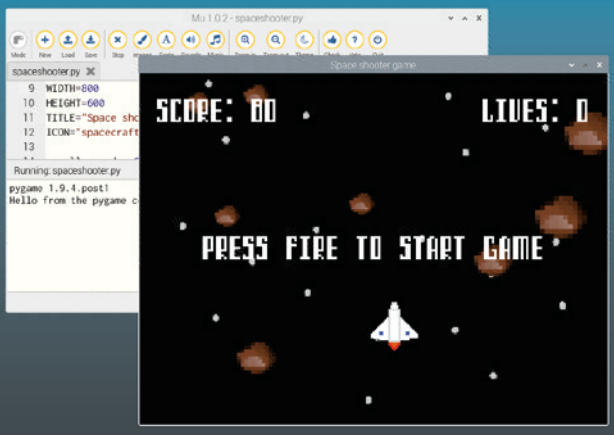
TECHNOLOGY IN ACTION™



Beginning Game Programming with Pygame Zero

Coding Interactive Games on Raspberry Pi Using Python

Stewart Watkiss

Apress®

Beginning Game Programming with Pygame Zero

Coding Interactive Games on Raspberry Pi Using Python

Stewart Watkiss

Apress®

Beginning Game Programming with Pygame Zero: Coding Interactive Games on Raspberry Pi Using Python

Stewart Watkiss
Redditch, UK

ISBN-13 (pbk): 978-1-4842-5649-7 ISBN-13 (electronic): 978-1-4842-5650-3
https://doi.org/10.1007/978-1-4842-5650-3

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Aaron Black
Development Editor: James Markham
Coordinating Editor: Jessica Vakili

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail rights@apress.com, or visit http://www.apress.com/rights-permissions.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at http://www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/978-1-4842-5649-7. For more detailed information, please visit http://www.apress.com/source-code.

Printed on acid-free paper

For my children Oliver and Amelia.
You are the inspiration in my life.

Table of Contents

About the Author

Stewart Watkiss is a keen maker and programmer. He has a master's degree in electronic engineering from the University of Hull and a master's degree in computer science from Georgia Institute of Technology.

He has over 20 years of experience in the IT industry, working in computer networking, Linux system administration, technical support, and cyber security. While working toward Linux certification, he created the web site `www.penguintutor.com`. The web site originally provided information for those studying toward certification but has since added information on electronics, projects, and learning computer programming.

Stewart often gives talks and runs workshops at local Raspberry Pi events. He is also a STEM Ambassador and Code Club volunteer, helping to support teachers and children learning programming.

About the Technical Reviewer

Sai Yamanoor is an embedded systems engineer working for an industrial gases company in Buffalo, NY. His interests, deeply rooted in DIY and open source hardware, include developing gadgets that aid behavior modification. He has published two books with his brother, and in his spare time, he likes to contribute to build things that improve quality of life. You can find his project portfolio at http://saiyamanoor.com.

Acknowledgments

My family has been very supportive in my maker activities and while writing this book. Thank you to my wife Sarah for her support and to my children Oliver and Amelia who have been a source of inspiration and help while writing the book. Oliver has been particularly helpful in testing the games and giving me feedback, and my daughter's knowledge of music was a great help while writing about making sounds.

I'd also like to thank the team behind the Raspberry Pi including the Raspberry Pi Foundation and the community that has grown around it. I've also been inspired by the work of Nicholas Tollervey who created the Mu editor that is used throughout the book and Daniel Pope who created Pygame Zero, without which the book wouldn't have been possible.

I'm also grateful to all the support from the team at Apress, to Jessica Vakili for her support in putting the book together, and to Sai Yamanoor for the technical review. There are also many other people who helped to contribute through reviews and getting the book production ready.

Introduction

This book is designed for anyone wanting to learn programming through making fun games. It will also be useful for someone who has already learned the basics of programming and wants to learn how to add fun graphics and create their own games.

It is focused on making the games rather than teaching programming theory. In this book, you're more likely to see code on how gravity affects a missile's trajectory rather than the most efficient way to search through data. Even then the code is kept simple as games should be more about playability rather than complex physics.

The book starts with a simple text-based game to cover the basics of programming in Python. It then quickly moves on to creating simple graphical games in Pygame Zero. The book introduces object-oriented programming to make it easier to make more complex games. It also explains how you can create your own graphics and sounds.

Throughout the book, you will get to apply the new techniques in a variety of 2D games. As well as some new games, there are some variations on class games including a space shooter game and battleships.

The games are designed to run on the Raspberry Pi, although they can be used on other platforms that support Python 3 with Pygame Zero.

The games you make will be playable and hopefully fun to play. They are only the beginning. If all you ever do is copy the code from this book, then you are only going to learn so much, but by adapting and improving these games, they can become more enjoyable as well as helping you learn more than you

ever will from just typing out code that's written down for you. For each of the games, there is a list of suggestions for you to develop the games further.

All the code and resource files used in the book are available from the page to accompany the book at https://www.apress.com/gb/book/9781484256497.

CHAPTER 1

Creating Computer Games

Writing computer games is a great way to make programming enjoyable, but it does have its disadvantages. The main disadvantage is that to make a working game you need to write a lot of code which takes a lot of time. A full working game is usually too much for a beginner programming book. Fear not, as this book uses worked examples and takes advantage of the simplicity of Python and Pygame Zero to make it as painless as possible. In this book you will create a few different games to illustrate different programming techniques.

Creating a game is more than just writing code. This book covers some of the other aspects of creating a computer game as well as the programming.

First you need an idea. That idea then needs to be developed to come up with a set of rules and controls. It will likely need additional resources such as images and sounds. You will then need to write the code to make it happen. Next (and now comes the fun part) you need to test it to find out what works and how it can be improved. You then go back to the start to redefine the idea and repeat the programming cycle.

In this chapter you'll also find out about Python and Pygame Zero and some of the reasons that make it suitable for game programming.

S. Watkiss, *Beginning Game Programming with Pygame Zero*,
https://doi.org/10.1007/978-1-4842-5650-3_1

Inspiration Rather Than Imitation

The first step is about coming up with an idea. For this you may take inspiration from games you have played, which can be existing computer games, card games, board games, or playground games. Or you could come up with a completely new game, perhaps taking inspiration from activities in the real world. If you are looking to create a game based on something that has already been done before, then you do need to be careful about infringing on other people's intellectual property, including copyright, patents, and trademarks.

Like many laws, the rules protecting games and computer programs are complex and vary among different countries. It would not be possible to provide real guidance on the complex legal intricacies, but there are some general rules that you should follow.

Copyright can protect various aspects of work such as words, graphics, code, and music. Copyright does not however cover the idea of the game or how it's played. The work is automatically copyrighted when it is created and doesn't normally need a specific copyright notice or registration, although that can provide additional protection.

Patents are far more complex and can cover ideas and concepts. Patents are intended for inventions, and in the case of game programming, they can be granted for specific technical aspect of a game. For example, there are patents covering the way that directions are shown in a car racing game and how players are identified in a soccer game. It's incredibly difficult to know about what patents may relate to a game you are developing. If you are creating a commercial game, then you may want to look at getting professional advice on patents.

Trademarks are a way to protect names and logos, and in the case of computer games, they can include the appearance of the characters. This may prevent you from using a recognizable character if that character

is protected under a trademark. If you want to use any character that is protected under a trademark, then you will need to get a license granting you permission from the trademark owner.

Playing Games

The best way to learn about what makes a good game is to play them. Rather than just playing one game, play lots of different ones. Play good games and bad games and think about what makes the game good and bad.

Are you getting bored playing the game or does it have you hooked so you can't drag yourself away from the screen? Which games make you want to keep playing and why?

As mentioned previously you don't just need to take inspiration from computer games. Play some board games as well. Think about what works well and what doesn't. Think about the differences between playing a game using physical objects and when it is on a computer screen; there are likely to be both advantages and disadvantages to both.

Create the Resources

When looking at additional resources, you will likely be thinking about graphics and sound effects. There are other resources that you may need including introductory videos, tutorials, and background music.

For most games you are going to want to include graphics. The look and size of these graphics can determine the programming. For example, if you have a character that needs to move around the screen, then you will need to know how the character moves (whether its feet move) and the amount of space that is needed for that character to move around. It therefore makes sense to at least create an outline of any graphics prior to starting programming.

Sound effects can sometimes be left until later in the project, although they are often still an important part of creating an overall game. If leaving them to be added later, then it is still a good idea to think about when they will be used and what impact they will have when designing the game.

Development Cycle

The main buzzword relating to programming is agile. Agile programming is a way of developing software creating code in small increments implementing a feature at a time and then going back to add more code. The term agile programming is normally used to refer to a programming technique used for developing software across a team with regular reviews and team meetings (called scrums), but a similar technique can be used when programming on your own.

Some key points about developing code using an agile style methodology:

1. Gather requirements. Meet with end users or review your ideas with yourself as though you are the customer.
2. Plan the development. Split the work into small chunks that can be implemented a bit at a time.
3. Design the code to complete the current feature.
4. Write the code.
5. Test the code. As well as testing the standalone code, test how it interacts with other parts.
6. Assess whether the code is still in line with the requirements.
7. Return to 1. Consider the code that has been created. Is that compatible with what it is trying to achieve?

Keep repeating this cycle for each part of the code you develop. You then reach a release version once all the required parts have been implemented. Follow the same cycle when adding more features or improving the code.

Some things that are useful when using agile programming:

- Design interfaces between how the different parts of the code interact.
- Work in short code sprints with incremental releases.
- Perform regular short reviews of what has been completed during the last step and what you will be creating next. Reviews are normally performed daily in a work environment but differ if you are working in your spare time.
- Perform test-driven development by having specific tests that the code needs to pass. Automated tests are popular in agile programming, but you can also test manually.
- Refactor code regularly; review code for improvements for clarity/performance.
- Regularly check with the users (or yourself if it's a personal project) to see that the design is in line with the expectations.
- Use rubber duck debugging (see Chapter 11).

The games in this book are created based around agile programming. There will not be any of the code reviews specifically listed in the book, but you will see how the code is built up starting a feature at a time.

Making Programming Enjoyable

Whether you have a full-time job writing computer games, or it's something you do in your spare time, programming should be something you enjoy. I find a great deal of satisfaction from creating something that I would like to play myself.

While you can try and think of the concepts in advance, you may not know whether you enjoy the game until you get to play it. It's then when you get to tune the game to make sure it is the right difficulty or if there are features that you will want to add. This is discussed more in Chapter 4 when you will see some of the techniques used to improve on an initial game design.

Python and Pygame Zero

Python is a popular programming language used throughout education and in industry. It is available across a number of different computer operating systems including Apple Mac OS X, Microsoft Windows, and Linux. Some of the benefits of learning Python are it is easy to learn, uses less code (compared with some other languages), and can help teach good programming techniques.

Pygame is a library that can be used within Python to make graphical game programming easier. Pygame Zero is a library that uses Pygame but makes graphical game programming even easier than Pygame by reducing the amount of code needed. Using these, it is possible to create characters on the screen and move them around very easily.

This book uses version 3.7 of Python running on Linux which is the current version on the Raspberry Pi. The games should work across different computer systems and more recent versions of Python with Pygame Zero installed.

There are different styles for programming in Python. In this book the first few programs are written using primarily functional programming techniques, but then the later programs will be based around object-oriented programming. The functional programming style is generally considered easier to learn when starting programming, but once you start creating longer programs, then it is often easier to write and understand the code when written using object-oriented programming.

Compiled vs. Interpreted

Different computers and operating systems work in different ways. If you are creating a game designed for a phone or tablet (using a touch screen), then you may need to design the interface differently than if you are designing a game for a game console with a game controller. Also, different processors inside the computer and different ways that the operating system works mean that it can be difficult to write games that will work across multiple computers.

When writing computer code, you will normally use a programming language which uses a text-based language. Computers can't run that directly, and the code needs to be converted into the machine code that the computer can understand. When using a computer language such as C, the code must be converted to machine code before you can run the program. This is known as a compiled language and the program needs to be compiled into machine code that matches the computer architecture it will run on.

Python does this differently by converting the code to the machine language using an interpreter. This is done while the program is running. The benefit of this is that as long as there is an interpreter for the computer you want to run the code, you don't normally need to do anything extra to run it on that computer. The disadvantage is that interpreted languages can run slower because it needs to convert this code while it is running.

This performance won't be an issue with any of the games in this book, but you should be aware of it if programming a graphics-intensive game.

There is also a hybrid where the code is compiled to an intermediate form, but then still needs an interpreter (or something similar) for it to run on each particular computer architecture. This is how Java works using the Java Virtual Machine to convert from the Java Bytecode to machine language the computer can understand.

As Python is interpreted, it should be able to run on a variety of different computers without needing any changes. Unfortunately, it can sometimes be a little tricky to install the Python interpreter and the Pygame Zero libraries on some platforms. Fortunately, there is a simpler solution using the Mu editor which is the preferred editor for those starting with Pygame Zero programming.

Choosing a Programming Environment

In this book the games have been designed for a Raspberry Pi, which is a small, inexpensive computer designed specifically for those learning computing and computer programming. There are different variants of the Raspberry Pi including the tiny Raspberry Pi Zero and the fully featured Raspberry Pi 4. You can use any model of Raspberry Pi for the games in this book, although I would suggest using a Raspberry Pi 2 or better for performance reasons. If you are also using the Raspberry Pi for designing images for the games (as explained in Chapter 5), then a Raspberry Pi 4 may be advantageous, but it is not a requirement.

The Raspberry Pi is ideal for learning Python as most of the software you need is already pre-installed. The programs will still run on other computers and you are free to develop the code on another platform if you prefer, but there are a few extra steps involved on other systems.

Python programs are text files, and as such, you can create them in any text editor. If you've not programmed with Python before, then I suggest

you start with the Mu editor. The Mu editor is not the most powerful editor available, but its simplicity makes it ideal for getting started. It also handles most of the setup including Pygame Zero.

If using a Raspberry Pi, then latest versions of Raspbian include Mu, but if it's not already installed, then you can install Mu from a command shell. Start the command shell by clicking the black terminal icon at the top of the screen.

Then enter the following commands:

```
sudo apt update
sudo apt install mu-editor
```

You can then run Mu from the Raspbian menu system. From the start menu, select the programming menu, then click Mu, which should look like Figure 1-1.

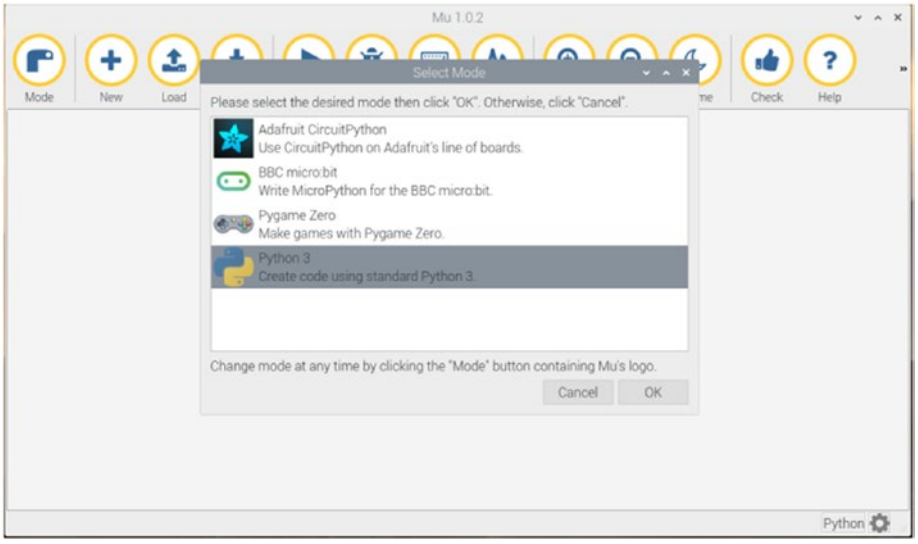


Figure 1-1. *A screenshot of the Mu editor*

If you would like to install Mu on other operating systems, then you can download the Mu editor from https://codewith.mu/. When installing under Windows, the recommendation on the Mu web site is to install for "this user only". That will make it easier to add any modules that may be required later.

The Mu editor has different modes which are useful for different programming environments. This book uses the Python 3 and Pygame Zero modes.

When you have more experience, you may want to change to a more powerful editor. If using a Raspberry Pi, then you have a number to choose from and you can run the programs directly from the command line. If you are using a different environment, then you may need to set up a native Python environment with Pygame Zero.

Summary

This chapter has looked at some of the things you should think of before you start programming. It has given suggestions on where you can get inspiration from and a warning about some of the pitfalls that you should avoid around other people's intellectual property.

It has explained what Python is and why Pygame Zero is a good choice for those starting out in game programming.

In the next chapter, you will get started with creating code and create a command-line game using Python.

CHAPTER 2

Getting Started with Python

To get started with programming Python, this chapter begins with some basic command-line programming. This will create a simple text-based game that can be played using the keyboard. This is only the beginning; from the next chapter onward, you will be able to create graphical games that are fun to play.

Using the Mu Editor

When you first start the editor, it will ask you which mode to start in. The modes that you will use for the projects in this book are Python 3 and Pygame Zero. If you have already run the editor before, then it will start in the mode last used, in which case you can change the mode using the mode button on the top left of the editor.

For this chapter you will create basic text-based program, so you should select Python 3. In future chapters, you should use Pygame Zero.

When you first start Mu, there should be an empty screen with a comment `# Write your code here :-)`.

The # at the beginning of the line means that this is a comment and would be ignored. Comments are really useful for programmers to explain how the program works, but Python just ignores them. You can

S. Watkiss, *Beginning Game Programming with Pygame Zero*,
https://doi.org/10.1007/978-1-4842-5650-3_2

delete that line for now, but when you write your own code, I suggest you add comments to explain how the code works as that can be useful in understanding the code in future.

To get started, you can create a basic program called "Hello World". It is one of the smallest programs that you can create. This is literally one line of code as shown here:

```
print ("Hello World")
```

Replace the comment in the Mu editor with this print statement. You will then need to save the program before running it; I'd suggest saving it in the default folder (/home/pi/mu_code) and calling it helloworld.py. If you try to run the code before saving, then you will be prompted to save it first.

After saving the file, click Run and you will see the program running in the bottom part of the screen. In this case it prints Hello World to the text-based screen area. This is shown in Figure 2-1.

Figure 2-1. *The Hello World program running in the Mu editor*

Once you have finished, click the Stop icon to stop the program from running.

This is the most common way of running a Python program from Mu. Another alternative is to run the program from a Raspbian Linux command shell. Save the current program using the Save button. You will see where the file is saved by looking at the status message at the bottom of the editor, in this case

```
/home/pi/mu_code/helloworld.py
```

To run this program from the command line, launch the terminal program from the Raspbian menu launcher. The terminal is a text-based interface used to communicate with the operating system including starting other programs. You can change to the folder that the program is stored in by using the cd command. The filename consists of the directory which consists of all the characters up to the last "/" character (note that the directory separator on Linux faces the opposite way to the folder separator used on the Windows operating system).

In this case the directory path is /home/pi/mu_code/ and the filename is helloworld.py. To change to the directory and run the program, enter the following commands:

```
cd /home/pi/mu_code/
python3 helloworld.py
```

Your program will now run and display the same "Hello World" text as you saw previously in the Mu output screen. This is shown in Figure 2-2.

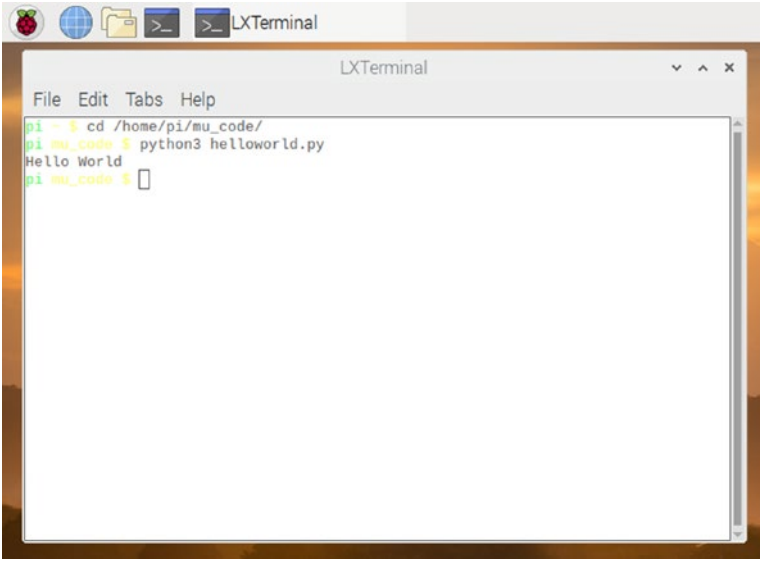


Figure 2-2. *Running the Hello World code from the command line*

Another way to run Python code is using the REPL. It stands for read-eval-print loop (but the name is not important). What the REPL does is it provides a way of running Python code in an interactive mode. This can be useful to test running small amounts of code prior to including it in your programs.

To run the same code in the REPL, click REPL in the Mu editor menu bar. You must be in the Python 3 mode to see that menu option. If the REPL icon is not shown, then use the mode icon on the Mu menu bar to change mode. After clicking the REPL icon, there will be an interactive shell at the bottom of the screen. Note that if your previous programming is still running, then it will show the program output and the REPL side by side, and if so, then click the Stop button which will give the REPL the full width of the editor.

You will see a prompt in the REPL screen which will normally show "IN [1]:". Enter the previous program code at the prompt

```
print ("Hello World")
```