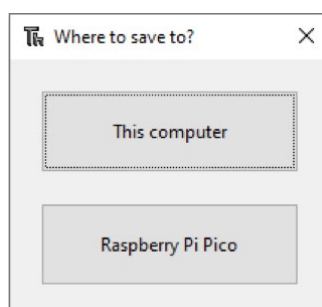
SETUP

- 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**.



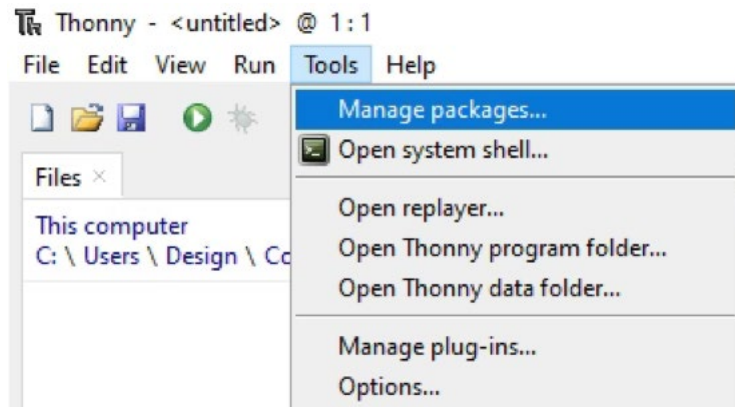
LESSON 9

SETUP CONTINUED

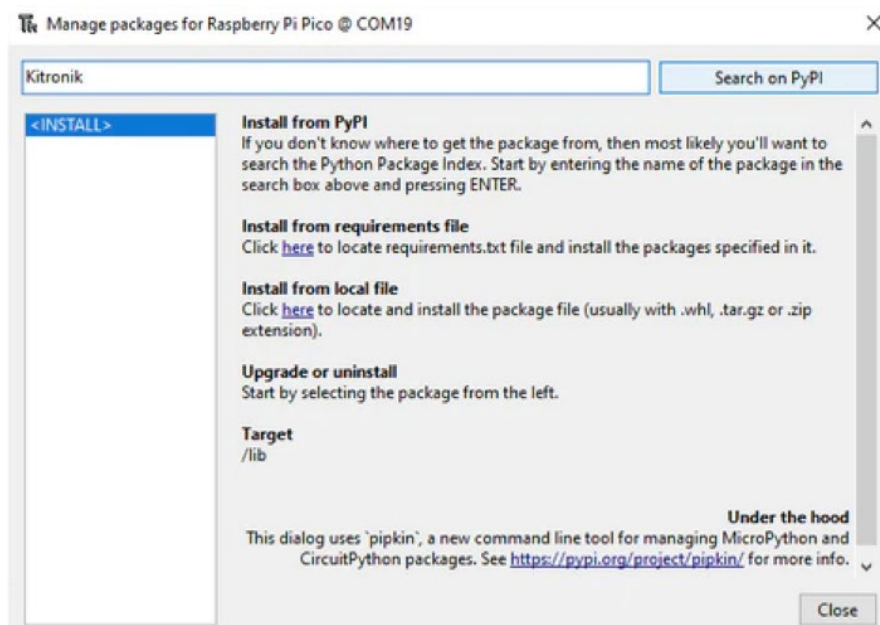
4 Now we can install the ZIP96Pico library onto our Pico.



- 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.

Decompose problems and solve them using algorithms. Explore different searching and sorting algorithms.



CREATE GAME FUNCTION: *ENEMY.MOVENORMAL*

With our randomly moving Enemy now working, we can create a new move function in the Enemy class to have the Enemy move towards the player. The start of the moveNormal function is the same as the moveRandom function. We want to reset the current position on the screen and set hitPlayer to False.

```
# Update the current position towards Player x, y
def moveNormal(self):
    # Reset the current position on the screen
    self.drawEmpty()
    # Reset hitPlayer
    self.hitPlayer = False
```



Curriculum mapping

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

Next, we need to figure out where the player is in the maze in comparison to the Enemy object. From this we can determine which direction is best to move in.



Given the x, y coordinates for both the player and the Enemy, what algorithm can you come up with to work out which direction the Enemy should move to head towards the player?

If the player position is less than the Enemy object's on the x axis then we want to decrease the Enemy x position. Otherwise, we want to increase the Enemy x position. We'll also do the same on the y axis.



```
# If Player x is less, decrease x
if self.player.x < self.x: x = -1
# Otherwise, increase x
else: x = 1

# If Player y is less, decrease y
if self.player.y < self.y: y = -1
# Otherwise, increase y
else: y = 1
```

Then let's determine whether there is a bigger gap between the player and Enemy on the x axis or the y axis. We can do this by getting the absolute difference between their x positions and y positions then comparing them. When the x axis gap is bigger, we'll have the Enemy move along the x axis by only updating its x position.



```
# Is the distance from the Player bigger in the x axis or y axis
changeX = abs(self.player.x - self.x) > abs(self.player.y - self.y)

# Try change in the direction with the biggest distance
if changeX: self.x += x
else: self.y += y
```



Curriculum mapping

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.

With the Enemy object's position updated we need to do the standard check whether they have moved into one of the Wall objects. The for loop and if statement is the same as usual but there need to be some differences in what we put inside them. First, we only need to undo the move on the axis that we changed. Then we'll invert the value of changeX to be used later. Also, the variable collision which we set to False at the start of the loop, needs to be set to True if we have a collision, again to be used later.



```
collision = False

# 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 in the direction we changed
        if changeX: self.x -= x
        else: self.y -= y
        # Invert changeX
        changeX = not changeX
        # Set we had a collision
        collision = True
        # Cannot collide with more than one Wall so leave loop
        break
```


The end of the moveNormal function is again the same as the moveRandom function. We want to check when the Enemy catches the player and draw the updated position to the screen.



```
# 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()
```



Curriculum mapping

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

TASK: TEST *ENEMY.MOVENORMAL* GAME FUNCTION



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

Let's now test our new Enemy.moveNormal function by adding a new Enemy object to the enemies array. 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 green.

```
enemies = [Enemy(6, 2, gamer.Screen.RED, walls, gems, player, screen, screenWidth, screenHeight)
            [Enemy(5, 5, gamer.Screen.GREEN, walls, gems, player, screen, screenWidth, screenHeight)]
```

Again, let's add the new Enemy object into the game loop underneath where we call moveRandom on the first Enemy. On the second Enemy, which we select by adding an elif or else if statement to the enemies loop, we want to call the new moveNormal function.

```
if i == 0: enemies[i].moveRandom()
# Second Enemy moves towards Player x, y
elif i == 1: enemies[i].moveNormal()
```

Don't forget to reset the new Enemy in the livesUpdate function.

```
enemies[0].reset(6, 2)
enemies[1].reset(5, 5)
```



Try testing the new Enemy and see if you can figure out what improvements we can make using the two variables we setup to use later (change and collision).



UPDATE GAME FUNCTION: *ENEMY.MOVENORMAL*



Unfortunately, the new Enemy object often can't make a valid move as there are walls in the way. To help solve this we setup collision and changeX earlier. Between the walls loop and player.collision statement inside moveNormal we are going to add some more code to make a move on the opposite axis if the first move was invalid.

First check if we even had a Wall collision, as we don't need to redo the move if it is valid. When the Enemy has collided with one of the walls, we do the move again but this time with changeX set to the opposite value it was before.

```
# If we had a collision
if collision:
    # Try change in the other direction
    if changeX: self.x += x
    else: self.y += y

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

TASK: TEST *ENEMY.MOVENORMAL* GAME FUNCTION



Now that the new Enemy object almost always makes a valid move, try playing the game to see if you can still collect all of the gems and win the game!

LESSON
9

CONCLUSION

LESSON 09 CONCLUSION



In this lesson, you have:

- Created an Enemy move function to use logic to move towards the Player
- Used for loop, if statement and Boolean logic to check for interaction between Enemy and Wall objects

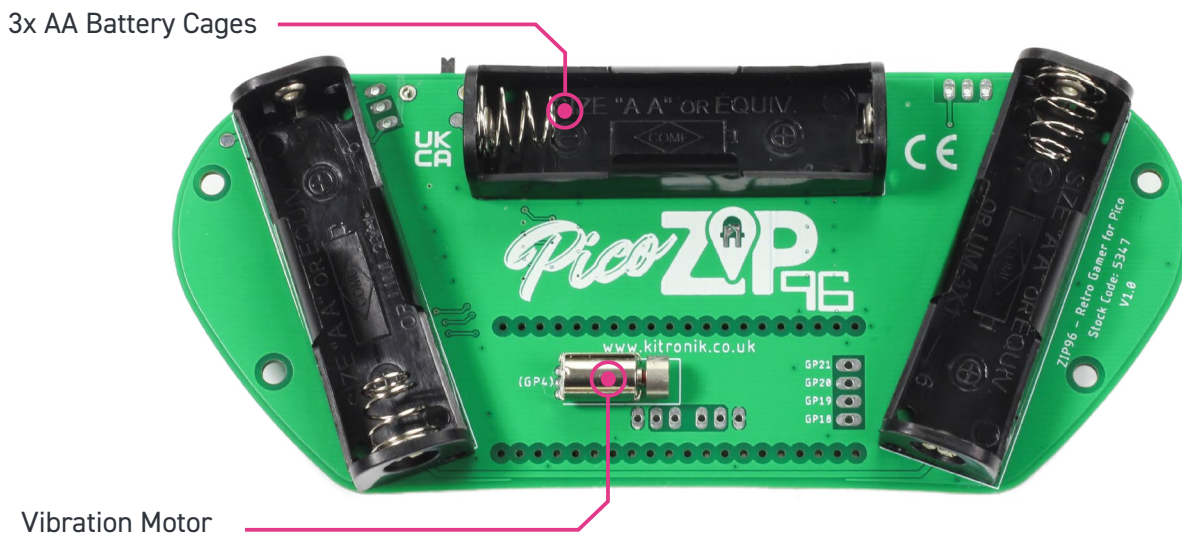
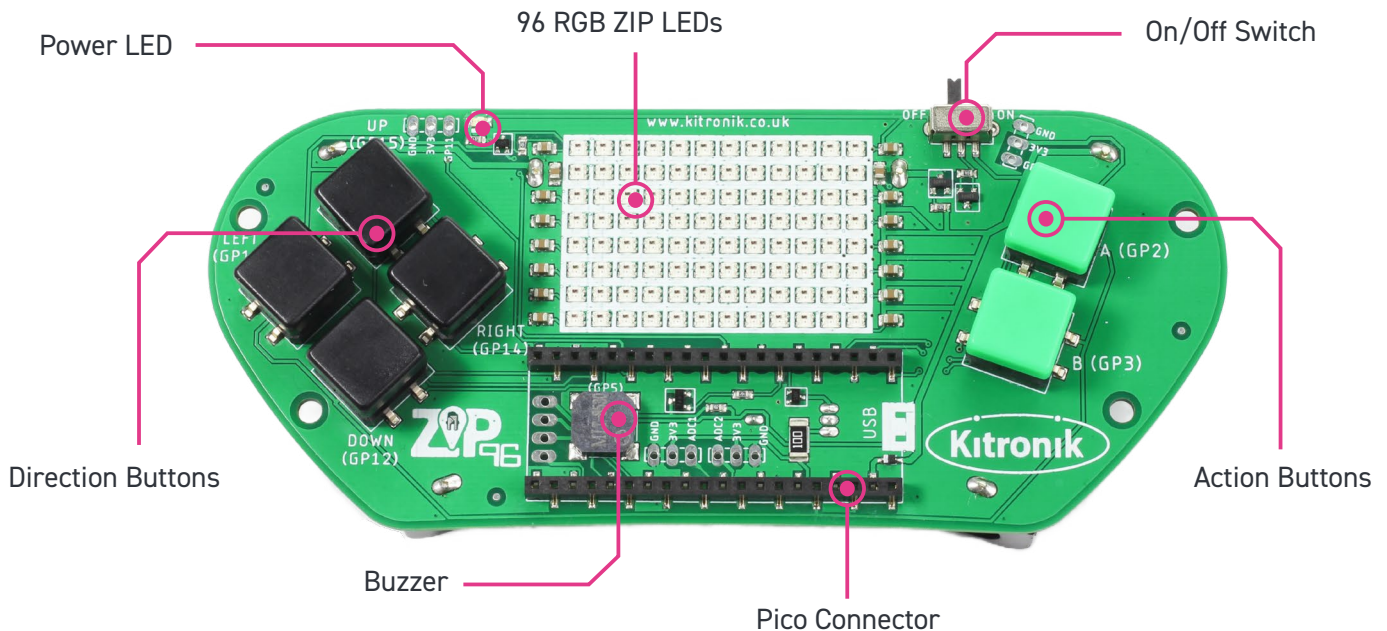
FULL LESSON CODE

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

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.