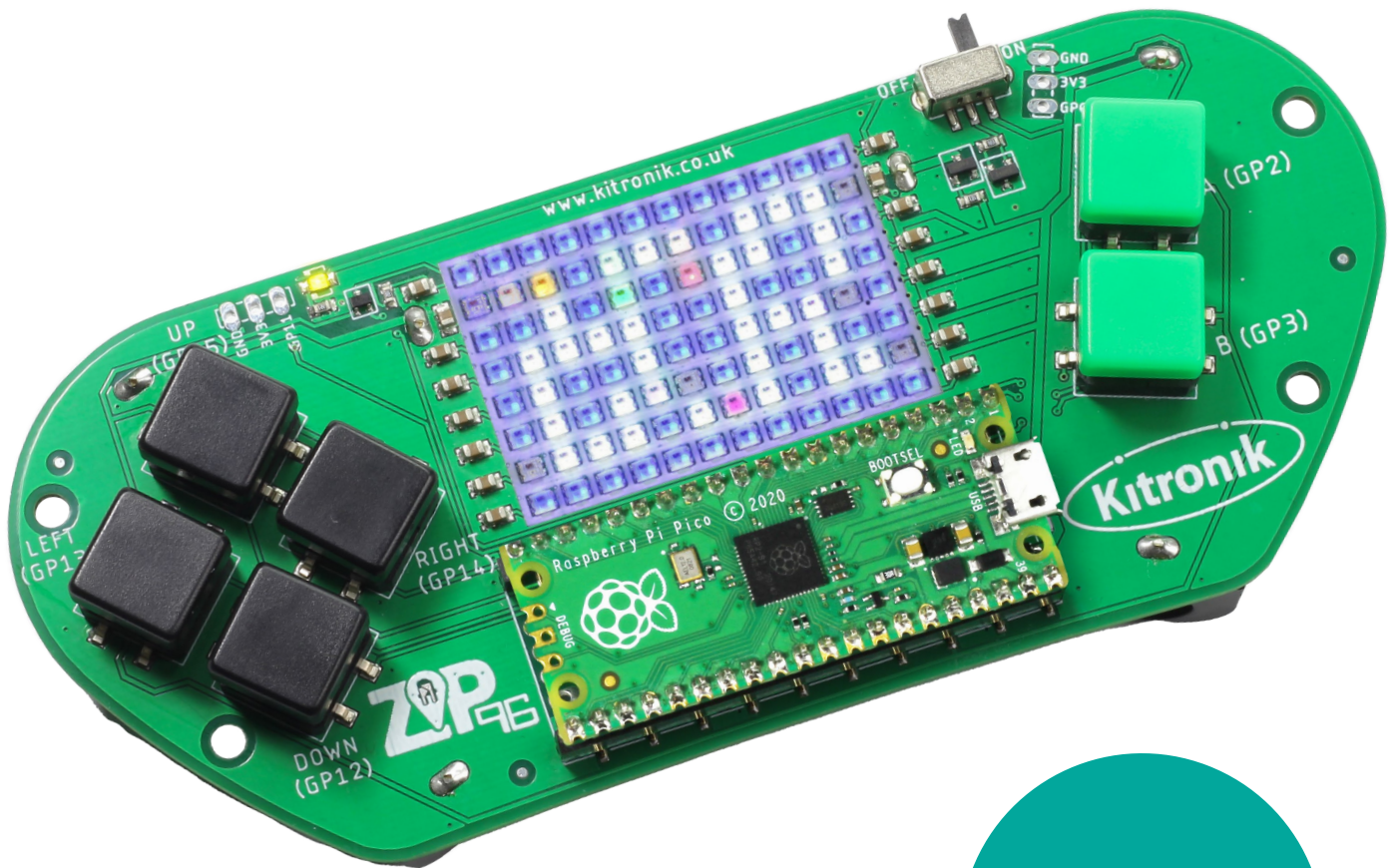


LESSON GUIDE TO THE PICO ZIP96



LESSON 1: 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 1

INTRODUCTION & SETUP



This is the first lesson in the 'A-mazing Game' series for Pico ZIP96. The Pico ZIP96 and game brief are introduced, and the software design process is explained and run through for the game.



CLASSROOM SETUP



Students will be working in pairs. They will need:

- Pen & Paper

The teacher will be writing on the board as well as demonstrating code on a projected board (if available).



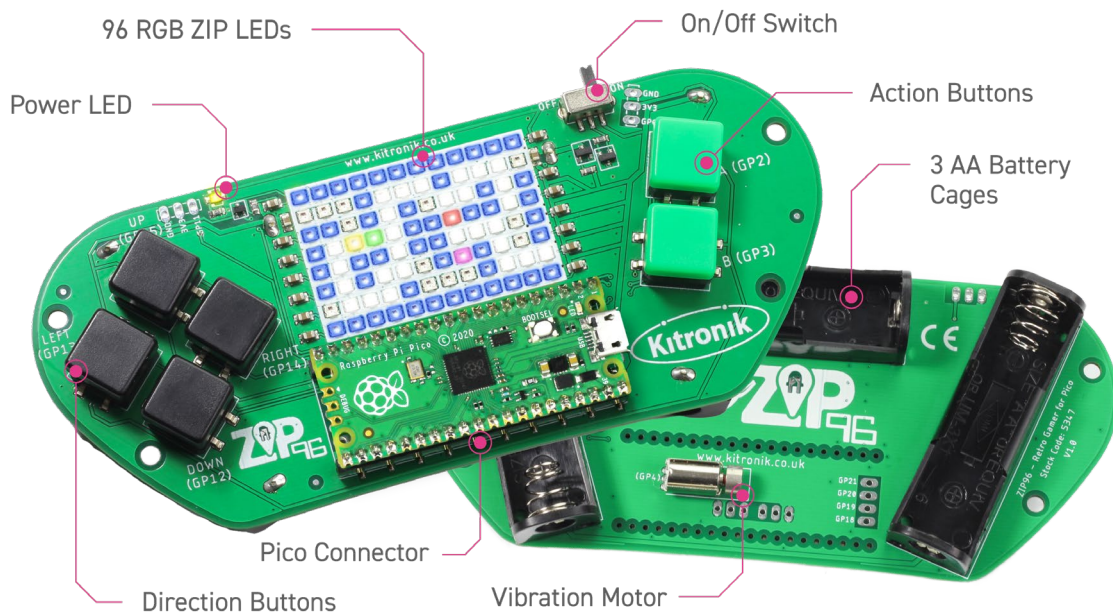
Curriculum mapping

- Decompose problems into smaller components, and make use of subroutines to build up well-structured programs.
- Document design decisions in the development of complex programs.

KEYWORDS:

SUBROUTINES, PROCEDURES, FUNCTIONS, MODULES, LIBRARIES, WELL-DESIGNED PROGRAMS, OBJECT-ORIENTED PROGRAMMING (OOP), CLASSES, SOFTWARE DESIGN, DOCUMENTATION, PRESENTATIONS

INTRODUCTION TO THE PICO ZIP96



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.



Curriculum mapping

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

When creating a new piece of software, it is important that we take time to think about what our software should do and how our software will be built to do these things. This is called software design and is a key stage in creating high quality software. Designing software before coding ensures the software meets all the project's requirements, and allows us to consider how we are going to break down the software into small sections. These sections are then pieced together later on to create the complete software. When software design is done properly it allows for the time spent coding to be reduced, the overall number of bugs to be reduced, and for the sections of software to work together better than if we just started coding straight away.



MAIN LESSON

SOFTWARE DESIGN

The requirements for this project can be found in the “What is our A-Mazing Game?” description on the previous page. Before designing software it is important that we first highlight what the project's requirements are, and this must be done at the start of any project.



Ask the students to summarize the key requirements from the A-Mazing Game brief.



For our A-Mazing Game the requirements are:



- Player moves around the maze.
- Player cannot move into walls.
- Player collects gems.
 - When all gems are collected, the player wins.
- Player has three lives to win the game.
- Enemies move around the maze.
- Enemies cannot move into walls.
- Enemies try to catch the player.
 - When an enemy catches the player, the player loses a life.

PROGRAMMING CONCEPTS



Now let's look at some basic object-oriented programming concepts.

THE FIRST CONCEPT IS A CLASS

A class is essentially a blueprint or template for a set of related code.

It defines what values are stored to begin with, and what behaviours can be performed. The behaviours defined in a class are called methods. Interaction between classes is done by calling the various methods which are available in the class interface.

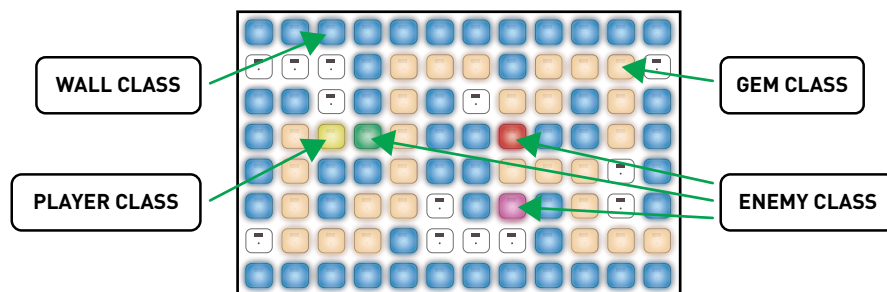
THE NEXT CONCEPT IS AN OBJECT

This is the actual running copy of a class. Many objects can be created using a single class and each object stores its own values, allowing them to be modified individually. By defining a single class to create multiple objects we can reuse code, leading to more efficient software.

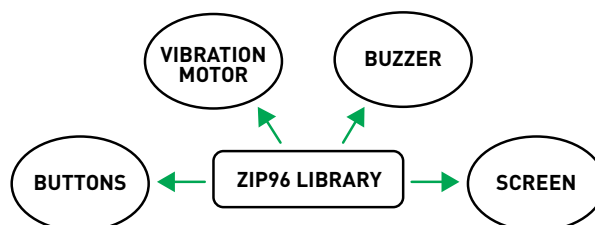
Inside an object the data refers to some characteristic or state that it is in. An object's behaviours are things that the object can do, often changing the data it stores.

LESSON 1

Before we start creating our A-Mazing Game, let's first take some time to consider how we want to build up the game from its smaller pieces. Let's think about what classes we are going to need to split up the game's functionality. From the title of our game we know it's going to be a maze, so from that we can say we'll need to have a Wall class to setup the mazes. We know that our game is going to have a player who runs through the maze, so let's say we are going to have a Player class. This will hold all the variables and methods that we are going to need for the player. Next, we know that to win the game the player needs to collect all of the gems and so we will need a Gem class. Finally, to make the game a fun challenge, we are going to create some enemies that chase after the player and for that we'll want to make an Enemy class.

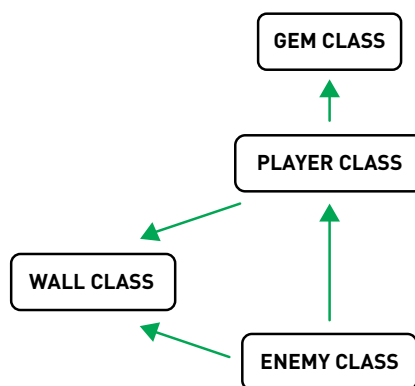


With our basic classes outlined, now we can think about our user's interactions with the game. We know they are going to need to control the player to move around the maze. For this we need to have access to the buttons on the Pico ZIP96. We also know they are going to need to see their player inside of the maze shown by the walls. On top of this they'll need to see the positions of the gems and enemies in the maze. For these we are going to access the screen on the Pico ZIP96. To access the Pico ZIP96 we are going to use a library called ZIP96Pico which we'll use from inside of our main game file.

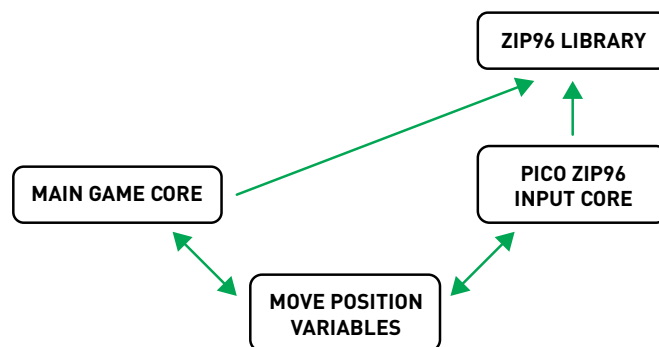


LESSON
1

Let's also take some time to think about how our classes might need to communicate with or make use of each other. To start, the Player class is going to need to store an array of the Wall objects to check that the player doesn't move into one of them. The Player class will also need to store an array of the Gem objects to check when the player does move onto them, to collect the Gem. The Enemy class is going to need to access the Player object, and an array of the Wall objects. The Enemy needs the Player object so they can chase after and know when they have caught the player. The Enemy then needs to store an array of the Wall objects as well to check that they don't move into one of them.



To make our game feel faster and our controls more responsive we are going to use two cores to handle different parts of functionality from our game. The first and main core will handle all our game's main functionality, including moving the player and enemies around the maze, and updating the screen. The second core is going to handle the input from the Pico ZIP96 buttons, and modify two move position variables. These move position variables will then be used in the main core to move the player at regular intervals, while freeing up the gamer input core to react to button presses much faster.



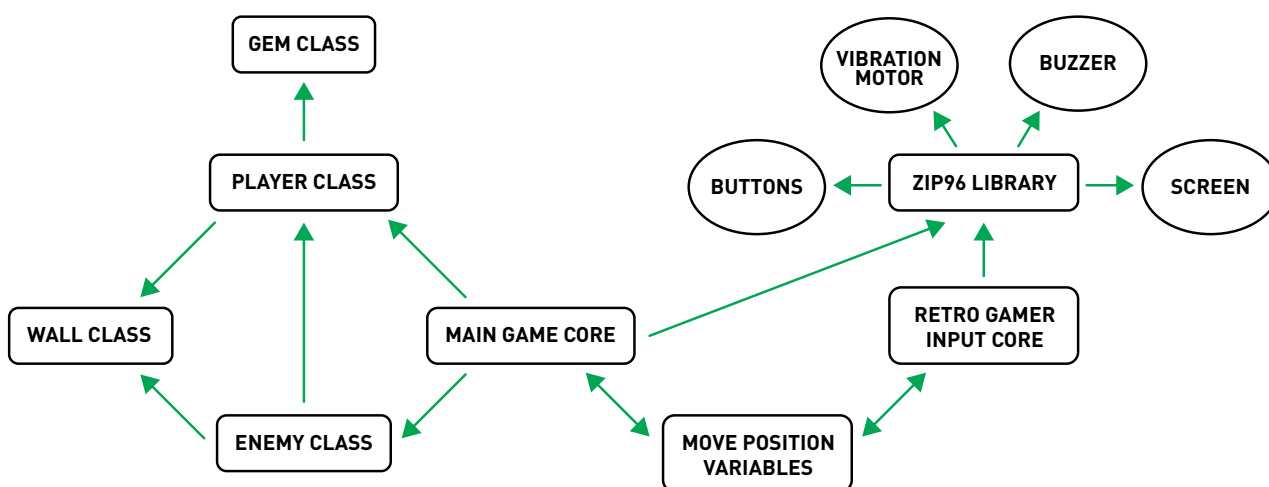
LESSON 1



Curriculum mapping

Document design decisions in the development of complex programs.

Finally, if we map out what we have learnt in an informal design diagram we can see how each part of our game can be broken down into a separate section, and how all the sections will piece together to make our A-Mazing Game.



CONCLUSION

LESSON 01 CONCLUSION



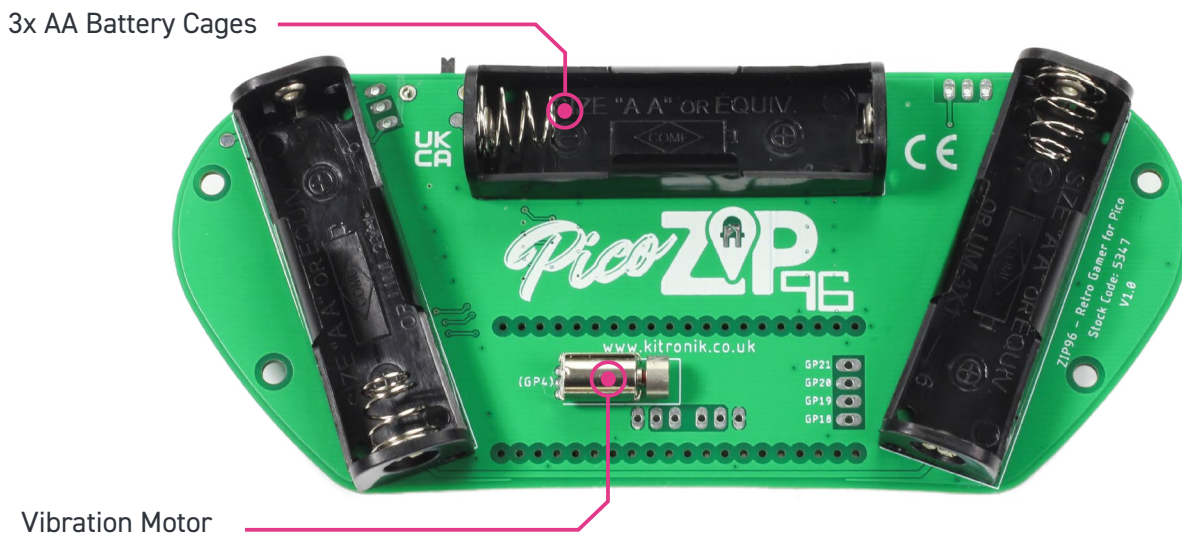
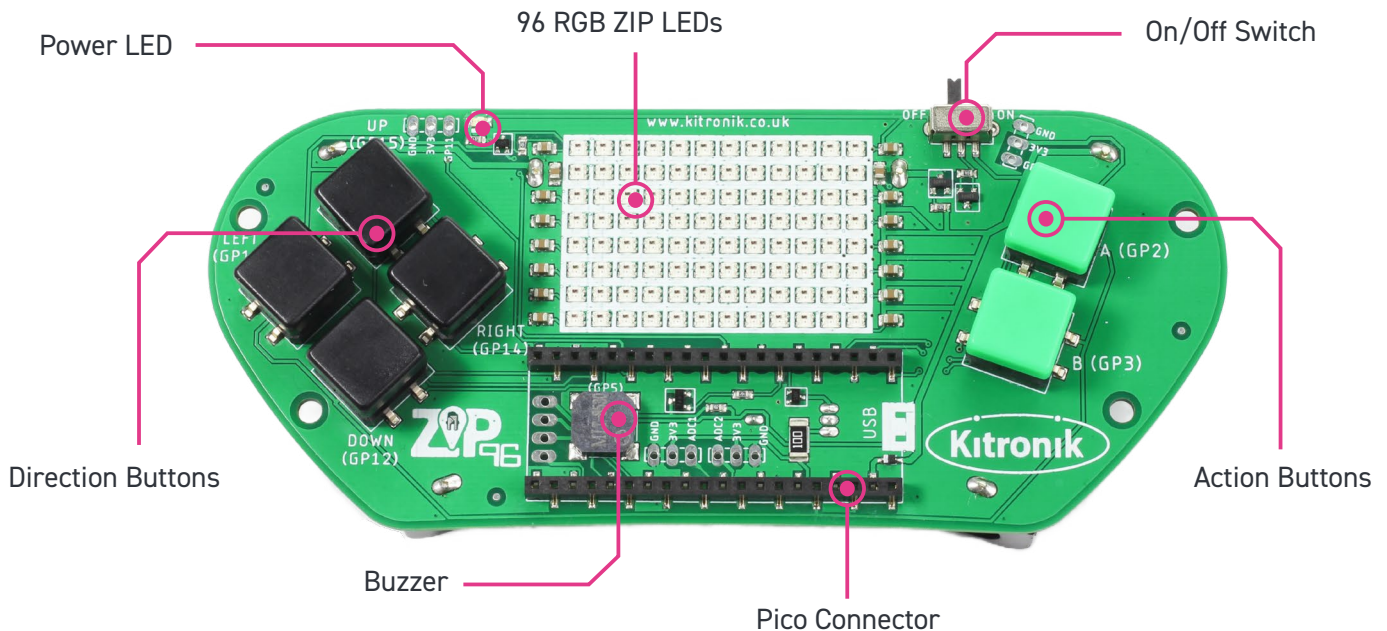
In this lesson, you have:

- Explored the concepts of:
 - Libraries
 - Classes
 - Objects
- Designed the A-Mazing Game



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.