INSTANT

Short | Fast | Focused

Pygame for Python Game Development How-to

Create engaging and fun games with Pygame, Python's Game development library

Ivan Idris

[PACKT] PUBLISHING

Instant Pygame for Python Game Development How-to

Create engaging and fun games with Pygame, Python's Game development library

Ivan Idris

[PACKT] PUBLISHING

BIRMINGHAM - MUMBAI

Instant Pygame for Python Game Development How-to

First published: March 2013

Production Reference: 1180313

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-78216-286-5

www.packtpub.com

Credits

Author
Ivan Idris

Reviewer
Will McGugan

Acquisition Editor
Kartikey Pandey

Commissioning Editor
Poonam Jain

Technical Editor
Jalasha D'costa

Project Coordinator
Sherin Padayatty

Proofreader
Katherine Tarr

Production Coordinator
Melwyn D'sa

Cover Work
Melwyn D'sa

Cover Image
Valentina D'silva

About the Author

Ivan Idris has an MSc in Experimental Physics. His graduation thesis had a strong emphasis on Applied Computer Science. After graduating, he worked for several companies as a Java Developer, Data warehouse Developer, and QA Analyst. His main professional interests are Business Intelligence, Big Data, and Cloud Computing. Ivan Idris enjoys writing clean, testable code and interesting technical articles. Ivan Idris is the author of *NumPy 1.5 Beginner's Guide* and *NumPy Cookbook* by *Packt Publishing*. You can find more information and a blog with a few NumPy examples at `ivanidris.net`.

I would like to take this opportunity to thank the reviewers and the team at Packt Publishing for making this book possible. Also thanks goes to my teachers, professors, and colleagues who taught me about science and programming. Last but not least, I would like to acknowledge my parents, family, and friends for their support.

About the Reviewer

Will McGugan is a freelance software developer based in London and the author of Game Development with Python and PyGame. Will worked in the video games industry for many years, on PC and console titles, where he specialized in graphics technology. Since leaving the games industry, he has focused on the Python programming language for web application development, and associated frontend technologies.

Currently, he is working on a web-based platform to monitor and remotely administer unattended sensor devices. He also maintains a number of open source libraries.
For more information on Will's projects, see his blog at `http://www.willmcgugan.com`.

I'd like to thank the fine people at Packt for their assistance with this project, and Ivan Idris for his work on this title.

www.PacktPub.com

Support files, eBooks, discount offers and more

You might want to visit `www.PacktPub.com` for support files and downloads related to your book.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at `www.PacktPub.com` and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at `service@packtpub.com` for more details.

At `www.PacktPub.com`, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.

PACKTLiB™

`http://PacktLib.PacktPub.com`

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can access, read and search across Packt's entire library of books.

Why Subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print and bookmark content
- On demand and accessible via web browser

Free Access for Packt account holders

If you have an account with Packt at `www.PacktPub.com`, you can use this to access PacktLib today and view nine entirely free books. Simply use your login credentials for immediate access.

To my family and friends

Table of Contents

Preface

Pygame is a fun Python API with which we can easily create simple games. It has support for drawing, sound, images, animation, OpenGL, and more. We will explore many aspects of Pygame with more than 20 recipes in this book. The book is self-contained and only assumes that you have basic knowledge of Python programming. You are encouraged to try out all the example games and modify them to your own needs.

What this book covers

Preparing your development environment (Simple) is a basic recipe that will help you with the installation of all the necessary software required in your development environment.

Running a simple game (Simple) is where we will create a basic game to get us started. The game demonstrates fonts and screen management in the time-honored tradition of Hello world examples.

Drawing with Pygame (Simple) teaches us how to draw basic shapes such as rectangles, ovals, circles, lines, and others. We will also learn important information about colors and color management.

Animating objects (Simple) starts with the "Hit the avatar!" running game example of this book. We will learn how to animate objects in our funny little game.

Using fonts(Simple) is about fonts and font management.

Using Matplotlib with Pygame (Simple) lets us plot a graph within Pygame with the amazing open source Python plotting library Matplotlib. Matplotlib is highly versatile and offers a ton of features for plotting and visualization.

Accessing surface pixel data (Intermediate) shows us how to manipulate pixel data stored in special arrays for efficient drawing. The efficient NumPy open source Python mathematical library is introduced in this recipe.

Accessing sound data (Simple) has us process audio data as arrays. This recipe requires you to listen well while running the example code for this recipe.

Playing a movie (Intermediate) guides us through the steps required to play a movie. The value an in-game movie will add to your game is just priceless.

Pygame on Android (Intermediate) introduces us to the wonderful world of Android. Android is a well-known open source mobile computing framework originally created by Google. We create an example Android game in the process.

Artificial intelligence (Intermediate) is a hot topic these days. We dive right in with the popular Scikits-learn open source Python framework. Of course, this is a huge topic that could require years to master. We expose the tip of the iceberg and give clustering a go.

Drawing sprites (Intermediate) talks us through sprite management. Sprites is a term from computer graphics denoting two-dimensional visible objects that we can manipulate on the screen. Sprites can be grouped together for easier management.

Using OpenGL with Pygame (Advanced) helps us get a hold on OpenGL, a famous open source graphics framework that is available on a variety of platforms and programming languages. OpenGL is used in the industry to render complex two-dimensional and three-dimensional objects.

Detecting collisions (Intermediate) is essential for good game development, whether we are crashing into a car, deploying air-to-air missiles, or playing football. We will give tips on how to detect collisions with ease.

Adding networking functionality (Advanced) runs us through a rudimentary client-server setup. We will use the brilliant open source Python Twisted framework to create a networked game. The game requires us to guess a word that is only known at the server side.

Debugging your game (Intermediate) gives you the life-saving debugging techniques you will need to create a robust working game. Debugging is stressful, so it helps when you have reliable tools. We will introduce such a tool.

Profiling your code (Intermediate) is something you should do to ensure that your game performs well. Tips and ideas are given in this recipe to facilitate the profiling process.

Puzzle game with Pygame (Advanced) showcases an interactive client-server game, building on all the previously learned material.

Simulating with Pygame (Advanced) simulates life in a simplistic yet fun manner.

What you need for this book

This book is pretty self-contained. Instructions to install required software are given throughout the book.

Who this book is for

This book is for Pythonistas who are interested in learning how to create games with all the accompanying bells and whistles. Even if you don't know Python that well, the book should be easy to follow.

Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text are shown as follows: " We can sort with the following `sort` command."

A block of code is set as follows:

```
import os, pygame
from pygame.locals import *
import numpy
from scipy import ndimage

def get_pixar(arr, weights):
  states = ndimage.convolve(arr, weights, mode='wrap')

  bools = (states == 13) | (states == 12 ) | (states == 3)

  return bools.astype(int)
```

Any command-line input or output is written as follows:

```
ffmpeg -i <infile> -vcodec mpeg1video -acodec libmp3lame -intra <outfile.
mpg>
```

New terms and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "you should see **Hello** being displayed followed by **1 IMPORTANT MESSAGE!** to **19 IMPORTANT MESSAGE!**".

> Warnings or important notes appear in a box like this.

Tips and tricks appear like this.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to `feedback@packtpub.com`, and mention the book title via the subject of your message.

If there is a book that you need and would like to see us publish, please send us a note in the **SUGGEST A TITLE** form on `www.packtpub.com` or e-mail `suggest@packtpub.com`.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on `www.packtpub.com/authors`.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at `http://www.PacktPub.com`. If you purchased this book elsewhere, you can visit `http://www.PacktPub.com/support` and register to have the files e-mailed directly to you.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting `http://www.packtpub.com/support`, selecting your book, clicking on the **errata submission form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from `http://www.packtpub.com/support`.

Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at `copyright@packtpub.com` with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

Questions

You can contact us at `questions@packtpub.com` if you are having a problem with any aspect of the book, and we will do our best to address it.

Instant Pygame for Python Game Development How-to

Welcome to *Pygame for Python Game Development How-to*. This book is for developers who want to create games with Pygame quickly and easily and get familiar with the important aspects of it. The typical things you would learn are as follows:

- Pygame basics
- Sprites
- OpenGL

Pygame is part of the Python framework, originally written by Pete Shinners, that as its name suggests can be used to create video games. Pygame is free and open source since 2004 and licensed under the GPL license, which means that you are allowed to basically make any type of game. Pygame is built on top of the **Simple DirectMedia Layer** (**SDL**). SDL is a C framework that gives access to graphics, sound, keyboard, and other input devices on various operating systems including Linux, Mac OS X, and Windows.

Preparing your development environment (Simple)

We will install Python, Pygame, and other software we will need.

Getting ready

Before we install Pygame, we need to have Python installed. On some operating systems Python is already installed. Pygame should be compatible with all Python versions. We will also need the NumPy numerical library. I am the author of two books published by Packt Publishing about NumPy – *NumPy Beginner's Guide* and *NumPy Cookbook*. Please refer to these books for more info about NumPy.

How to do it...

- Installing on Debian and Ubuntu

  Python might be already installed on Debian and Ubuntu, but the development headers are usually not. On Debian and Ubuntu, install python and python-dev with these commands:

  ```
  sudo apt-get install python
  sudo apt-get install python-dev
  ```

  Pygame can be found in the Debian archives `http://packages.qa.debian.org/p/pygame.html`. We can install NumPy with the following command:

  ```
  sudo apt-get install python-numpy
  ```

- Installing on Windows

  The Windows Python installer can be found on `www.python.org/download`. On this website we can also find installers for Mac OS X and source tarballs for Linux, Unix, and Mac OS X.

  From the Pygame website (`http://www.pygame.org/download.shtml`), we can download the appropriate binary installer for the Python version we are using.

  Download a NumPy installer for Windows from the SourceForge website (`http://sourceforge.net/projects/numpy/files/`).

- Installing Python on the Mac

  Python comes preinstalled on Mac OS X. We can also get Python via MacPorts, Fink, or similar projects. We can install for instance the Python 2.6 port by running the following command:

  ```
  sudo port install python26
  ```

Binary Pygame packages for Mac OS X 10.3 and up can be found on `http://www.pygame.org/download.shtml`. We can get a NumPy installer from the SourceForge website (`http://sourceforge.net/projects/numpy/files/`). Download the appropriate `.DMG` file. Usually the latest one is the best.

- Installing from source

  Pygame is using the `distutils` system for compiling and installing. To start installing Pygame with the default options, simply run the following command:

  ```
  python setup.py
  ```

  If you need more information about the available options, type the following command:

  ```
  python setup.py help
  ```

  In order to compile the code, you need to have a compiler for your operating system. Setting this up is beyond the scope of this book. More information about compiling Pygame on Windows can be found on `http://pygame.org/wiki/CompileWindows`. More information about compiling Pygame on Mac OS X can be found at `http://pygame.org/wiki/MacCompile`.

Running a simple game (Simple)

We will create a simple game that we will improve on further in the book. As is traditional in books about programming, we will start with a Hello World! example. It's not a game per se. It's important to notice the so-called main game loop where all the action happens and the usage of the `Font` module to render text. In this program we will manipulate a Pygame's `Surface` object, that is used for drawing and we will handle a quit event.

How to do it...

1. **Imports**: First we will import the required Pygame modules. If Pygame is installed properly, we should get no errors, otherwise please return to the *Preparing your development environment (Simple)* recipe:

   ```
   import pygame, sys
   from pygame.locals import *
   ```

2. **Initialization**: We will initialize Pygame by creating a display of 400 by 300 pixels and setting the window title to `Hello world`:

   ```
   pygame.init()
   screen = pygame.display.set_mode((400, 300))

   pygame.display.set_caption('Hello World!')
   ```

3. **The main game loop**: Games usually have a game loop, which runs forever until, for instance, a quit event occurs. In this example, we will only set a label with the text `Hello world` at coordinates (100, 100). The text has a font size of 19, red color, and falls back to the default font:

```
while True:
   sys_font = pygame.font.SysFont("None", 19)
   rendered = sys_font.render('Hello World', 0, (255, 100, 100))
   screen.blit(rendered, (100, 100))

   for event in pygame.event.get():
      if event.type == QUIT:
         pygame.quit()
         sys.exit()

   pygame.display.update()
```

We get the following screenshot as the end result:

The following is the complete code for the Hello World example:

```
import pygame, sys
from pygame.locals import *

pygame.init()
screen = pygame.display.set_mode((400, 300))

pygame.display.set_caption('Hello World!')

while True:
   sysFont = pygame.font.SysFont("None", 19)
   rendered = sysFont.render('Hello World', 0, (255, 100, 100))
   screen.blit(rendered, (100, 100))
```

```
        for event in pygame.event.get():
            if event.type == QUIT:
                pygame.quit()
                sys.exit()

        pygame.display.update()
```

How it works...

It might not seem like much, but we learned a lot in this recipe. The functions that passed the review are summarized in the following table:

Function	Description
`pygame.init()`	This function performs the initialization and needs to be called before any other Pygame functions are called.
`pygame.display.set_mode((400, 300))`	This function creates a so-called `Surface` object to draw on. We give this function a tuple representing the width and height of the surface.
`pygame.display.set_caption('Hello World!')`	This function sets the window title to a specified string value.
`pygame.font.SysFont("None", 19)`	This function creates a system font from a comma-separated list of fonts (in this case none) and a font size parameter.
`sysFont.render('Hello World', 0, (255, 100, 100))`	This function draws text on a surface. The second parameter indicates whether anti-aliasing is used. The last parameter is a tuple representing the RGB values of a color.
`screen.blit(rendered, (100, 100))`	This function draws on a surface.
`pygame.event.get()`	This function gets a list of `Event` objects. Events represent some special occurrence in the system, such as a user quitting the game.
`pygame.quit()`	This function cleans up resources used by Pygame. Call this function before exiting the game.
`pygame.display.update()`	This function refreshes the surface.

Drawing with Pygame (Simple)

Before we start creating cool games, we need an introduction to the drawing functionality of Pygame. As we noticed in the previous recipe, in Pygame we draw on the `Surface` objects. There is a myriad of drawing options—different colors, rectangles, polygons, lines, circles, ellipses, animation, and different fonts.

How to do it...

The following steps will help you diverge into the different drawing options you can use with Pygame:

1. **Imports**: We will need the NumPy library to randomly generate RGB values for the colors, so we will add an extra import for that:

   ```
   import numpy
   ```

2. **Initializing colors**: Generate four tuples containing three RGB values each with NumPy:

   ```
   colors = numpy.random.randint(0, 255, size=(4, 3))
   ```

   Then define the white color as a variable:

   ```
   WHITE = (255, 255, 255)
   ```

3. **Set the background color**: We can make the whole screen white with the following code:

   ```
   screen.fill(WHITE)
   ```

4. **Drawing a circle**: Draw a circle in the center with the window using the first color we generated:

   ```
   pygame.draw.circle(screen, colors[0], (200, 200), 25, 0)
   ```

5. **Drawing a line**: To draw a line we need a start point and an end point. We will use the second random color and give the line a thickness of `3`:

   ```
   pygame.draw.line(screen, colors[1], (0, 0), (200, 200), 3)
   ```

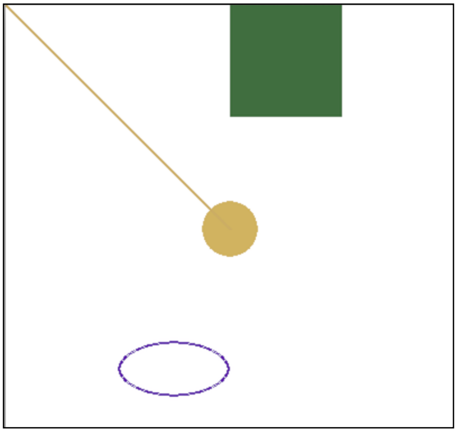
6. **Drawing a rectangle**: When drawing a rectangle, we are required to specify a color, the coordinates of the upper-left corner of the rectangle, and its dimensions:

   ```
   pygame.draw.rect(screen, colors[2], (200, 0, 100, 100))
   ```

7. **Drawing an ellipse**: You might be surprised to discover that drawing an ellipse requires similar parameters as for rectangles. The parameters actually describe an imaginary rectangle that can be drawn around the ellipse:

   ```
   pygame.draw.ellipse(screen, colors[3], (100, 300, 100, 50), 2)
   ```

The resulting window with a circle, line, rectangle, and ellipse using random colors:



The code for the drawing demo is as follows:

```
import pygame, sys
from pygame.locals import *
import numpy

pygame.init()
screen = pygame.display.set_mode((400, 400))

pygame.display.set_caption('Drawing with Pygame')
colors = numpy.random.randint(0, 255, size=(4, 3))

WHITE = (255, 255, 255)

#Make screen white
screen.fill(WHITE)

#Circle in the center of the window
pygame.draw.circle(screen, colors[0], (200, 200), 25, 0)

# Half diagonal from the upper-left corner to the center
pygame.draw.line(screen, colors[1], (0, 0), (200, 200), 3)

pygame.draw.rect(screen, colors[2], (200, 0, 100, 100))

pygame.draw.ellipse(screen, colors[3], (100, 300, 100, 50), 2)

while True:
```

```
    for event in pygame.event.get():
       if event.type == QUIT:
          pygame.quit()
          sys.exit()

    pygame.display.update()
```

Animating objects (Simple)

Now that we know how to draw with Pygame, it's time to try something more dynamic. Most games, even the most static ones, have some level of animation. From a programmer's standpoint, animation is nothing more than displaying an object at a different place at a different time, thus simulating movement.

Pygame offers a `Clock` object that manages how many frames are drawn per second. This ensures that animation is independent of how fast the user's CPU is.

How to do it...

We will load an image and use NumPy again to define a clockwise path around the screen:

1. First, we need to create a clock as follows:

   ```
   clock = pygame.time.Clock()
   ```

2. **Loading an image**: As part of the source code accompanying this book, there should be a picture of a head. We will load this image and move it around on the screen:

   ```
   img = pygame.image.load('head.jpg')
   ```

3. **Initializing arrays**: We will define some arrays to hold the coordinates of the positions, where we would like to put the image during the animation. Since the object will be moved, there are four logical sections of the path: right, down, left, and up. Each of these sections will have 40 equidistant steps. We will initialize all the values in the sections to 0:

   ```
   steps = numpy.linspace(20, 360, 40).astype(int)
   right = numpy.zeros((2, len(steps)))
   down = numpy.zeros((2, len(steps)))
   left = numpy.zeros((2, len(steps)))
   up = numpy.zeros((2, len(steps)))
   ```

4. **Setting the coordinates of the positions**: It's trivial to set the coordinates of the positions of the image. However, there is one tricky bit to notice, the `[::-1]` notation leads to reversing the order of the array elements:

   ```
   right[0] = steps
   right[1] = 20
   ```

```
down[0] = 360
down[1] = steps

left[0] = steps[::-1]
left[1] = 360

up[0] = 20
up[1] = steps[::-1]
```

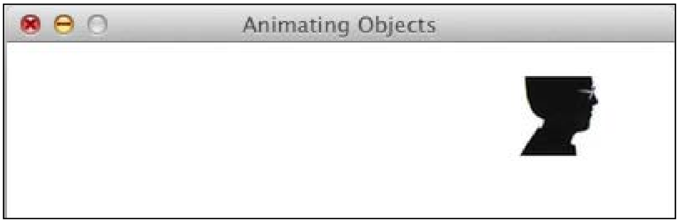
5. **Joining the sections**: The path sections can be joined, but before we can do this, the arrays have to be transposed with the `T` operator:

   ```
   pos = numpy.concatenate((right.T, down.T, left.T, up.T))
   ```

6. **Setting the clock rate**: In the main event loop, we will let the clock tick at a rate of 30 frames per second:

   ```
   clock.tick(30)
   ```

   The following screenshot is of the moving head:



You should be able to watch a movie of this animation on `https://www.youtube.com/watch?v=m2TagGiq1fs`.

The code of this example uses almost everything we learned so far, but should still be simple enough to understand:

```
import pygame, sys
from pygame.locals import *
import numpy

pygame.init()
clock = pygame.time.Clock()
screen = pygame.display.set_mode((400, 400))

pygame.display.set_caption('Animating Objects')
img = pygame.image.load('head.jpg')
```

```
steps = numpy.linspace(20, 360, 40).astype(int)
right = numpy.zeros((2, len(steps)))
down = numpy.zeros((2, len(steps)))
left = numpy.zeros((2, len(steps)))
up = numpy.zeros((2, len(steps)))

right[0] = steps
right[1] = 20

down[0] = 360
down[1] = steps

left[0] = steps[::-1]
left[1] = 360

up[0] = 20
up[1] = steps[::-1]

pos = numpy.concatenate((right.T, down.T, left.T, up.T))
i = 0

while True:
   # Erase screen
   screen.fill((255, 255, 255))

   if i >= len(pos):
      i = 0

   screen.blit(img, pos[i])
   i += 1

   for event in pygame.event.get():
      if event.type == QUIT:
         pygame.quit()
         sys.exit()

   pygame.display.update()
   clock.tick(30)
```

How it works...

We learned a bit about animation in this recipe. The most important concept we learned is the clock. The new functions that we used are described in the following table:

Function	Description
`pygame.time.Clock()`	This function creates a game clock
`numpy.linspace(20, 360, 40)`	This function creates an array with 40 equidistant values between 20 and 360
`numpy.zeros((2, len(steps)))`	This function creates an array of the specified dimensions filled with zeroes
`numpy.concatenate((right.T, down.T, left.T, up.T))`	This function concatenates arrays to form a new array
`clock.tick(30)`	This function executes a tick of the game clock, where 30 is the number of frames per second

Using fonts (Simple)

Frequently there is a need to display some text, for instance, a counter or a message.

How to do it...

Pygame has a `font` module that can help us to show text.

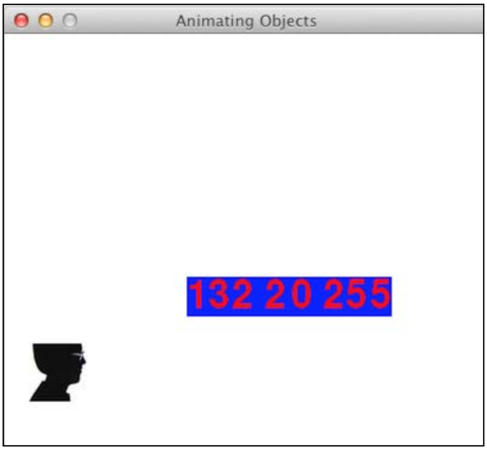
1. **Creating a font**: We can create a font by specifying, the font filename, and font size as constructor parameters:

   ```
   font = pygame.font.Font('freesansbold.ttf', 32)
   ```

2. **Displaying text**: Since we made an image move around the edge in the previous recipe, it would be great to display a counter and the position of the image in the center of the screen with a blue background and red letters. The following code snippet accomplishes this:

   ```
   text = "%d %d %d" % (i, pos[i][0], pos[i][1])
   rendered = font.render(text, True, RED, BLUE)
   screen.blit(rendered, (150, 200))
   ```

> A screenshot of the animation is shown as follows and should be on YouTube too at `https://www.youtube.com/watch?v=xhjfcFhaXN0`.



The code is almost the same as for the previous recipe, with the addition of code for the creation and display of fonts:

```
import pygame, sys
from pygame.locals import *
import numpy

pygame.init()
clock = pygame.time.Clock()
screen = pygame.display.set_mode((400, 400))

pygame.display.set_caption('Animating Objects')
img = pygame.image.load('head.jpg')

steps = numpy.linspace(20, 360, 40).astype(int)
right = numpy.zeros((2, len(steps)))
down = numpy.zeros((2, len(steps)))
left = numpy.zeros((2, len(steps)))
up = numpy.zeros((2, len(steps)))

right[0] = steps
right[1] = 20

down[0] = 360
down[1] = steps
```

```
left[0] = steps[::-1]
left[1] = 360

up[0] = 20
up[1] = steps[::-1]

pos = numpy.concatenate((right.T, down.T, left.T, up.T))
i = 0

# create a font
font = pygame.font.Font('freesansbold.ttf', 32)
RED = (255, 0, 0)
BLUE = (0, 0, 255)

while True:
   # Erase screen
   screen.fill((255, 255, 255))

   if i >= len(pos):
      i = 0

   screen.blit(img, pos[i])

   # displaying text in the center of the screen
   text = "%d %d %d" % (i, pos[i][0], pos[i][1])
   rendered = font.render(text, True, RED, BLUE)
   screen.blit(rendered, (150, 200))
   i += 1

   for event in pygame.event.get():
      if event.type == QUIT:
         pygame.quit()
         sys.exit()

   pygame.display.update()
   clock.tick(30)
```

Using Matplotlib with Pygame (Simple)

Matplotlib is an open source library for easy plotting. We can integrate Matplotlib into Pygame game and create various plots. You can find the Matplotlib installation instructions at `http://matplotlib.org/users/installing.html`.

How to do it...

In this recipe we will take the position coordinates of the previous recipe and make a graph of them:

1. **Using a non-interactive backend**: In order to integrate Matplotlib with Pygame, we need to use a non-interactive backend, otherwise Matplotlib will present us with a GUI window by default. We will import the main Matplotlib module and call the `use` function. This function has to be called immediately after importing the main `matplotlib` module and before other `matplotlib` modules are imported:

   ```
   import matplotlib

   matplotlib.use("Agg")
   ```

2. **Creating a Matplotlib canvas**: Non-interactive plots can be drawn on a Matplotlib canvas. Creating this canvas requires imports, a figure, and a subplot. We will specify the figure to be 3 by 3 inches large. More details can be found at the end of this recipe:

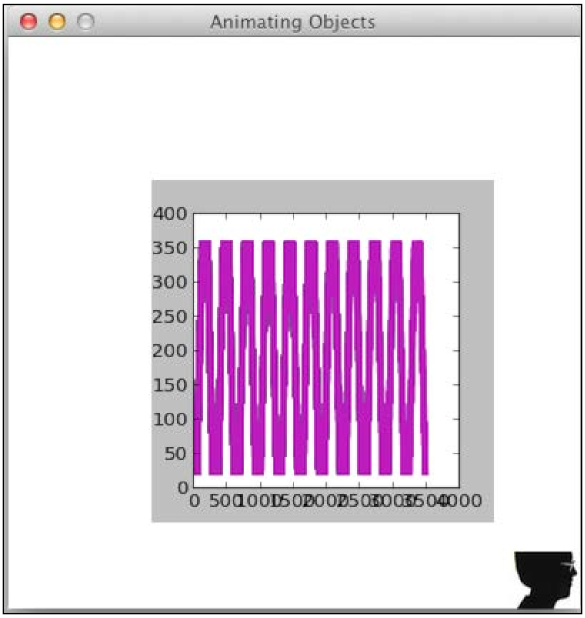
   ```
   import matplotlib.pyplot as plt
   import matplotlib.backends.backend_agg as agg

   fig = plt.figure(figsize=[3, 3])
   ax = fig.add_subplot(111)
   canvas = agg.FigureCanvasAgg(fig)
   ```

3. **Plotting data**: In a non-interactive mode, plotting data is a bit more complicated than in the default mode. Since we need to plot repeatedly, it makes sense to organize the plotting code in a function. The plot is eventually drawn on the canvas. The canvas adds a bit of complexity to our setup. At the end of this example, you can find more detailed explanation of the functions:

   ```
   def plot(data):
      ax.plot(data)
      canvas.draw()
      renderer = canvas.get_renderer()

      raw_data = renderer.tostring_rgb()
      size = canvas.get_width_height()
   ```

```
    return pygame.image.fromstring(raw_data, size, "RGB")
```

> The following screenshot shows the animation in action. You can also view a screencast on YouTube at `https://www.youtube.com/watch?v=t6qTeXxtnl4`.



We get the following code after the changes:

```
import pygame, sys
from pygame.locals import *
import numpy
import matplotlib

matplotlib.use("Agg")

import matplotlib.pyplot as plt
import matplotlib.backends.backend_agg as agg

fig = plt.figure(figsize=[3, 3])
ax = fig.add_subplot(111)
canvas = agg.FigureCanvasAgg(fig)

def plot(data):
    ax.plot(data)
```

```
    canvas.draw()
    renderer = canvas.get_renderer()

    raw_data = renderer.tostring_rgb()
    size = canvas.get_width_height()

    return pygame.image.fromstring(raw_data, size, "RGB")

pygame.init()
clock = pygame.time.Clock()
screen = pygame.display.set_mode((400, 400))

pygame.display.set_caption('Animating Objects')
img = pygame.image.load('head.jpg')

steps = numpy.linspace(20, 360, 40).astype(int)
right = numpy.zeros((2, len(steps)))
down = numpy.zeros((2, len(steps)))
left = numpy.zeros((2, len(steps)))
up = numpy.zeros((2, len(steps)))

right[0] = steps
right[1] = 20

down[0] = 360
down[1] = steps

left[0] = steps[::-1]
left[1] = 360

up[0] = 20
up[1] = steps[::-1]

pos = numpy.concatenate((right.T, down.T, left.T, up.T))
i = 0
history = numpy.array([])
surf = plot(history)

while True:
    # Erase screen
    screen.fill((255, 255, 255))

    if i >= len(pos):
        i = 0
```

```
      surf = plot(history)


   screen.blit(img, pos[i])
   history = numpy.append(history, pos[i])
   screen.blit(surf, (100, 100))

   i += 1

   for event in pygame.event.get():
      if event.type == QUIT:
         pygame.quit()
         sys.exit()

   pygame.display.update()
   clock.tick(30)
```

How it works...

The plotting-related functions are explained in this table:

Function	Description
`matplotlib.use("Agg")`	This function specifies to use the non-interactive backend
`plt.figure(figsize=[3, 3])`	This function creates a figure of 3 by 3 inches
`fig.add_subplot(111)`	This function creates a subplot (in this case we only need 1 subplot)
`agg.FigureCanvasAgg(fig)`	This function creates a canvas in non-interactive mode
`ax.plot(data)`	This function creates a plot using specified data
`canvas.draw()`	This function draws on the canvas
`canvas.get_renderer()`	This function gets a renderer for the canvas

Accessing surface pixel data (Intermediate)

The Pygame `surfarray` module handles the conversion between Pygame `Surface` objects and NumPy arrays. As you may recall, NumPy can manipulate big arrays in a fast and efficient manner.

How to do it...

In this recipe we will tile a small image to fill the game screen.

1. **Copying pixels to array**: The `array2d` function copies pixels into a two-dimensional array. There is a similar function for three-dimensional arrays. We will copy the pixels from the avatar image into an array:

   ```
   pixels = pygame.surfarray.array2d(img)
   ```

2. **Creating the game screen**: A NumPy array has a shape attribute that corresponds to the dimensions of the array. This attribute is a tuple. A two-dimensional array for instance, will have a two-element shape tuple. Let's create the game screen from the shape of the pixels array using the `shape` attribute of the array. The screen will be seven times larger in both directions:

   ```
   X = pixels.shape[0] * 7
   Y = pixels.shape[1] * 7
   screen = pygame.display.set_mode((X, Y))
   ```

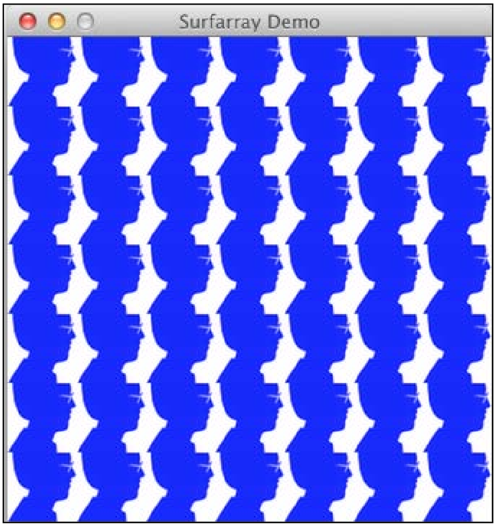
3. **Tiling the image**: Tiling the image is easy with the NumPy `tile` function. The data needs to be converted to integer values, since colors are defined as integers:

   ```
   new_pixels = numpy.tile(pixels, (7, 7)).astype(int)
   ```

4. **Displaying the array**: The `surfarray` module has the following special function (`blit_array`) to display the array on the screen:

   ```
   pygame.surfarray.blit_array(screen, new_pixels)
   ```

The following screenshot displays the result of the code:



The following code does the tiling of the image:

```
import pygame, sys
from pygame.locals import *
import numpy

pygame.init()
img = pygame.image.load('head.jpg')
pixels = pygame.surfarray.array2d(img)
X = pixels.shape[0] * 7
Y = pixels.shape[1] * 7
screen = pygame.display.set_mode((X, Y))
pygame.display.set_caption('Surfarray Demo')
new_pixels = numpy.tile(pixels, (7, 7)).astype(int)


while True:
   screen.fill((255, 255, 255))
   pygame.surfarray.blit_array(screen, new_pixels)

   for event in pygame.event.get():
      if event.type == QUIT:
         pygame.quit()
         sys.exit()

   pygame.display.update()
```

How it works...

The following table gives us a brief description of the new functions and attributes we used:

Function	Description
`pygame.surfarray.array2d(img)`	This copies pixel data into a 2D array
`pixels.shape[0]`	The `shape` attribute holds the dimensions of a NumPy array as a tuple
`numpy.tile(pixels, (7, 7))`	This tiles an array the given dimensions specified as a tuple
`pygame.surfarray.blit_array(screen, new_pixels)`	This displays array values on the screen

Accessing sound data (Simple)

A good game needs to have great music and sound effects. The Pygame `mixer` module lets us play a sound or any audio for that matter.

How to do it...

We will download a WAV audio file using standard Python. We will play this sound when the game quits. This example requires you to actually execute the example code, because this book has no audio support.

1. **Creating a sound object**: We can create a Pygame `Sound` object after specifying the name of the audio file. This class as you would expect embodies the concept of sounds:

   ```
   audio = pygame.mixer.Sound(WAV_FILE)
   ```

2. **Playing the sound**: The `Sound` object has a `play` method, which has a number of loops parameters. If the value of this parameter is set to -1, the sound will loop indefinitely:

   ```
   audio.play(-1)
   ```

3. **Pausing the game**: Sometimes we need to pause the execution of a game, as in our case in order to be able to hear a sound. We can do this with the following code snippet:

   ```
   pygame.time.delay(TIMEOUT * 1000)
   ```

   The delay is specified in milliseconds, that's why we are multiplying by 1000.

4. **Stopping the sound**: After a while we need to stop the sound with the corresponding `stop` method:

```
audio.stop()
```

The audio demo code is listed as follows:

```
import pygame, sys
from pygame.locals import *
import numpy
import urllib2
import time

WAV_FILE = 'smashingbaby.wav'

def play():
    audio = pygame.mixer.Sound(WAV_FILE)
    audio.play(-1)
    TIMEOUT = 1
    pygame.time.delay(TIMEOUT * 1000)
    audio.stop()
    time.sleep(TIMEOUT)

pygame.init()
pygame.display.set_caption('Sound Demo')
response = urllib2.urlopen('http://www.thesoundarchive.com/
austinpowers/smashingbaby.wav')
filehandle = open(WAV_FILE, 'w')
filehandle.write(response.read())
filehandle.close()
screen = pygame.display.set_mode((400, 400))

while True:
   sys_font = pygame.font.SysFont("None", 19)
   rendered = sys_font.render('Smashing Baby', 0, (255, 100, 100))
   screen.blit(rendered, (100, 100))

   for event in pygame.event.get():
      if event.type == QUIT:
         play()
         pygame.quit()
         sys.exit()

   pygame.display.update()
```

How it works...

The most important functions of this demo are summed up in the following table:

Function	Description
`pygame.mixer.Sound(WAV_FILE)`	This function creates a `Sound` object given a filename.
`audio.play(-1)`	This function plays and loops indefinitely (-1 means indefinitely). By default the sound is played only once. This corresponds with 0 loops. If the value is 2, the sound will be played once and then repeated 2 more times.
`pygame.time.delay(TIMEOUT * 1000)`	This function pauses the game for a specified number of milliseconds.
`audio.stop()`	This function stops audio playback.

Playing a movie (Intermediate)

Most commercial games these days have small movie clips that try to explain the plot to us. For instance, a first-person shooter could have a movie showing a briefing about the next mission. Movie playback is a cool feature to have. Pygame offers limited support for MPEG movies.

Getting ready

We need to have a MPEG movie for this demo. Once you have a movie you can convert it to be used in a Pygame game with the following command:

```
ffmpeg -i <infile> -vcodec mpeg1video -acodec libmp3lame -intra <outfile.mpg>
```

Installing `ffmpeg` and the command-line options are outside the scope of this book, but shouldn't be too difficult (see `http://ffmpeg.org/`).

How to do it...

The movie playback is set up similarly to the audio playback that we covered in the previous recipe. The following code demonstrates playing a MPEG video. Pay particular attention to the `play` function:

```
import pygame, sys
from pygame.locals import *
import time
```

```
pygame.init()
screen = pygame.display.set_mode((400, 400))
pygame.display.set_caption('Movie Demo')

def play():
    movie = pygame.movie.Movie('out.mpg')
    movie.play()
    TIMEOUT = 7
    pygame.time.delay(TIMEOUT * 1000)
    movie.stop()

while True:
   screen.fill((255, 255, 255))

   for event in pygame.event.get():
      if event.type == QUIT:
         play()
         pygame.quit()
         sys.exit()

   pygame.display.update()
```

How it works...

The relevant functions for the movie playback can found in this table:

Function	Description
`pygame.movie.Movie('out.mpg')`	This function creates a `Movie` object given the filename of the MPEG movie
`movie.play()`	This function starts playing the movie
`movie.stop()`	This function stops playback of the movie

Pygame on Android (Intermediate)

Android is an open source smartphone operating system initially developed by Google. Most of the Android apps are written in the Java programming language and run on a Java-based virtual machine. Fortunately, we can create Pygame games for Android phones. This is not a trivial matter and we will only cover the bare basics.

Getting ready

We will install the **Pygame Subset For Android** (**PGS4A**). You will need to have the JDK, Python 2.7 or a later version installed before we start. Download the appropriate software for your operating system from `http://pygame.renpy.org/dl`.

To install the necessary software, we will require an Internet connection and quite a lot of room on your hard drive. If you don't have a couple of gigabytes to spare, you may need to make more space. We can install the Android SDK and other software we will need such as Apache Ant by running the following command:

```
android.py installsdk
```

This will start a wizard that will guide you through the installation. It's safe to accept all the default options during the installation procedure, but you do have to generate a key. Unless you are really serious about creating apps, you don't have to worry how secure this key is.

How to do it...

We will create a simple game that prints "Hello World From Android!" and call it `mygame`.

1. **Setting up the game**: Create a directory with the same name as the name of the game and place a `main.py` file in there with the following contents:

```
import pygame

# Import the android module. If we can't import it, set it to None
- this
# lets us test it, and check to see if we want android-specific
# behavior.
try:
    import android
except ImportError:
    android = None

# Event constant.
TIMEREVENT = pygame.USEREVENT

# The FPS the game runs at.
FPS = 30

def main():
    pygame.init()

    # Set the screen size.
    screen = pygame.display.set_mode((480, 800))
```

```
    # Map the back button to the escape key.
    if android:
        android.init()
        android.map_key(android.KEYCODE_BACK, pygame.K_ESCAPE)

    # Use a timer to control FPS.
    pygame.time.set_timer(TIMEREVENT, 1000 / FPS)

    while True:
        ev = pygame.event.wait()

        # Android-specific:
        if android:
            if android.check_pause():
                android.wait_for_resume()

        # Draw the screen based on the timer.
        if ev.type == TIMEREVENT:
            screen.fill((255, 255, 255))
            font = pygame.font.Font('freesansbold.ttf', 32)
            rendered = font.render('Hello From Android!', 0, (255,
100, 100))
            screen.blit(rendered, (100, 100))
            pygame.display.flip()

        # When the user hits back, ESCAPE is sent. Handle it and
        # end the game.
        elif ev.type == pygame.KEYDOWN and ev.key == pygame.K_
ESCAPE:
            break

# This isn't run on Android.
if __name__ == "__main__":
    main()
```

This is basically the code from the PGS4A website changed to print a welcome message. A more thorough explanation will be given at the end of the recipe.

2. **Configuring the game**: We can configure the game with the following command:

   ```
   android.py configure mygame
   ```

   We will accept all the defaults and set the storage setting to internal.

3. **Building, installing, and running the game**: Android is essentially a Java framework, so there is a lot of compiling involved. This is a bit different than in the Python world. Since this game is simple, building will not take that long. First we will start the emulator—this is an application that mimics the behavior of an actual phone. Find the `android` executable that is part of the Android SDK. Launch it and choose **Tools** | **Manage AVDs...** | **New...** in the GUI application that opens. Create an **Android Virtual Device** (**AVD**) and give it a name. Hit the **Launch...** button. A phone emulator will start. If it is locked, you can unlock it by pressing *F2*.

   We can now build and install the game with the command:

   ```
   android.py build mygame release install
   ```

How it works...

The relevant functions used in this code are described as follows:

Function	Description
`android.init()`	This function initializes Android
`android.map_key(android.KEYCODE_BACK, pygame.K_ESCAPE)`	This function maps the Android back button to the Pygame escape button
`pygame.time.set_timer(TIMEREVENT, 1000 / FPS)`	This function fires events at specified time intervals given in milliseconds
`android.check_pause()`	This function checks for a pause request
`android.wait_for_resume()`	This function puts the game in sleep mode

Artificial intelligence (Intermediate)

Often we need to mimic intelligent behavior within a game. The `scikits-learn` project aims to provide an API for Machine Learning. What I like most about it is the amazing documentation.

Getting ready

We can install `scikit-learn` by typing the following command at the command line:

```
pip install -U scikit-learn
```

Or:

```
easy_install -U scikit-learn
```

This might not work because of permissions, so you might need to put `sudo` in front of the commands or log in as admin.

How to do it...

We will generate some random points and cluster them, which means that points that are close to each other are put in the same cluster. This is only one of the many techniques that you can apply with `scikits-learn`. **Clustering** is a type of machine learning algorithm that aims to group items based on similarities.

1. **Generating random points**: We will generate 30 random point positions within a square of 400 by 400 pixels:

   ```
   positions = numpy.random.randint(0, 400, size=(30, 2))
   ```

2. **Calculating the affinity matrix**: We will use the Euclidean distance to the origin as the **affinity metric**. The affinity matrix is a matrix holding affinity scores, in this case distances:

   ```
   positions_norms = numpy.sum(positions ** 2, axis=1)
   S = - positions_norms[:, numpy.newaxis] - positions_norms[numpy.
   newaxis, :] + 2 * numpy.dot(positions, positions.T)
   ```

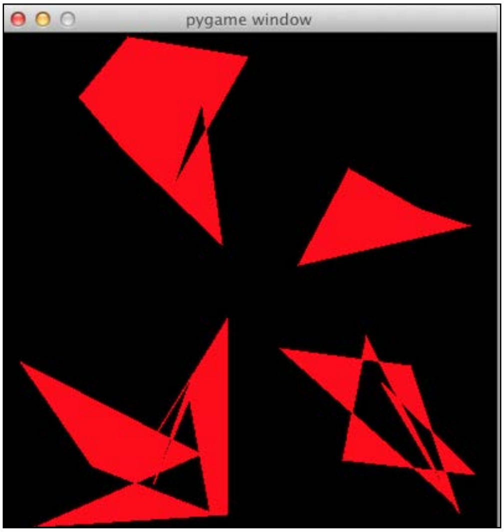
3. **Clustering the points**: Give the `AffinityPropagation` class the result from the previous step. This class labels the points with the appropriate cluster number:

   ```
   aff_pro = sklearn.cluster.AffinityPropagation().fit(S)

   labels = aff_pro.labels_
   ```

4. **Drawing polygons**: We will draw polygons for each cluster. The function involved requires a list of points, a color (let's paint it red), and a surface:

   ```
   pygame.draw.polygon(screen, (255, 0, 0), polygon_points[i])
   ```

The result is a bunch of polygons for each cluster as shown in the following screenshot:



The clustering example code is shown as follows:

```
import numpy
import sklearn.cluster
import pygame, sys
from pygame.locals import *


positions = numpy.random.randint(0, 400, size=(30, 2))

positions_norms = numpy.sum(positions ** 2, axis=1)
S = - positions_norms[:, numpy.newaxis] - positions_norms[numpy.
newaxis, :] + 2 * numpy.dot(positions, positions.T)

aff_pro = sklearn.cluster.AffinityPropagation().fit(S)
labels = aff_pro.labels_

polygon_points = []

for i in xrange(max(labels) + 1):
   polygon_points.append([])


# Sorting points by cluster
```

```
for i, l in enumerate(labels):
   polygon_points[l].append(positions[i])

pygame.init()
screen = pygame.display.set_mode((400, 400))


while True:
   for point in polygon_points:
      pygame.draw.polygon(screen, (255, 0, 0), point)

   for event in pygame.event.get():
      if event.type == QUIT:
         pygame.quit()
         sys.exit()

   pygame.display.update()
```

How it works...

The most important lines in the artificial intelligence recipe are described in more detail in the following table:

Function	Description
`numpy.random.randint(0, 400, size=(30, 2))`	This creates an array of 30 by 2 random integers. This corresponds to 30 points in two-dimensional space. The values are between 0 and 400.
`numpy.sum(positions ** 2, axis=1)`	This sums an array of the square of the positions array.
`numpy.dot(positions, positions.T)`	This computes the dot product of the positions array and its transpose.
`sklearn.cluster.AffinityPropagation().fit(S)`	This creates an `AffinityPropagation` object and performs a fit using an affinity matrix.
`pygame.draw.polygon(screen, (255, 0, 0), polygon_points[i])`	This draws a polygon given a surface, a color (red in this case), and a list of points.

Drawing sprites (Intermediate)

Sprite is a term from computer graphics meaning a two-dimensional visible object, that has been optimized for rendering. Pygame offers the `Sprite` class that deals with sprites. It can draw sprites on a `Surface` object. It also has collision functions. For complex games, we can group sprites together for easy management. Sprites are not thread safe, so you should take care when using multiple threads.

How to do it...

We will redo the animation demo, but this time with sprites and using `Rect` objects, which represent rectangles. A `Rect` object has `left`, `top`, `width`, and `height` attributes. We will use these and other attributes throughout the example. Also we will let the avatar spin when the mouse button is clicked. However, we will not care for now where we click exactly.

We will create a class that extends the `Sprite` class. Sprite classes have an `update` method which fires for each frame. All logic involving the movement of the sprite should be placed here.

1. **Constructor**: First, we need to create the sprite and perform subclassing. All the initialization logic goes here. Further details for the functions can be found in the next section. We define an image, rectangle, and variables tracking the movement of the avatar:

   ```
   class Head(pygame.sprite.Sprite):
       def __init__(self):
           pygame.sprite.Sprite.__init__(self)
           self.image, self.rect = load_image('head.jpg', -1)
           screen = pygame.display.get_surface()
           self.area = screen.get_rect()
           self.STEP = 9
           self.MARGIN = 12
           self.xstep = self.STEP
           self.ystep = 0
           self.dizzy = 0
           self.direction = 'right'
   ```

2. **The update method**: The `update` method calls helper methods that either cause the head to spin or move it in clockwise direction. The movement is achieved with this line:

   ```
   newpos = self.rect.move((self.xstep, self.ystep))
   ```

The following line take care of the rotation:

```
self.image = pygame.transform.rotate(self.original, self.degrees)
```

> You can find a short clip of the game on YouTube (`https://www.youtube.com/watch?v=EFQlc_siPrI`). A screenshot of the game is shown as follows:

The complete code of the Sprite demo is listed as follows:

```
import os, pygame
from pygame.locals import *

def load_image(name, colorkey=None):
    try:
        image = pygame.image.load(name)
    except pygame.error, message:
        print 'Cannot load image:', name

    image = image.convert()

    return image, image.get_rect()

class Head(pygame.sprite.Sprite):
    def __init__(self):
        pygame.sprite.Sprite.__init__(self)
        self.image, self.rect = load_image('head.jpg', -1)
        screen = pygame.display.get_surface()
        self.area = screen.get_rect()
        self.STEP = 9
        self.MARGIN = 12
        self.xstep = self.STEP
        self.ystep = 0
        self.degrees = 0
        self.direction = 'right'

    def update(self):
```

```
        if self.degrees:
            self._spin()
        else:
            self._move()

    def _move(self):
        newpos = self.rect.move((self.xstep, self.ystep))

        if self.direction == 'right' and self.rect.right > self.
area.right - self.MARGIN:
            self.xstep = 0
            self.ystep = self.STEP
            self.direction = 'down'

        if self.direction == 'down' and self.rect.bottom > self.
area.bottom - self.MARGIN:
            self.xstep = -self.STEP
            self.ystep = 0
            self.direction = 'left'

        if self.direction == 'left' and self.rect.left < self.
area.left + self.MARGIN:
            self.xstep = 0
            self.ystep = -self.STEP
            self.direction = 'up'

        if self.direction == 'up' and self.rect.top < self.area.
top + self.MARGIN:
            self.xstep = self.STEP
            self.ystep = 0
            self.direction = 'right'

        self.rect = newpos

    def _spin(self):
        center = self.rect.center
        self.degrees = self.degrees + 12
        if self.degrees >= 360:
            self.degrees = 0
            self.image = self.original
        else:
            self.image = pygame.transform.rotate(self.original,
self.degrees)
        self.rect = self.image.get_rect(center=center)
```

```
    def hit(self):
        if not self.degrees:
            self.degrees = 1
            self.original = self.image


def main():
    pygame.init()
    screen = pygame.display.set_mode((400, 400))
    pygame.display.set_caption('Sprite Demo')

    background = pygame.Surface(screen.get_size())
    background = background.convert()
    background.fill((250, 250, 250))

    if pygame.font:
        font = pygame.font.Font(None, 36)
        text = font.render("Hit the avatar!", 1, (0, 0, 200))
        textpos = text.get_rect(centerx = background.get_
width()/2, centery = background.get_height()/2)
        background.blit(text, textpos)

    screen.blit(background, (0, 0))
    pygame.display.flip()

    clock = pygame.time.Clock()
    head = Head()
    sprite = pygame.sprite.RenderPlain(head)

    while True:
        clock.tick(60)

        for event in pygame.event.get():
            if event.type == QUIT:
                return
            elif event.type == MOUSEBUTTONDOWN:
               head.hit()

        sprite.update()

        screen.blit(background, (0, 0))
        sprite.draw(screen)
        pygame.display.flip()
```

```
    if __name__ == '__main__':
       main()
```

How it works...

A more in-depth description of the various functions used in this demo is given as follows:

Function	Description
`pygame.sprite.Sprite.__init__(self)`	This creates sprites.
`screen.get_rect()`	This gets a `Rect` object.
`pygame.display.get_surface()`	This gets a `Surface` object.
`self.rect.move((self.xstep, self.ystep))`	This moves a rectangle given a `x` and `y` coordinate.
`pygame.transform.rotate(self.original, self.degrees)`	This rotates an image given a `Surface` object and angle in degrees. Positive values correspond with counter clockwise rotation, negative with clockwise rotation.
`self.image.get_rect(center=center)`	This gets the rectangle for the image given its center coordinates.
`pygame.sprite.RenderPlain(head)`	This renders the sprite.

Using OpenGL with Pygame (Advanced)

OpenGL specifies an API for 2D and 3D computer graphics. The API consists of functions and constants. We will be concentrating on the Python implementation called **PyOpenGL**.

Getting ready

Install PyOpenGL with the following command:

```
pip install PyOpenGL PyOpenGL_accelerate
```

You might need to have root access to execute this command. The corresponding `easy_install` command is as follows:

```
easy_install PyOpenGL PyOpenGL_accelerate
```

How to do it...

For the purpose of demonstration we will draw a Sierpinski gasket with OpenGL. This is a fractal pattern in the shape of a triangle created by the mathematician Waclaw Sierpinski. The triangle is obtained via a recursive and in principle infinite procedure.

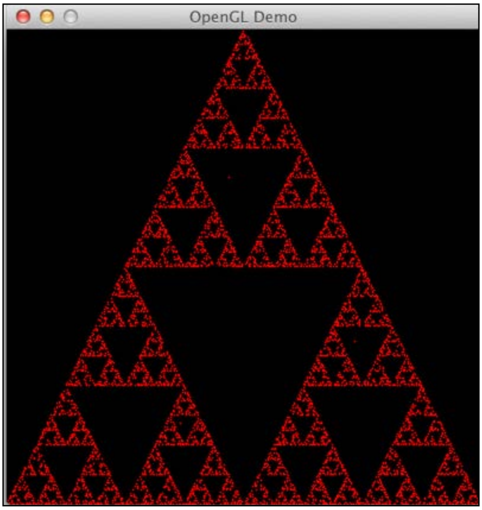
1. **OpenGL Initialization**: First, we will start out by initializing some of the OpenGL-related primitives. This includes setting the display mode and background color. A line-by-line explanation is given at the end of the recipe:

```
def display_openGL(w, h):
  pygame.display.set_mode((w,h), pygame.OPENGL|pygame.DOUBLEBUF)

  glClearColor(0.0, 0.0, 0.0, 1.0)
  glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT)

  gluOrtho2D(0, w, 0, h)
```

2. **Displaying points**: The algorithm requires us to display points, the more the better. First, we set the drawing color to red. Second, we define the vertices (I call them points myself) of a triangle. Then we define random indices, which are to be used to choose one of the three triangle vertices. We pick a random point somewhere in the middle, it doesn't really matter where. After that we draw points halfway between the previous point and one of the vertices picked at random. Finally, we "flush" the result:

```
glColor3f(1.0, 0, 0)
vertices = numpy.array([[0, 0], [DIM/2, DIM], [DIM, 0]])
NPOINTS = 9000
indices = numpy.random.random_integers(0, 2, NPOINTS)
point = [175.0, 150.0]

for index in indices:
   glBegin(GL_POINTS)
   point = (point + vertices[index])/2.0
   glVertex2fv(point)
   glEnd()

glFlush()
```

The Sierpinski triangle looks like this:



The full Sierpinski gasket demo code with all the imports is shown as follows:

```
import pygame
from pygame.locals import *
import numpy

from OpenGL.GL import *
from OpenGL.GLU import *

def display_openGL(w, h):
  pygame.display.set_mode((w,h), pygame.OPENGL|pygame.DOUBLEBUF)

  glClearColor(0.0, 0.0, 0.0, 1.0)
  glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT)


  gluOrtho2D(0, w, 0, h)


def main():
    pygame.init()
    pygame.display.set_caption('OpenGL Demo')
    DIM = 400
    display_openGL(DIM, DIM)
    glColor3f(1.0, 0, 0)
    vertices = numpy.array([[0, 0], [DIM/2, DIM], [DIM, 0]])
```

```
    NPOINTS = 9000
    indices = numpy.random.random_integers(0, 2, NPOINTS)
    point = [175.0, 150.0]

    for index in indices:
       glBegin(GL_POINTS)
       point = (point + vertices[index])/2.0
       glVertex2fv(point)
       glEnd()

    glFlush()
    pygame.display.flip()

    while True:
        for event in pygame.event.get():
            if event.type == QUIT:
                return


if __name__ == '__main__':
  main()
```

How it works...

As promised here is a line-by-line explanation of the most important parts of the example:

Function	Description
`pygame.display.set_mode((w,h), pygame.OPENGL|pygame.DOUBLEBUF)`	This sets the display mode to the required width, height, and OpenGL display.
`glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT)`	This clears the buffers using a mask. Here we clear the color buffer and depth buffer bits.
`gluOrtho2D(0, w, 0, h)`	This defines a 2D orthographic projection matrix with the coordinates of the left, right, top, and bottom clipping planes.
`glColor3f(1.0, 0, 0)`	This defines the current drawing color using three float values for RGB (0-1 instead of 0-255 that is usual for Pygame). In this case we will be painting in red.

Function	Description
`glBegin(GL_POINTS)`	This delimits the vertices of primitives or a group of primitives. Here the primitives are points.
`glVertex2fv(point)`	This renders a point given a vertex.
`glEnd()`	This closes a section of code started with `glBegin`.
`glFlush()`	This forces execution of GL commands.

Detecting collisions (Intermediate)

In the sprite demo, we left out the collision detection bit. Pygame has a number of useful collision detection functions in the `Rect` class. For instance, we can check whether a point is in a rectangle or whether two rectangles overlap.

How to do it...

Beside the collision detection we will replace the mouse cursor with an image of a hammer that we created. It's not a very pretty image, but it beats the boring old cursor.

1. **Updating the hit method**: We will update the `hit` method of the sprite demo code. In the new version, we check whether the mouse cursor is within the avatar sprite. Actually to make it easier to hit the head, we create a slightly bigger rectangle:

```
def hit(self):
         mouse_x, mouse_y = pygame.mouse.get_pos()
         collided = False
         bigger_rect = self.rect.inflate(40, 40)

         if bigger_rect.collidepoint(mouse_x, mouse_y):
            collided = True

         if not self.degrees and collided:
            self.degrees = 1
            self.original = self.image
            self.nhits += 1
         else:
            self.nmisses += 1
```

2. **Replacing the mouse cursor**: All the steps necessary to replace the mouse cursor were already covered. Except making the mouse cursor invisible:

```
pygame.mouse.set_visible(False)
```

A screenshot of the game is shown as follows:

The complete code for this example can be found in the code bundle of this book.

How it works...

We learned a bit about collision detection, the mouse cursor, and rectangles in this recipe:

Function	Description
`pygame.mouse.get_pos()`	This gets the mouse position as a tuple.
`self.rect.inflate(40, 40)`	This creates a bigger rectangle based on an offset. If the offset is negative this results in a smaller rectangle.
`bigger_rect.collidepoint(mouse_x, mouse_y)`	This checks whether a point is within a rectangle.
`pygame.mouse.set_visible(False)`	This hides the mouse cursor.

Adding networking functionality (Advanced)

Games become more engaging when you are able to play against other people. Usually this means playing over the Internet using some sort of client-server architecture. In the Python world, Twisted is commonly used for this kind of architecture.

Getting ready

Twisted can be installed in several ways depending on your operating system. For more information see `https://twistedmatrix.com/trac/wiki/Downloads`.

How to do it...

Unfortunately, we cannot create a massive multiplayer game in this tutorial, but we can create a simple client-server setup, which will lay the foundations for a puzzle we will create in a later recipe.

1. **The server**: First, we will set up the server, which will echo the message from the client and prepend it with a sequence number:

    ```
    from twisted.internet import reactor, protocol

    class Server(protocol.Protocol):
        def __init__(self):
           self.count = 0

        def dataReceived(self, msg):
             self. count += 1
             self.transport.write("%d %s" % (self.count, msg))


    def main():
        factory = protocol.ServerFactory()
        factory.protocol = Server
        reactor.listenTCP(8888,factory)
        reactor.run()

    if __name__ == '__main__':
        main()
    ```

    As you can see the server runs on port 8888 over TCP (see `http://en.wikipedia.org/wiki/Transmission_Control_Protocol`).

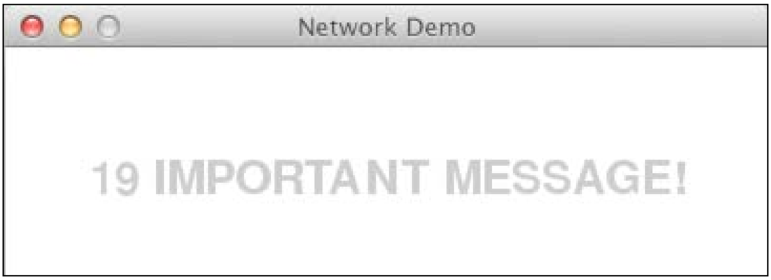
2. **Client setup**: The client sends messages over the same port as the server and also shows the messages from the server in a Pygame GUI. We will go over the details in the next section. In a later example we will do more interesting things with this code:

    ```
    from twisted.internet import reactor, protocol
    from pygame.locals import *
    import pygame

    class Client(protocol.Protocol):
    ```

```
    def __init__(self):
       self.msg = 'Hello'
       self.end_msg = False

    def sendMessage(self, msg):
        self.transport.write(msg)
        self.update(msg)

    def dataReceived(self, msg):
       self.msg = msg

       if msg.startswith("19"):
          self.end_msg = True

    def update(self, msg):
        screen = pygame.display.get_surface()

        screen.fill((255, 255, 255))
        font = pygame.font.Font(None, 36)
        text = font.render(self.msg, 1, (200, 200, 200))
        textpos = text.get_rect(centerx=screen.get_width()/2,
centery=screen.get_height()/2)
        screen.blit(text, textpos)
        pygame.display.flip()

        if self.end_msg:
           reactor.stop()

def send(p):
    p.sendMessage("Hello!")

    for i in xrange(1, 20):
      reactor.callLater(i * .1, p.sendMessage, "IMPORTANT
MESSAGE!")

def main():
    pygame.init()
    screen = pygame.display.set_mode((400, 400))
    pygame.display.set_caption('Network Demo')

    c = protocol.ClientCreator(reactor, Client)
    c.connectTCP("localhost", 8888).addCallback(send)
    reactor.run()
```

```
        while True:
            for event in pygame.event.get():
                if event.type == QUIT:
                    return

if __name__ == '__main__':
    main()
```

We need to start the server, before we can start the client. In the game GUI, you should see **Hello** being displayed followed by **1 IMPORTANT MESSAGE!** to **19 IMPORTANT MESSAGE!** as shown in the following screenshot:



How it works...

We saw in this example how to create a simple server and client with a Pygame GUI. In principle, we can now extend this setup to create a multiplayer game. The details of the Twisted client and server setup are given as follows:

Function	Description
`self.transport.write("%d %s" % (self.count, msg))`	This writes a message. In this case we are prepending a sequence number to the message.
`factory = protocol.ServerFactory()`	This creates a Twisted server factory, which itself creates Twisted servers.
`reactor.listenTCP(8888,factory)`	This listens to port 8888 using the given factory.
`reactor.run()`	This starts the server or client.

Function	Description
`reactor.stop()`	This stops the client or server.
`reactor.callLater(i * .1, p.sendMessage, "IMPORTANT MESSAGE!")`	This registers a callback function with a parameter to be executed after a specified time in seconds.
`protocol.ClientCreator(reactor, Client)`	This creates a Twisted client.
`c.connectTCP("localhost", 8888).addCallback(send)`	This connects the client via TCP on port 8888 and registers a callback function.

Debugging your game (Intermediate)

Debugging is one of those things that nobody really likes, but is very important to master. It can take hours, and because of Murphy's law you, most likely, don't have that time. Therefore, it is important to be systematic and know your tools well. After you are done finding the bug and implementing a fix, you should have a test in place. This way at least you will not have to go through the hell of debugging again.

PuDB is a visual full screen, console-based Python debugger that is easy to install. PuDB supports cursor keys and vi commands. The debugger can also be integrated with IPython, if required.

Getting ready

In order to install puDB, we only need to execute the following command:

```
sudo easy_install pudb
```

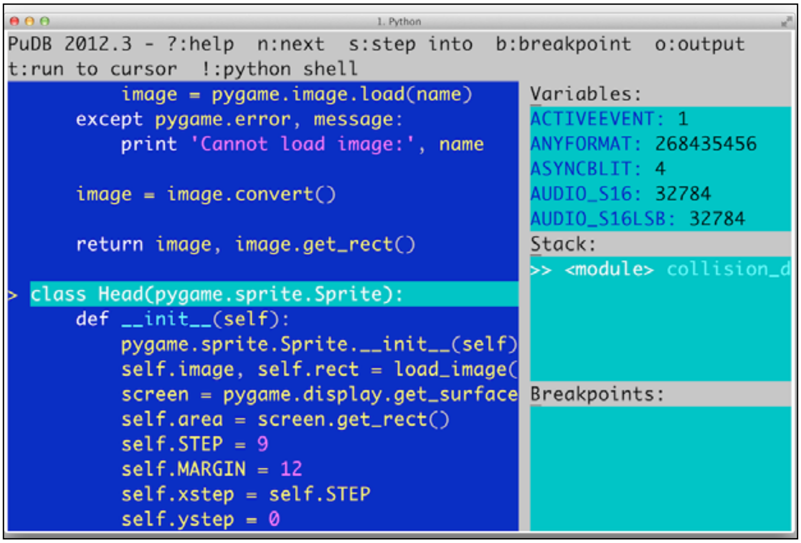
How to do it...

To debug the collision demo code, type the following command on the command line:

```
python -m pudb collision_demo.py
```

> The source code should be available for download from the Packt Publishing website.

The following screenshot shows the most important debugging commands at the top:



We can also see the code being debugged, variables, the stack, and the defined breakpoints. Typing `q` exits most menus. Typing `n` moves the debugger to the next line. We can also move with the cursor keys or vi *J* and *K* keys to, for instance, set a breakpoint by typing `b`.

Profiling your code (Intermediate)

Performance is important for games, luckily there are many Python profiling tools. Profiling is about building a profile of a software program in order to collect information about memory usage or time complexity.

`cProfile` is a C extension introduced in Python 2.5. It can be used for **deterministic profiling**. Deterministic profiling means that the time measurements are precise and no sampling is used. Contrast this with statistical profiling, where measurements come from random samples.

How to do it...

The following steps will help you profile your code:

1. **Creating a profile file**: We will profile the collision demo code and store the profile output in a file as follows:

   ```
   python -m cProfile -o collision_demo.profile collision_demo.py
   ```

2. **The pstats browser**: After creating the file, we can view and sort the data in a special command-line browser:

```
python -m pstats collision_demo.profile
Welcome to the profile statistics browser.
```

3. **Getting help**: Being able to get help is always a good thing, just type the following commands at the command line:

```
collision_demo.profile% help


Documented commands (type help <topic>):
========================================
EOF  add  callees  callers  help  quit  read  reverse  sort  stats
strip
```

4. **Sorting**: We can sort with the following `sort` command:

```
collision_demo.profile% sort
Valid sort keys (unique prefixes are accepted):
stdname -- standard name
nfl -- name/file/line
pcalls -- call count
file -- file name
calls -- call count
time -- internal time
line -- line number
cumulative -- cumulative time
module -- file name
name -- function name
```

5. **Top 3 called functions**: We can get the top 3 called functions by sorting and calling `stats`:

```
collision_demo.profile% sort calls
collision_demo.profile% stats 3

         380943 function calls (380200 primitive calls) in 18.056
seconds


   Ordered by: call count
   List reduced from 801 to 3 due to restriction <3>
```

```
   ncalls  tottime  percall  cumtime  percall
filename:lineno(function)
    52156    0.013    0.000    0.013    0.000 {method 'endswith'
of 'str' objects}
31505/31368    0.003    0.000    0.003    0.000 {len}
    27573    0.022    0.000    0.022    0.000 {method 'lower' of
'str' objects}
```

How it works...

We profiled the collision demo. The following table summarizes the profiler output:

Column	Description
`Ncalls`	Number of calls
`Tottime`	Total time spent in a function
`Percall`	Time per call, calculated by dividing the total time by the calls count
`Cumtime`	Cumulative time spent in function and functions called by the function, including recursive calls

Puzzle game with Pygame (Advanced)

We will pick up where we left in the networking example. This time we will create a puzzle game that lets us guess a word. This is just a prototype mind you. It still needs a lot of polishing.

How to do it...

The following steps will help you to create the intended puzzle game:

1. **Server changes**: The changes in the server are pretty trivial. We just check whether we guessed the correct word:

```
from twisted.internet import reactor, protocol

class Server(protocol.Protocol):
    def dataReceived(self, msg):
         resp = '*' * 20
         print msg

         if msg == 'secret':
            resp = msg

         self.transport.write(resp)
```

```
def main():
    factory = protocol.ServerFactory()
    factory.protocol = Server
    reactor.listenTCP(8888,factory)
    reactor.run()

if __name__ == '__main__':
    main()
```

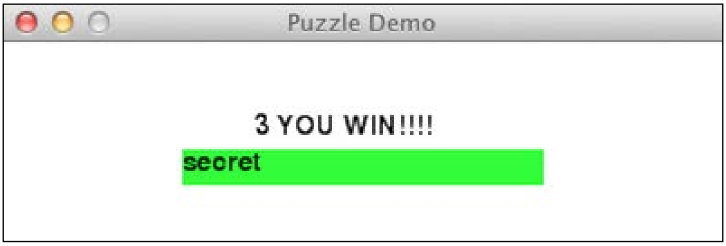
2. **Client changes**: The most important changes are the handling of key presses in an input box and handling of the response from the server. The input box lets us type text, edit it with the *Backspace* key, and submit with the *Enter* key. A label above the textbox displays the number of attempts and the game status. We use a Twisted looping callback to update the GUI every 30 milliseconds:

```
from twisted.internet import reactor, protocol
from pygame.locals import *
import pygame
from twisted.internet.task import LoopingCall

class Client(protocol.Protocol):
    def __init__(self):
       self.STARS = '*' * 20
       self.msg = self.STARS
       self.font = pygame.font.Font(None, 22)
       self.screen = pygame.display.get_surface()
       self.label = 'Guess the word:'
       self.attempts = 0


    def sendMessage(self, msg):
        self.transport.write(msg)

    def dataReceived(self, msg):
       self.msg = msg

       if self.msg != self.STARS:
         self.label = 'YOU WIN!!!!'

       self.update_prompt()

    def update_prompt(self):
```

```
        self.screen.fill((255, 255, 255))
        BG = (0, 255, 0)
        FG = (0, 0, 0)

        pygame.draw.rect(self.screen, BG, (100, 200, 200, 20))

        self.screen.blit(self.font.render(self.msg, 1, FG), (100,
200))
        self.screen.blit(self.font.render("%d %s" % (self.
attempts, self.label), 1, FG),
              (140, 180))
        pygame.display.flip()

def handle_events(p):
   while True:
      for event in pygame.event.get():
         if event.type == QUIT:
            reactor.stop()
            return
         elif event.type == KEYDOWN:
            key = event.key

            if p.msg == '*' * 20:
               p.msg = ''

            if key == K_BACKSPACE:
               p.msg = p.msg[0:-1]
               p.update_prompt()
            elif key == K_RETURN:
               p.attempts += 1
               p.sendMessage(p.msg)
               return
            elif ord('a') <= key <= ord('z'):
               p.msg += chr(key)
               p.update_prompt()

def send(p):
   p.update_prompt()
   tick = LoopingCall(handle_events, p)
   tick.start(.03)


def main():
    pygame.init()
```

```
    screen = pygame.display.set_mode((400, 400))
    pygame.display.set_caption('Puzzle Demo')

    c = protocol.ClientCreator(reactor, Client)
    c.connectTCP("localhost", 8888).addCallback(send)
    reactor.run()

if __name__ == '__main__':
    main()
```

The following screenshot was taken after guessing the word:



How it works...

Although this seems to be a pretty extensive recipe, only a few lines of the code might require some explanation:

Function	Description
`LoopingCall(handle_events, p)`	This creates a looping callback. A callback function that is called periodically.
`tick.start(.03)`	This starts the looping callback with a period of 30 milliseconds.

Simulating with Pygame (Advanced)

As the last example, we will simulate life with Conway's Game of Life. The original game of life is based on a few basic rules. We start out with a random configuration on a two-dimensional square grid. Each cell in the grid can be either dead or alive. This state depends on the eight neighbors of the cell. Convolution can be used to evaluate the basic rules of the game. We will need the SciPy package for the convolution bit.

Getting ready

Install SciPy with either of the following two commands:

- `sudo pip install scipy`
- `easy_install scipy`

How to do it...

The following code is an implementation of Game of Life with some modifications:

- Clicking once with the mouse draws a cross until we click again
- The *R* key resets the grid to a random state
- Pressing *B* creates blocks based on the mouse position
- *G* creates gliders

The most important data structure in the code is a two-dimensional array holding the color values of the pixels on the game screen. This array is initialized with random values and then recalculated in the game loop. More information about the involved functions can be found in the next section. As previously mentioned, the following is the code:

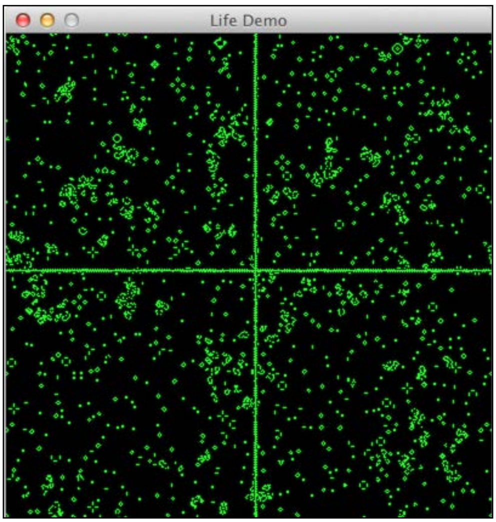
```
import os, pygame
from pygame.locals import *
import numpy
from scipy import ndimage

def get_pixar(arr, weights):
  states = ndimage.convolve(arr, weights, mode='wrap')

  bools = (states == 13) | (states == 12 ) | (states == 3)

  return bools.astype(int)

def draw_cross(pixar):
   (posx, posy) = pygame.mouse.get_pos()
   pixar[posx, :] = 1
   pixar[:, posy] = 1

def random_init(n):
   return numpy.random.random_integers(0, 1, (n, n))

def draw_pattern(pixar, pattern):
     print pattern
```

```
      if pattern == 'glider':
       coords = [(0,1), (1,2), (2,0), (2,1), (2,2)]
      elif pattern == 'block':
       coords = [(3,3), (3,2), (2,3), (2,2)]
      elif pattern == 'exploder':
       coords = [(0,1), (1,2), (2,0), (2,1), (2,2), (3,3)]
      elif pattern == 'fpentomino':
       coords = [(2,3),(3,2),(4,2),(3,3),(3,4)]


      pos = pygame.mouse.get_pos()

      xs = numpy.arange(0, pos[0], 10)
      ys = numpy.arange(0, pos[1], 10)

      for x in xs:
         for y in ys:
            for i, j in coords:
                pixar[x + i, y + j] = 1


def main():
    pygame.init ()

    N = 400
    pygame.display.set_mode((N, N))
    pygame.display.set_caption("Life Demo")

    screen = pygame.display.get_surface()

    pixar = random_init(N)
    weights = numpy.array([[1,1,1], [1,10,1], [1,1,1]])

    cross_on = False

    while True:
       pixar = get_pixar(pixar, weights)

       if cross_on:
          draw_cross(pixar)

       pygame.surfarray.blit_array(screen, pixar * 255 ** 3)
       pygame.display.flip()
```

```
        for event in pygame.event.get():
          if event.type == QUIT:
              return
          if event.type == MOUSEBUTTONDOWN:
             cross_on = not cross_on
          if event.type == KEYDOWN:
             if event.key == ord('r'):
                pixar = random_init(N)
                print "Random init"
             if event.key == ord('g'):
                draw_pattern(pixar, 'glider')
             if event.key == ord('b'):
                draw_pattern(pixar, 'block')
             if event.key == ord('e'):
                draw_pattern(pixar, 'exploder')
             if event.key == ord('f'):
                draw_pattern(pixar, 'fpentomino')


if __name__ == '__main__':
    main()
```

You should be able to view a screencast on YouTube at `https://www.youtube.com/watch?v=NNsU-yWTkXM`.

A screenshot of the game in action is shown as follows:



How it works...

We used some NumPy and SciPy functions that need explaining:

Function	Description
`ndimage.convolve(arr, weights, mode='wrap')`	This applies the convolve operation on the given array, using weights in wrap mode. The mode has to do with the array borders. See `http://en.wikipedia.org/wiki/Convolution` for the mathematical details.
`bools.astype(int)`	This converts the array of Booleans to integers.
`numpy.arange(0, pos[0], 10)`	This creates an array from `0` to `pos[0]` in steps of 10. So if `pos[0]` is equal to 1000, we will get 0, 10, 20 ... 990.

PACKT PUBLISHING

Thank you for buying Instant Pygame for Python Game Development How-to

About Packt Publishing

Packt, pronounced 'packed', published its first book "*Mastering phpMyAdmin for Effective MySQL Management*" in April 2004 and subsequently continued to specialize in publishing highly focused books on specific technologies and solutions.

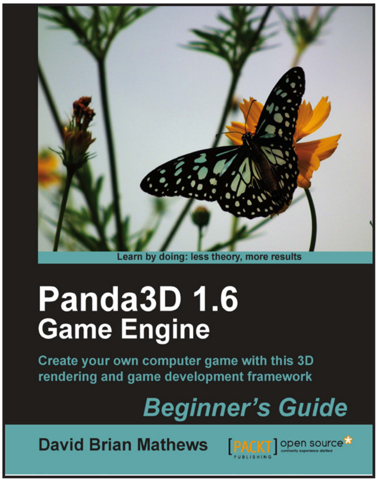
Our books and publications share the experiences of your fellow IT professionals in adapting and customizing today's systems, applications, and frameworks. Our solution based books give you the knowledge and power to customize the software and technologies you're using to get the job done. Packt books are more specific and less general than the IT books you have seen in the past. Our unique business model allows us to bring you more focused information, giving you more of what you need to know, and less of what you don't.

Packt is a modern, yet unique publishing company, which focuses on producing quality, cutting-edge books for communities of developers, administrators, and newbies alike. For more information, please visit our website: `www.packtpub.com`.

Writing for Packt

We welcome all inquiries from people who are interested in authoring. Book proposals should be sent to `author@packtpub.com`. If your book idea is still at an early stage and you would like to discuss it first before writing a formal book proposal, contact us; one of our commissioning editors will get in touch with you.

We're not just looking for published authors; if you have strong technical skills but no writing experience, our experienced editors can help you develop a writing career, or simply get some additional reward for your expertise.



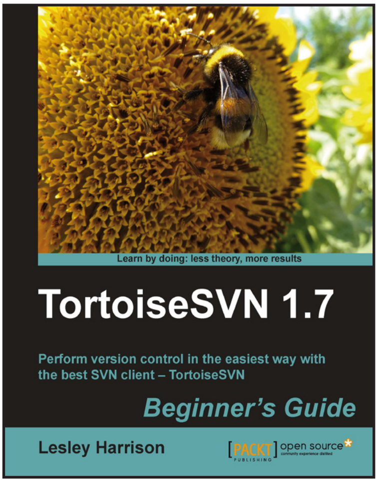
Panda3D 1.6 Game Engine Beginner's Guide

ISBN: 978-1-84951-272-5 Paperback: 356 pages

Create your own computer game with this 3D rendering and game development framework

1. The first and only guide to building a finished game using Panda3D
2. Learn about tasks that can be used to handle changes over time
3. Respond to events like keyboard key presses, mouse clicks, and more
4. Take advantage of Panda3D's built-in shaders and filters to decorate objects with gloss, glow, and bump effects

---



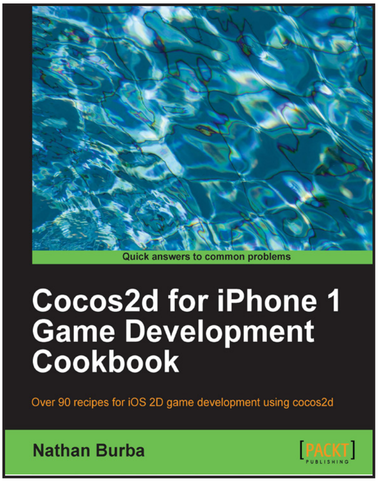
TortoiseSVN 1.7 Beginner's Guide

ISBN: 978-1-84951-344-9 Paperback: 260 pages

Perform version control in the easiest way with the best SVN client - TortoiseSVN

1. Master version control techniques with TortoiseSVN without the need for boring theory
2. Revolves around a real-world example based on a software company
3. The first and the only book that focuses on version control with TortoiseSVN
4. Reviewed by Stefan Kung, lead developer for the TortoiseSVN project

Cocos2d for iPhone 1 Game Development Cookbook



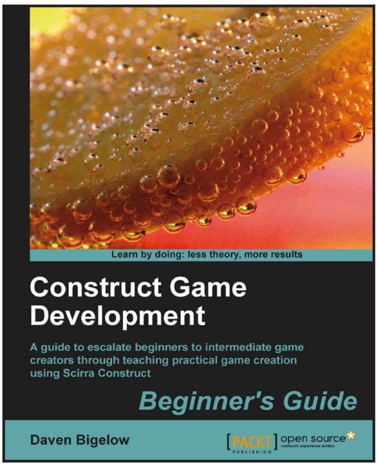
ISBN: 978-1-84951-400-2 Paperback: 446 pages

Over 90 recipes for iOS 2D game development using cocos2d

1. Discover advanced Cocos2d, OpenGL ES, and iOS techniques spanning all areas of the game development process
2. Learn how to create top-down isometric games, side-scrolling platformers, and games with realistic lighting
3. Full of fun and engaging recipes with modular libraries that can be plugged into your projectl

---

Construct Game Development Beginners Guide



ISBN: 978-1-84951-660-0 Paperback: 298 pages

A guide to escalate beginners to intermediate game creators through teaching practical game creation using Scirra Construct

1. Learn the skills necessary to make your own games through the creation of three very different sample games
2. Create animated sprites, use built-in physics and shadow engines of Construct Classic
3. A wealth of step-by-step instructions and images to lead the way

Please check **www.PacktPub.com** for information on our titles