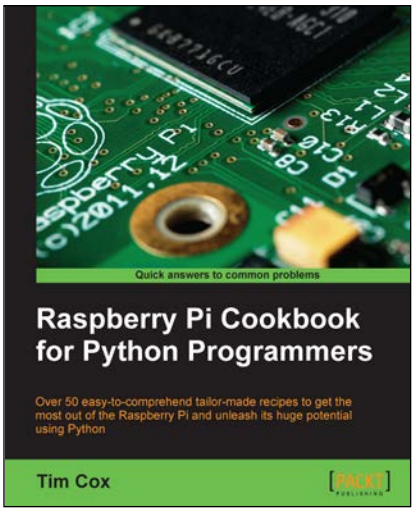
[PACKT] PUBLISHING

Raspberry Pi Cookbook for Python Programmers

Tim Cox



Chapter No. 4

"Creating Games and Graphics"

In this package, you will find:

A Biography of the author of the book

A preview chapter from the book, Chapter NO.4 "Creating Games and Graphics"

A synopsis of the book’s content

Information on where to buy this book

About the Author

Tim Cox lives in England with his wife and two young daughters and works as a software engineer. His passion for programming can be traced back to one Christmas in the mid 1980s when he received a Sinclair Spectrum 48k+ home computer (a joint present with his two elder brothers). By typing out and modifying BASIC programs, while dreaming about building robots, an interest in computers and electronics was sparked, which has never faded. This interest saw him through university, where he earned a BEng in Electronics and Electrical Engineering, and into a career in developing embedded software for a wide range of applications, for automotive, aerospace, and the oil industry, among others.

Keen to support the vision behind the Raspberry Pi, reignite engineering in schools, and encourage a new generation of engineers, Tim co-founded the MagPi magazine. Thanks to the dedication and time of the volunteers who contribute to it every month, it continues to have monthly issues and attract an ever-increasing number of readers (and writers) worldwide. Through his site `PiHardware.com`, Tim produces electronic kits and helps people learn about the Raspberry Pi and hardware interfacing; each of them is supported with detailed instructions and tutorials to help novices build the knowledge and skills for their projects.

Writing a book about the Raspberry Pi wouldn't have been possible without the creation itself, so thanks to the Raspberry Pi foundation for their hard work (and good humor) in making it a huge success. The Raspberry Pi community consists of an excellent group of exceptionally helpful people from all over the world, and it has been a pleasure to be involved with it from the start. In particular, I would like to thank The MagPi team that has supported me by reviewing the chapters and helping me achieve the best possible standard. Also thanks to the Pi3D team who worked hard to get their library running with Python 3 for the book.

Thanks to my family, particularly my wife Kirsty, who has supported me every step of the way and daily suffered my obsession with the Raspberry Pi. The excitement my daughters, Phoebe and Amelia, have as they discover new things inspires me to share and teach as much as I can.

Raspberry Pi Cookbook for Python Programmers

Since the release of the Raspberry Pi computer in February 2012, hundreds of thousands of people have been introduced to a new way of computing. Modern home computers, tablets, and phones are typically focused on providing content to the user to consume, either as a passive viewer or through basic interaction via games and activities.

However, the Raspberry Pi turns this concept on its head. The idea is that the user provides the input and the imagination, and the Raspberry Pi becomes an extension of their creativity. The Raspberry Pi provides a simple, low-cost platform that you can use to experiment with and play with your own ideas. It won't feed you information; it will let you discover it firsthand.

This book takes everything I have found exciting and interesting with the Raspberry Pi and puts it in an easy-to-follow format.

I hope that people will read this book and start their own Raspberry Pi journey; it has so much to offer, and the book is aimed squarely at showing off what *you* can achieve with it.

Like any good cookbook, the pages should be worn and used, and it should be something that is always being pulled off the shelf to refer to. I hope it will become your own, personal, go-to reference.

What This Book Covers

Chapter 1, *Getting Started with a Raspberry Pi Computer*, introduces the Raspberry Pi and explores the various ways that it can be set up and used, including how it can be used on a network and connected to remotely with another computer.

Chapter 2, *Starting with Python Strings, Files, and Menus*, guides us on how to take our first steps using Python 3, start with the basics, manipulate text, use files, and create menus to run our programs.

Chapter 3, *Using Python for Automation and Productivity*, explains the use of graphical user interfaces to create our own applications and utilities.

Chapter 4, *Creating Games and Graphics*, explains how to create a drawing application and graphical games using the Tkinter Canvas.

Chapter 5, *Creating 3D Graphics*, discusses how we can use the hidden power of the Raspberry Pi's graphical processing unit to learn about 3D graphics and landscapes and produce our very own 3D maze for exploration.

Chapter 6, *Using Python to Drive Hardware*, establishes the fact that to experience the Raspberry Pi at its best, we really have to use it with our own electronics. It discusses how to create circuits with LEDs and switches, and use them to indicate the system status and provide control. Finally, it shows us how to create our own game controller and light display.

Chapter 7, *Sense and Display Real-world Data*, explains the use of an analog-to-digital convertor to provide sensor readings to the Raspberry Pi. We discover how to store and graph the data in real time as well as display it on an LCD text display. Finally, we transfer the data to the Internet, which will allow us to view and share the captured data anywhere in the world.

Chapter 8, *Creating Projects with the Raspberry Pi Camera Module*, teaches us how to use the Raspberry Pi camera module, creating our own applications to produce time-lapse videos, stop-frame animations, and a bedtime book reader controlled with QR codes.

Chapter 9, *Building Robots*, takes you through building two different types of robots (a Rover- Pi and a Pi-Bug). We look at motor and servo control, using sensors, and adding a compass sensor for navigation.

Chapter 10, *Interfacing with Technology*, teaches us how to use the Raspberry Pi to trigger remote mains sockets, with which we can control household appliances. We learn how to communicate with the Raspberry Pi over a serial interface and use a smartphone to control everything using Bluetooth. Finally, we look at creating our own applications to control USB devices.

Appendix, *Hardware and Software List*, provides us with the full list of the hardware components and modules used in the book, along with suitable places to purchase them from. A full list of the software used is also provided, along with links to documentation.

4

Creating Games and Graphics

In this chapter, we will cover:

- Using IDLE3 to debug your programs
- Drawing lines using a mouse on a Tkinter Canvas
- Creating a bat and ball game
- Creating an overhead scrolling game

Introduction

Games are often a great way to explore and extend your programming skills as they present an inherent motivating force to modify and improve your creation, add new features, and create new challenges. They are also great for sharing your ideas with others, even if they aren't interested in programming.

This chapter focuses on using the Tkinter Canvas widget to create and display objects on screen for the user to interact with. Using these techniques, a wide variety of games and applications can be created that are limited only by your own creativity.

We also take a quick look at using the debugger built into IDLE, a valuable tool to test and develop your programs without the need to write extensive test code.

The first example demonstrates how we can monitor and make use of the mouse to create objects and draw directly on the `Canvas` widget. Next, we create a bat and ball game, which shows how the positions of objects can be controlled and how interactions between them can be detected and responded to. Finally, we take things a little further and use Tkinter to place our own graphics onto the `Canvas` widget to create an overhead view treasure hunt game.

Using IDLE3 to debug your programs

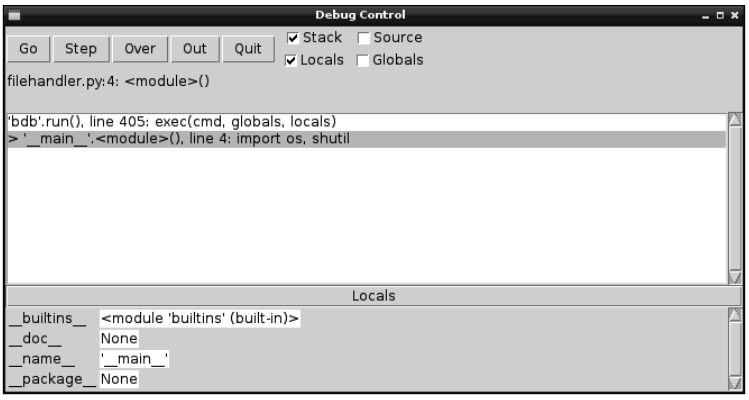
A key aspect of programming is being able to test and debug your code, and a useful tool to achieve this is a debugger. The IDLE editor (make sure you use IDLE3 to support the Python3 code we use in this book) includes a basic debugger. It allows you to step through your code, observe the values of local and global variables, and set breakpoints.

How to do it...

To enable the debugger, start IDLE3 and select **Debugger** from the **Debug** menu; it will open up the following window (if you are currently running some code, you will need to stop it first):

The IDLE3 debugger window

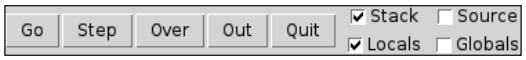
Open up the code you want to test (via **File** | **Open...**) and try running it (*F5*). You will find that the code will not start since the debugger has automatically stopped at the first line The following screenshot shows the debugger has stopped on the first line of code in `filehandler.py` which is line 4:`import os, shutil`.



The IDLE3 debugger at the start of the code

How it works...

The control buttons shown in the following screenshot allow you to run and/or jump through the code.

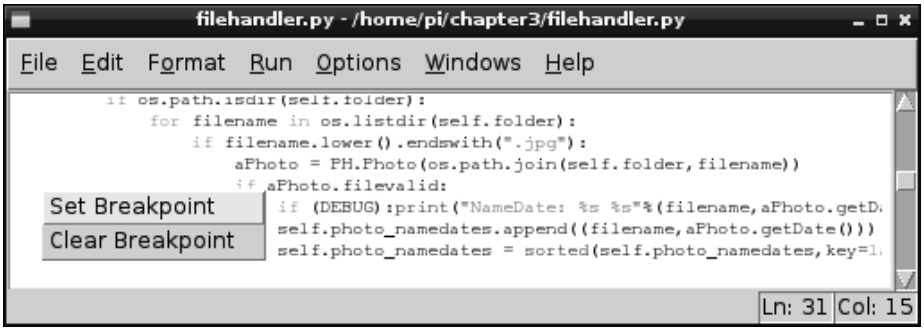


Debugger controls

The functions of the control buttons are as follows:

- **Go**: This button will execute the code as normal.
- **Step**: This button will execute the line of code one step at a time and then stop again. If a function is called, it will enter that function and allow you to step through that too.
- **Over**: This button is like the `Step` command, but if there is a function call, it will execute the whole function and stop at the following line.
- **Out**: This button will keep executing the code until it has completed the function it is currently in, allowing you to continue until you come out of the function.
- **Quit**: This button ends the program immediately.

In addition to the previously mentioned controls, you can **Set Breakpoint** and **Clear Breakpoint** directly within the code. A breakpoint is a marker that you can insert in the code (by right-clicking on the editor window), which the debugger will always break on (stop at) when it is reached.



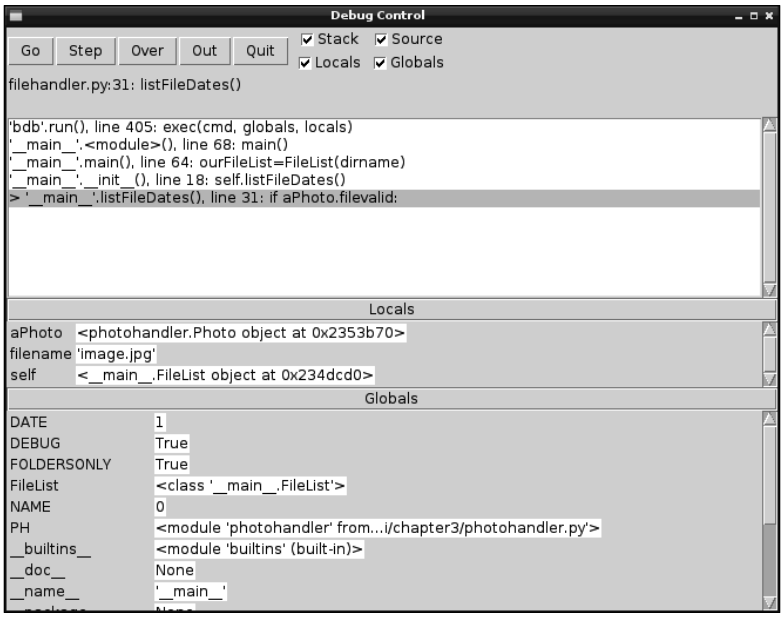
Set and clear breakpoints directly in your code

The checkboxes (on the right-hand side of the control buttons) allow you to choose what information to display when you step through the code or when the debugger stops somewhere due to a breakpoint. **Stack** is shown in the main window, which is similar to what you would see if the program hit an unhandled exception. The **Stack** option shows all the function calls made to get to the current position in the code, right up to the line it has stopped at. The **Source** option highlights the line of code currently being executed and, in some cases, the code inside the imported modules too (if they are noncompiled libraries).

You can also select whether to display **Locals** and/or **Globals**. By default, the **Source** and **Globals** options are usually disabled as they can make the process quite slow if there is a lot of data to display.

Python uses the concept of `Local` and `Global` variables to define the scope (where and when the variables are valid). `Global` variables are defined at the top level of the file and are visible from any point in the code after it has been defined. However, in order to alter its value from anywhere other than the top level, Python requires you to use the `global` keyword first. Without the `global` keyword, you will create a local copy with the same name (the value of which will be lost when you exit the function). `Local` variables are defined when you create a variable within a function; once outside of the function, the variable is destroyed and is not visible anymore.

Below **Stack** data are the **Locals**, in this case `aPhoto`, `filename`, and `self`. Then (if enabled), we have all the global values that are currently valid providing useful details about the status of the program (`DATE = 1, DEBUG = True, FOLDERSONLY = True`, and so on).



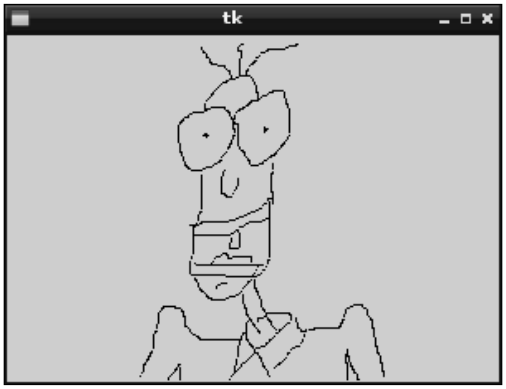
The Stack, Locals, and Globals options within the debugger

The debugger isn't particularly advanced as it does not allow you to expand complex objects such as the `photohandler.Photo` object to see what data it contains. However, if required, you can adjust your code and assign the data you want to observe to some temporary variables during testing.

It is worth learning how to use the debugger as it is a much easier way to track down particular problems and check whether or not things are functioning as you expect them to.

Drawing lines using a mouse on Tkinter Canvas

The Tkinter Canvas widget provides an area to create and draw objects on. The following script demonstrates how to use `mouse` events to interact with Tkinter. By detecting the mouse clicks, we can use Tkinter to draw a line that follows the movement of the mouse.



A simple drawing application using Tkinter

Getting ready

As before, we need to have Tkinter installed and either the Raspbian desktop running (`startx` from the command line) or an SSH session with X11 Forwarding and an X server running (see *Chapter 1*, *Getting Started with a Raspberry Pi Computer*). We will also need a mouse connected.

How to do it...

Create the following script, `painting.py`:

```
#!/usr/bin/python3
#painting.py
import tkinter as TK
```

```
#Set defaults
btn1pressed = False
newline = True

def main():
  root = TK.Tk()
  the_canvas = TK.Canvas(root)
  the_canvas.pack()
  the_canvas.bind("<Motion>", mousemove)
  the_canvas.bind("<ButtonPress-1>", mouse1press)
  the_canvas.bind("<ButtonRelease-1>", mouse1release)
  root.mainloop()

def mouse1press(event):
  global btn1pressed
  btn1pressed = True

def mouse1release(event):
  global btn1pressed, newline
  btn1pressed = False
  newline = True

def mousemove(event):
  if btn1pressed == True:
    global xorig, yorig, newline
    if newline == False:
      event.widget.create_line(xorig,yorig,event.x,event.y,
                                                  smooth=TK.TRUE)
    newline = False
    xorig = event.x
    yorig = event.y

if __name__ == "__main__":
  main()
#End
```

How it works...

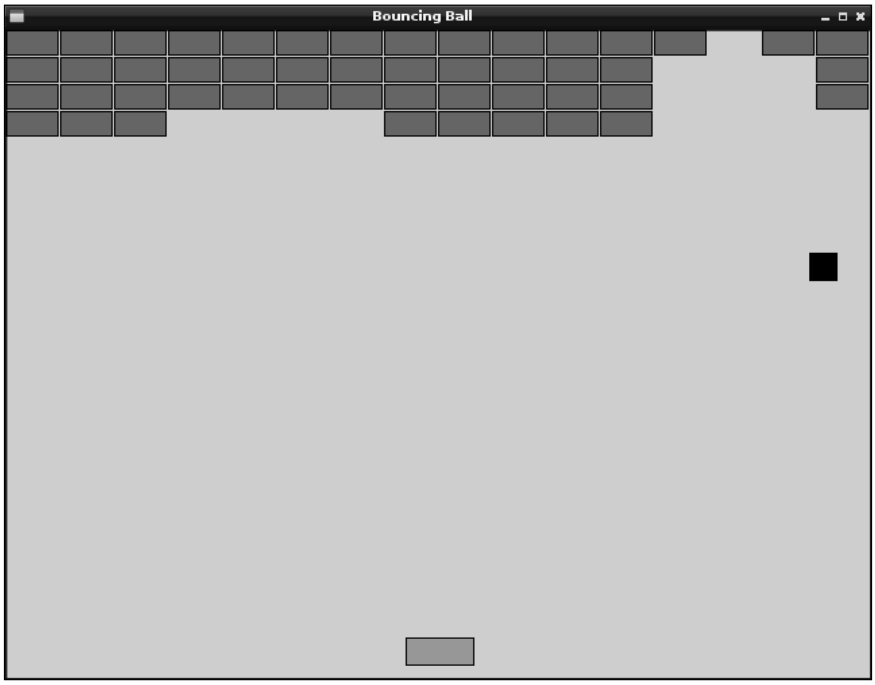
The Python code creates a Tkinter window that contains a `Canvas` object called `the_canvas`. We use the `bind` function here, which will bind a specific `event` that occurs on this widget (`the_canvas`) to a specific action or key press. In this case, we bind the `<Motion>` function of the mouse plus the click and release of the first mouse button (`<ButtonPress-1>` and `<ButtonRelease-1>`). Each of these events are then used to call the `mouse1press()`, `mouse1release()` and `mousemove()` functions.

The logic here is as follows. We track the status of the mouse button using the `mouse1press()` and `mouse1release()` functions and within the `mousemove()` function. We then check the status of the button.

If the mouse has been clicked, the function `mousemove()` will check to see whether we are drawing a new line (we set new coordinates for this) or continuing an old one (we draw a line from the previous coordinates to the coordinates of the current event that has triggered `mousemove()`). We just need to ensure that we reset to the `newline` command whenever the mouse button is released to reset the start position of the line.

Creating a bat and ball game

A classic bat and ball game can be created using the drawing tools of canvas and by detecting the collisions of the objects. The user will be able to control the green paddle using the left and right cursor keys to aim the ball at the bricks and hit them until they have all been destroyed.



Control the bat to aim the ball at the bricks

Getting ready

This example requires a screen and keyboard attached to the Raspberry Pi or X11 Forwarding and X server.

How to do it...

Create the following script, `bouncingball.py`:

```
#!/usr/bin/python3
# bouncingball.py
import tkinter as TK
import time

VERT,HOREZ=0,1
xTOP,yTOP = 0,1
xBTM,yBTM = 2,3
MAX_WIDTH,MAX_HEIGHT = 640,480
xSTART,ySTART = 100,200
BALL_SIZE=20
RUNNING=True

def close():
  global RUNNING
  RUNNING=False
  root.destroy()

def move_right(event):
  if canv.coords(paddle)[xBTM]<(MAX_WIDTH-7):
    canv.move(paddle, 7, 0)

def move_left(event):
  if canv.coords(paddle)[xTOP]>7:
    canv.move(paddle, -7, 0)

def determineDir(ball,obj):
  global delta_x,delta_y
  if (ball[xTOP] == obj[xBTM]) or (ball[xBTM] == obj[xTOP]):
    delta_x = -delta_x
  elif (ball[yTOP] == obj[yBTM]) or (ball[yBTM] == obj[yTOP]):
    delta_y = -delta_y

root = TK.Tk()
```

```
root.title("Bouncing Ball")
root.geometry('%sx%s+%s+%s' %(MAX_WIDTH, MAX_HEIGHT, 100, 100))
root.bind('<Right>', move_right)
root.bind('<Left>', move_left)
root.protocol('WM_DELETE_WINDOW', close)

canv = TK.Canvas(root, highlightthickness=0)
canv.pack(fill='both', expand=True)

top = canv.create_line(0, 0, MAX_WIDTH, 0, fill='blue',
                       tags=('top'))
left = canv.create_line(0, 0, 0, MAX_HEIGHT, fill='blue',
                        tags=('left'))
right = canv.create_line(MAX_WIDTH, 0, MAX_WIDTH, MAX_HEIGHT,
                    fill='blue', tags=('right'))
bottom = canv.create_line(0, MAX_HEIGHT, MAX_WIDTH, MAX_HEIGHT,
                    fill='blue', tags=('bottom'))

ball = canv.create_rectangle(0, 0, BALL_SIZE, BALL_SIZE,
                    outline='black', fill='black', tags=('ball'))
paddle = canv.create_rectangle(100, MAX_HEIGHT - 30, 150, 470,
                    outline='black', fill='green', tags=('rect'))

brick=list()
for i in range(0,16):
  for row in range(0,4):
    brick.append(canv.create_rectangle(i*40, row*20,
                       ((i+1)*40)-2, ((row+1)*20)-2,
                        outline='black', fill='red',
                        tags=('rect')))

delta_x = delta_y = 1
xold,yold = xSTART,ySTART
canv.move(ball, xold, yold)

while RUNNING:
  objects = canv.find_overlapping(canv.coords(ball)[0],
                                  canv.coords(ball)[1],
                                  canv.coords(ball)[2],
                                  canv.coords(ball)[3])

  #Only change the direction once (so will bounce off 1st
  # block even if 2 are hit)
  dir_changed=False
```

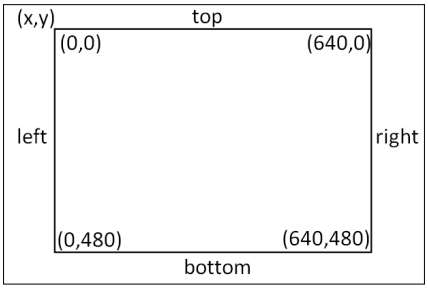
```
    for obj in objects:
      if (obj != ball):
        if dir_changed==False:
          determineDir(canv.coords(ball),canv.coords(obj))
          dir_changed=True
        if (obj >= brick[0]) and (obj <= brick[len(brick)-1]):
          canv.delete(obj)
        if (obj == bottom):
          text = canv.create_text(300,100,text="YOU HAVE MISSED!")
          canv.coords(ball, (xSTART,ySTART,
                             xSTART+BALL_SIZE,ySTART+BALL_SIZE))
          delta_x = delta_y = 1
          canv.update()
          time.sleep(3)
          canv.delete(text)
    new_x, new_y = delta_x, delta_y
    canv.move(ball, new_x, new_y)

    canv.update()
    time.sleep(0.005)
  #End
```

How it works...

We create a Tkinter application that is 640 x 480 pixels and bind the `<Right>` and `<Left>` cursor keys to the `move_right()` and `move_left()` functions. We use `root.protocol('WM_DELETE_WINDOW', close)` to detect when the window is closed so that we can cleanly exit the program (via `close()`, which sets `RUNNING` to `False`).

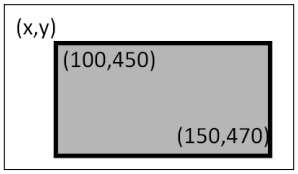
We then add a `Canvas` widget to the application that will hold all our objects. We create the following objects: `top`, `left`, `right`, and `bottom`. These make up our bounding sides for our game area. The canvas coordinates are `0,0` in the top-left corner and `640,480` in the bottom-right corner, so the start and end coordinates can be determined for each side (using `canv.create_line(xStart, yStart, xEnd, yEnd)`).



The coordinates of the Canvas widget

You can also add multiple `tags` to the objects; `tags` are often useful for defining specific actions or behaviors of objects. For instance, they allow for different types of events to occur when specific objects or bricks are hit. We see more uses of `tags` in the next example.

Next, we define the ball and paddle objects, which are added using `canv.create_rectangle()`. This requires two sets of coordinates that define the opposite corners of the objects (in this case, the top-left and bottom-right corners).

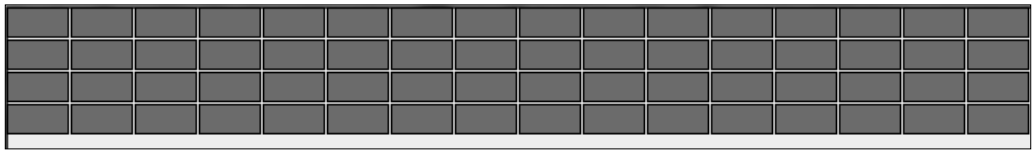


Finally, we can create the bricks!

We want our bricks to be 40 x 20 pixels wide so we can fit 16 bricks across our game area of 640 pixels (in four rows). We can create a list of brick objects with their positions defined automatically as shown in the following code:

```
brick=list()
for i in range(0,16):
    for row in range(0,4):
        brick.append(canv.create_rectangle(i*40, row*20,
          ((i+1)*40)-2, ((row+1)*20)-2, outline='black',
            fill='red', tags=('rect')))
```

A brick-like effect is provided by making the bricks slightly smaller (`-2`) to create a small gap.



Four rows of 16 bricks are generated at the top of Canvas

We now set the default settings before starting the main control loop. The movement of the ball will be governed by `delta_x` and `delta_y`, which are added or subtracted to the ball's previous position in each cycle.

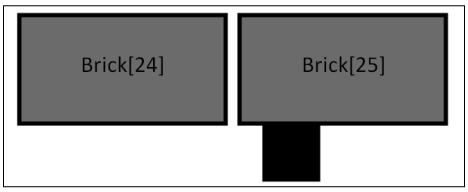
Next, we set the starting position of the ball and use the `canv.move()` function to move the ball by that amount. The `move()` function will add `100` to the `x` and `y` coordinates of the ball object, which was originally created at position `0,0`.

Now that everything is set up, the main loop can run; this will check that the ball has not hit anything (using the `canv.find_overlapping()` function), make any adjustments to the `delta_x` or `delta_y` values, and then apply them to move the ball to the next location.

The sign of `delta_x` and `delta_y` determines the direction of the ball. Positive values will make the ball travel diagonally downwards and towards the right. While `-delta_x` will make it travel towards the left, either downwards or upwards depending on whether `delta_y` is positive or negative.

After the ball has been moved, we use `canv.update()` to redraw any changes made to the display, and `time.sleep()` allows a small delay before checking and moving the ball again.

Object collisions are detected using the `canv.find_overlapping()` function. This returns a list of canvas objects that are found to be overlapping the bounds of a rectangle defined by the supplied coordinates. For example, in the case of the square ball, are any of the co-ordinates of the canvas objects within the rectangle?



If the ball is found to be overlapping another object, such as the walls, the paddle, or one or more of the bricks, we need to determine which direction the ball should next travel in. Since we are using the coordinates of the ball as the area within which to check, they will always be listed so that we ignore them when we check the list of objects.

We use the `dir_changed` flag to ensure that if we hit two bricks at the same time, we do not change direction twice before we move the ball. Otherwise, this would cause the ball to continue moving in the same direction even though it has collided with the bricks.

So if the ball is overlapping something else, we can call `determineDir()` with the coordinates of the ball and the object to work out what the new direction should be.

When the ball collides with something, we want the ball to bounce off it; fortunately, this is easy to simulate as we just need to change the sign of either `delta_x` or `delta_y` depending on whether we have hit something on the sides or the top/bottom. If the ball hits the bottom of another object, it means we were travelling upwards and should now travel downwards. However, we will continue to travel in the same direction on the x axis (be it left or right or just up) as it can be seen from the following code:

```
if (ball[xTOP] == obj[xBTM]) or (ball[xBTM] == obj[xTOP]):
        delta_x = -delta_x
```

The `determineDir()` function looks at the coordinates of the ball and the object and looks for a match between either the left and right x coordinates or the top and bottom y coordinates. This is enough to say whether the collision is on the sides or top/bottom, and we can set the `delta_x` or `delta_y` signs accordingly as it can be seen from the following code:

```
if (obj >= brick[0]) and (obj <= brick[-1]):
                canv.delete(obj)
```

Next, we can determine if we have hit a brick by checking whether the overlapping object ID is between the first and last ID bricks. If it was a brick, we can remove it using `canv.delete()`.

Python allows the index values to wrap around rather than access the invalid memory, so an index value of `-1` will provide us with the last item in the list.

We also check to see whether the object being overlapped is on the bottom line (in which case, the player has missed the ball with the paddle), so a short message is displayed briefly. We reset the position of the `ball` and `delta_x`/`delta_y` values. The `canv.update()` function ensures that the display is refreshed with the message before it is deleted (3 seconds later).

Finally, the ball is moved by the `delta_x`/`delta_y` distance and the display is updated. A small delay is added here to reduce the rate of updates and the CPU time used. Otherwise, you will find that your Raspberry Pi will become unresponsive if it is spending 100 percent of its effort in running the program.

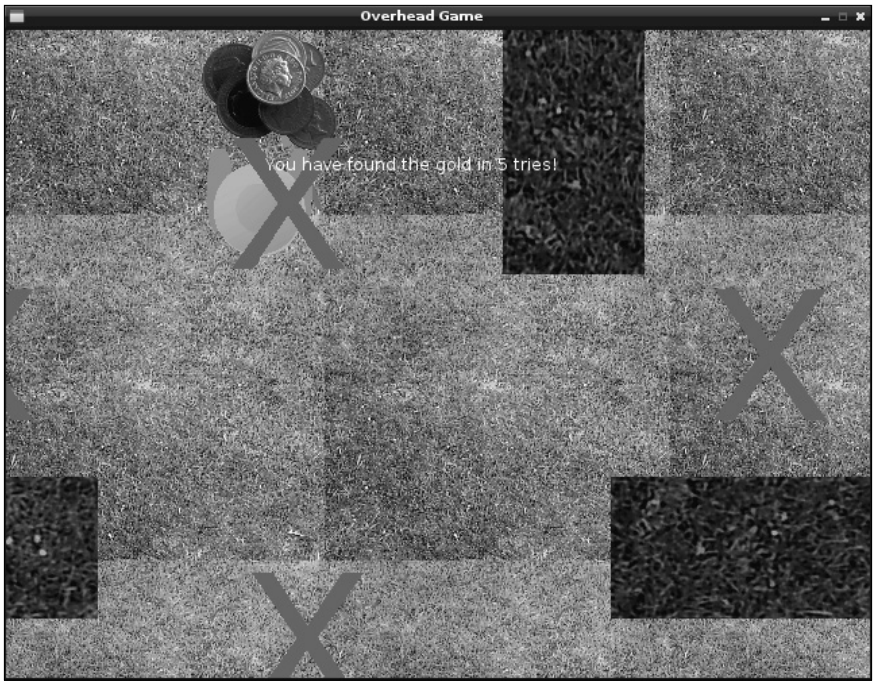
When the user presses the cursor keys, the `move_right()` and `move_left()` functions are called. They check the position of the paddle object, and if the objects are not at the edge, they will move the paddle accordingly. If the ball hits the paddle, the collision detection will ensure that the ball bounces off just like one of the bricks.

You can extend this game further by adding a score for each block destroyed, allow the player a finite number of lives which are lost when they miss the ball and even write some code to read in new brick layouts.

Creating an overhead scrolling game

By using objects and images in our programs, we can create many types of 2D graphical games.

In this recipe, we will create a treasure hunt game where the player is trying to find buried treasure (by pressing *Enter* to dig for it). Each time the treasure has not been found, the player is given a clue to how far away the treasure is; they can then use the cursor keys to move around and search until they find it.



Dig for treasure in your own overhead scrolling game!

Although this is a basic concept for a game, it could easily be extended to include multiple layouts, traps, and enemies to avoid, perhaps even additional tools or puzzles to solve. With a few adjustments to the graphics, the character could be exploring a dungeon or spaceship or hopping through the clouds collecting rainbows!

Getting ready

The following example uses a number of images; these are available as part of the book's resources. You will need to place the nine images in the same directory as the Python script.

The required image files can be seen in the code bundle of this chapter.

How to do it...

Create the following script, `scroller.py`:

1. Begin by importing the required libraries and parameters:

```
#!/usr/bin/python3
# scroller.py
import tkinter as TK
import time
import math
from random import randint

STEP=7
xVAL,yVAL=0,1
MAX_WIDTH,MAX_HEIGHT=640,480
SPACE_WIDTH=MAX_WIDTH*2
SPACE_HEIGHT=MAX_HEIGHT*2
LEFT,UP,RIGHT,DOWN=0,1,2,3
SPACE_LIMITS=[0,0,SPACE_WIDTH-MAX_WIDTH,
              SPACE_HEIGHT-MAX_HEIGHT]
DIS_LIMITS=[STEP,STEP,MAX_WIDTH-STEP,MAX_HEIGHT-STEP]
BGN_IMG="bg.gif"
PLAYER_IMG=["playerL.gif","playerU.gif",
            "playerR.gif","playerD.gif"]
WALL_IMG=["wallH.gif","wallV.gif"]
GOLD_IMG="gold.gif"
MARK_IMG="mark.gif"
newGame=False
checks=list()
```

2. Provide functions to handle the movement of the player:

```
def move_right(event):
  movePlayer(RIGHT,STEP)
def move_left(event):
  movePlayer(LEFT,-STEP)
def move_up(event):
  movePlayer(UP,-STEP)
def move_down(event):
  movePlayer(DOWN,STEP)

def foundWall(facing,move):
  hitWall=False
  olCoords=[canv.coords(player)[xVAL],
            canv.coords(player)[yVAL],
```

```
                canv.coords(player)[xVAL]+PLAYER_SIZE[xVAL],
                canv.coords(player)[yVAL]+PLAYER_SIZE[yVAL]]
    olCoords[facing]+=move
    objects = canv.find_overlapping(olCoords[0],olCoords[1],
                                    olCoords[2],olCoords[3])
    for obj in objects:
      objTags = canv.gettags(obj)
      for tag in objTags:
        if tag == "wall":
          hitWall=True
    return hitWall

  def moveBackgnd(movement):
    global bg_offset
    bg_offset[xVAL]+=movement[xVAL]
    bg_offset[yVAL]+=movement[yVAL]
    for obj in canv.find_withtag("bg"):
      canv.move(obj, -movement[xVAL], -movement[yVAL])

  def makeMove(facing,move):
    if facing == RIGHT or facing == LEFT:
      movement=[move,0] #RIGHT/LEFT
      bgOffset=bg_offset[xVAL]
      playerPos=canv.coords(player)[xVAL]
    else:
      movement=[0,move] #UP/DOWN
      bgOffset=bg_offset[yVAL]
      playerPos=canv.coords(player)[yVAL]
    #Check Bottom/Right Corner
    if facing == RIGHT or facing == DOWN:
      if (playerPos+PLAYER_SIZE[xVAL]) < DIS_LIMITS[facing]:
        canv.move(player, movement[xVAL], movement[yVAL])
      elif bgOffset < SPACE_LIMITS[facing]:
        moveBackgnd(movement)
    else:   #Check Top/Left Corner
      if (playerPos) > DIS_LIMITS[facing]:
        canv.move(player, movement[xVAL], movement[yVAL])
      elif bgOffset > SPACE_LIMITS[facing]:
        moveBackgnd(movement)

  def movePlayer(facing,move):
    hitWall=foundWall(facing,move)
    if hitWall==False:
      makeMove(facing,move)
    canv.itemconfig(player,image=playImg[facing])
```

3. Add functions to check how far the player is from the hidden gold:

```
def check(event):
  global checks,newGame,text
  if newGame:
    for chk in checks:
      canv.delete(chk)
    del checks[:]
    canv.delete(gold,text)
    newGame=False
    hideGold()
  else:
    checks.append(
            canv.create_image(canv.coords(player)[xVAL],
                              canv.coords(player)[yVAL],
                              anchor=TK.NW, image=checkImg,
                              tags=('check','bg')))
    distance=measureTo(checks[-1],gold)
    if(distance<=0):
      canv.itemconfig(gold,state='normal')
      canv.itemconfig(check,state='hidden')
      text = canv.create_text(300,100,fill="white",
                      text=("You have found the gold in"+
                            " %d tries!"%len(checks)))
      newGame=True
    else:
      text = canv.create_text(300,100,fill="white",
                 text=("You are %d steps away!"%distance))
      canv.update()
      time.sleep(1)
      canv.delete(text)

def measureTo(objectA,objectB):
  deltaX=canv.coords(objectA)[xVAL]-\
                       canv.coords(objectB)[xVAL]
  deltaY=canv.coords(objectA)[yVAL]-\
                       canv.coords(objectB)[yVAL]
  w_sq=abs(deltaX)**2
  h_sq=abs(deltaY)**2
  hypot=math.sqrt(w_sq+h_sq)
  return round((hypot/5)-20,-1)
```

4. Add functions to help find a location to hide the gold in:

```
def hideGold():
  global gold
  goldPos=findLocationForGold()
  gold=canv.create_image(goldPos[xVAL], goldPos[yVAL],
                         anchor=TK.NW, image=goldImg,
                         tags=('gold','bg'), state='hidden')

def findLocationForGold():
  placeGold=False
  while(placeGold==False):
    goldPos=[randint(0-bg_offset[xVAL],
           SPACE_WIDTH-GOLD_SIZE[xVAL]-bg_offset[xVAL]),
           randint(0-bg_offset[yVAL],
           SPACE_HEIGHT-GOLD_SIZE[yVAL]-bg_offset[yVAL])]
    objects = canv.find_overlapping(goldPos[xVAL],
                          goldPos[yVAL],
                          goldPos[xVAL]+GOLD_SIZE[xVAL],
                          goldPos[yVAL]+GOLD_SIZE[yVAL])
    findNewPlace=False
    for obj in objects:
      objTags = canv.gettags(obj)
      for tag in objTags:
        if (tag == "wall") or (tag == "player"):
          findNewPlace=True
    if findNewPlace == False:
      placeGold=True
  return goldPos
```

5. Create the Tkinter application window and bind the keyboard events:

```
root = TK.Tk()
root.title("Overhead Game")
root.geometry('%sx%s+%s+%s' %(MAX_WIDTH,
                              MAX_HEIGHT,
                              100, 100))
root.resizable(width=TK.FALSE, height=TK.FALSE)
root.bind('<Right>', move_right)
root.bind('<Left>', move_left)
root.bind('<Up>', move_up)
root.bind('<Down>', move_down)
root.bind('<Return>', check)

canv = TK.Canvas(root, highlightthickness=0)
canv.place(x=0,y=0,width=SPACE_WIDTH,height=SPACE_HEIGHT)
```

6. Initialize all the game objects (the background tiles, the player, the walls, and the gold):

```
#Create background tiles
bgnImg = TK.PhotoImage(file=BGN_IMG)
BGN_SIZE = bgnImg.width(),bgnImg.height()
background=list()
COLS=int(SPACE_WIDTH/BGN_SIZE[xVAL])+1
ROWS=int(SPACE_HEIGHT/BGN_SIZE[yVAL])+1
for col in range(0,COLS):
  for row in range(0,ROWS):
    background.append(canv.create_image(col*BGN_SIZE[xVAL],
                      row*BGN_SIZE[yVAL], anchor=TK.NW,
                      image=bgnImg, tags=('background','bg')))
bg_offset=[0,0]

#Create player
playImg=list()
for img in PLAYER_IMG:
  playImg.append(TK.PhotoImage(file=img))
#Assume images are all same size/shape
PLAYER_SIZE=playImg[RIGHT].width(),playImg[RIGHT].height()
player = canv.create_image(100,100, anchor=TK.NW,
                           image=playImg[RIGHT], tags=('player'))

#Create walls
wallImg=[TK.PhotoImage(file=WALL_IMG[0]),
         TK.PhotoImage(file=WALL_IMG[1])]
WALL_SIZE=[wallImg[0].width(),wallImg[0].height()]
wallPosH=[(0,WALL_SIZE[xVAL]*1.5),
          (WALL_SIZE[xVAL],WALL_SIZE[xVAL]*1.5),
          (SPACE_WIDTH-WALL_SIZE[xVAL],WALL_SIZE[xVAL]*1.5),
          (WALL_SIZE[xVAL],SPACE_HEIGHT-WALL_SIZE[yVAL])]
wallPosV=[(WALL_SIZE[xVAL],0),(WALL_SIZE[xVAL]*3,0)]
wallPos=[wallPosH,wallPosV]
wall=list()
for i,img in enumerate(WALL_IMG):
  for item in wallPos[i]:
    wall.append(canv.create_image(item[xVAL],item[yVAL],
              anchor=TK.NW, image=wallImg[i], tags=('wall','bg')))

#Place gold
goldImg = TK.PhotoImage(file=GOLD_IMG)
GOLD_SIZE=[goldImg.width(),goldImg.height()]
hideGold()
#Check mark
checkImg = TK.PhotoImage(file=MARK_IMG)
```

7. Finally, start the `mainloop()` command to allow Tkinter to monitor for events:

```
#Wait for actions from user
root.mainloop()
#End
```

How it works...

As before, we create a new Tkinter application that contains a `Canvas` widget that we can add all the objects of the game. We ensure that we bind the right, left, up, down and *Enter* keys, which will be our controls in the game.

First, we place our background image (`bg.gif`) onto the canvas widget. We calculate the number of images we can fit along the length and width to tile the whole canvas space and locate them using suitable coordinates.

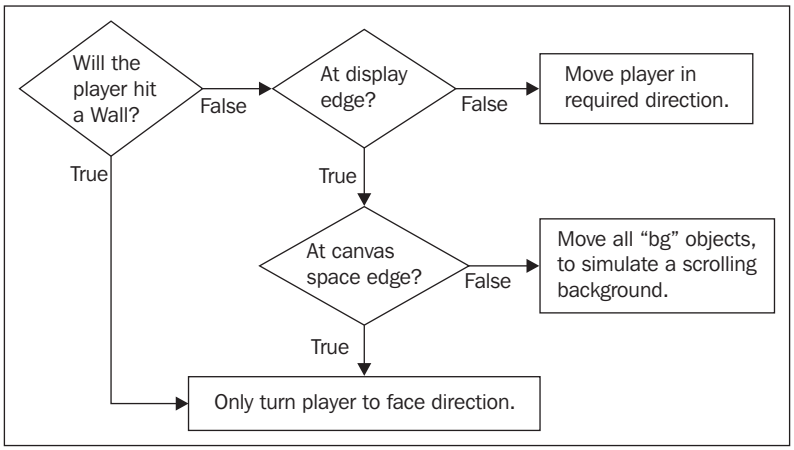
Next, we create the player image (by creating `playImg`, a list of Tkinter image objects for each direction the player can turn in) and place it on the canvas.

We now create the walls, the positions of which are defined by the `wallPosH` and `wallPosV` lists. These could be defined using the exact coordinates, perhaps even read from a file to provide an easy method to load different layouts for levels if required. By iterating through the lists, the horizontal and vertical wall images are put in position on the canvas.

To complete the layout, we just need to hide the gold somewhere. Using the `hideGold()` function, we randomly determine a suitable place to locate the gold. Within `findLocationForGold()`, we use `randint(0,value)` to create a pseudo-random number (it is not totally random but good enough for this use) between `0` and `value`. In our case, the value we want is between `0` and the edge of our canvas space minus the size of the gold image and any `bg_offset` that has been applied to the canvas. This ensures it is not beyond the edge of the screen. We then check the potential location using the `find_overlapping()` function to see whether any objects with `wall` or `player` tags are in the way. If so, we pick a new location. Otherwise, we place the gold on the canvas but with the `state="hidden"` value, which will hide it from view.

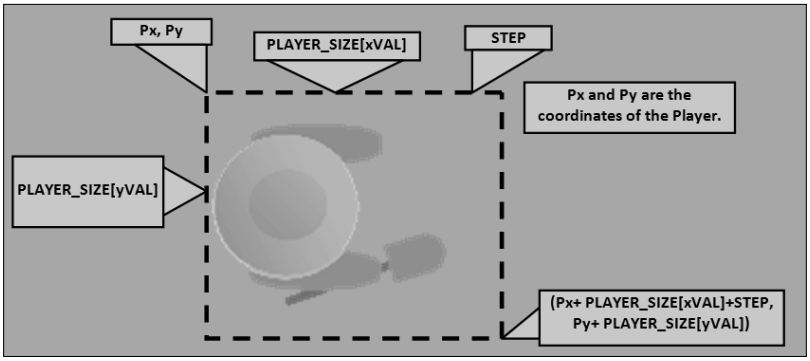
We then create `checkImg` (a Tkinter image) and use it while checking for gold to mark the area we have checked. Finally, we just wait for the user to press one of the keys.

The character will move around the screen whenever one of the cursor keys is pressed. The player's movement is determined by the `movePlayer()` function; it will first check whether the player is trying to move into a wall, then determine (within the `makeMove()` function) if the player is at the edge of the display or canvas space.



Every time a cursor key is pressed, we use the logic shown in the diagram to determine what to do

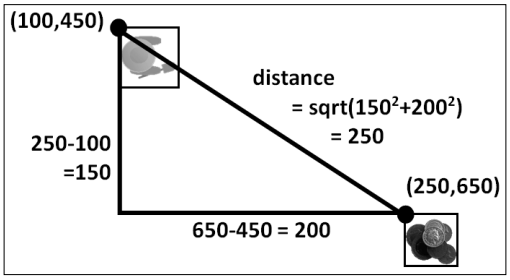
The `foundWall()` function works out whether the player will hit a wall by checking for any objects with `"wall"` tags within the area being covered by the player image, plus a little extra for the area that the player will be moving to next. The following diagram shows how the `olCoords` coordinates are determined:



The coordinates to check for objects that overlap (olCoords) are calculated

The `makeMove()` function checks if the player will be moving to the edge of the display (as defined by `DIS_LIMITS`) and whether they are at the edge of the canvas space (as defined by `SPACE_LIMITS`). Within the display, the player can be moved in the direction of the cursor, or all the objects tagged with `"bg"` within the canvas space are moved in the opposite direction, simulating scrolling behind the player. This is done by the `moveBackground()` function.

When the player presses *Enter*, we'll want to check for gold in the current location. Using the `measureTo()` function, the position of the player and the gold are compared (the distance between the x and y coordinates of each is calculated).



The result is scaled to provide a rough indication of how far away the player is from the gold. If the distance is greater than zero, we display how far away the player is from the gold and leave a cross to show where we have checked. If the player has found the gold, we display a message saying so and set `newGame` to `True`. The next time the player presses *Enter*, the places marked with a cross are removed and the gold is relocated to somewhere new.

With the gold hidden again, the player is ready to start again!

Where to buy this book

You can buy Raspberry Pi Cookbook for Python Programmers from the Packt Publishing website: `http://www.packtpub.com/raspberry-pi-cookbook-for-python-programmers/book`.

Free shipping to the US, UK, Europe and selected Asian countries. For more information, please read our shipping policy.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

www.PacktPub.com

For More Information:
www.packtpub.com/raspberry-pi-cookbook-for-python-programmers/book