Lab2 Report

Lab section 402

Xinyun Guo (xg284) Zhuoheng Li (zl764)

2019-10-09

Table of Content

Introduction

In lab 2, we added external buttons to the RPi setup to expand the control functions of video_control.py from Lab1. Also, we compared the performance of video_test.py using loop in lab 1 and modified versions of this program using callback. Furthermore, we developed test programs using the PyGame and expanded these programs into control panels for video_test.py

Design and Testing

1. Part 1

1.1 Add external buttons to the Rpi

In lab1, we created a video_control.py to control buttons on the piTFT. In this lab, we first add two external buttons to the video_control.py application from Lab1. These two additional buttons to the RPi setup are used for playing the video on the PiTFT.

We chose GPIO pin 26 and 19 for the pull-up and pull-down circuit respectively.

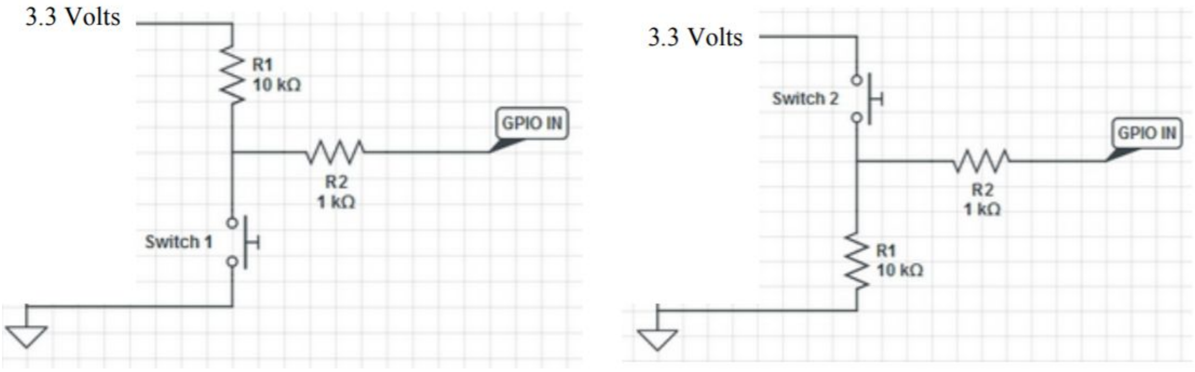


Figure 1 the pull-up and pull-down circuit for GPIO IN

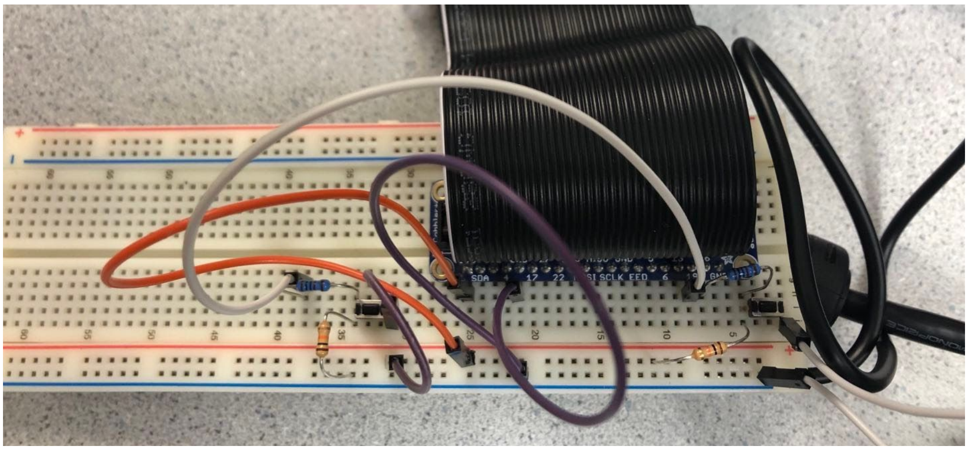
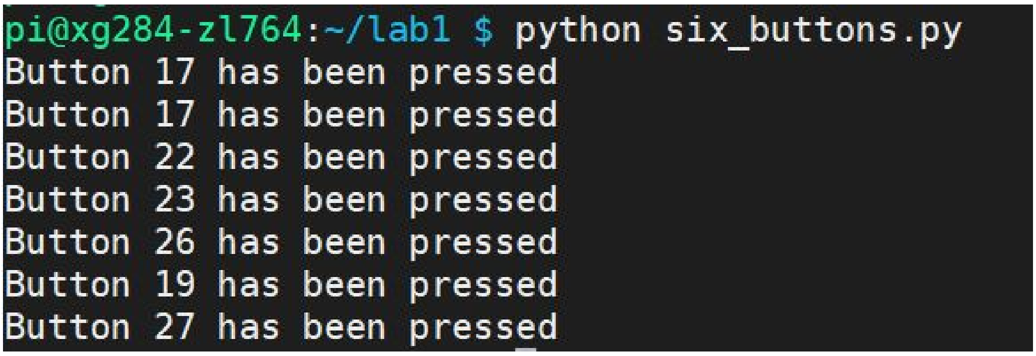


Figure 2 the circuit we connected

Once we completed the circuit, we extended four_buttons.py to create ‘six_buttons.py’ to make sure that two new buttons can be correctly read.



```
pi@xg284-zl764:~/lab1 $ python six_buttons.py
Button 17 has been pressed
Button 17 has been pressed
Button 22 has been pressed
Button 23 has been pressed
Button 26 has been pressed
Button 19 has been pressed
Button 27 has been pressed
```

Figure 3 test results for external buttons

Then, we extended ‘video_control.py’ to create ‘more_video_control.py’ with two new functions using external buttons to video control; fast forward 30 seconds and rewind 30 seconds. Finally, we modified the ‘start_video’ bash script to use the more_video_control.py.

1.2 Interrupt callbacks

In this step, we copied more_video_control.py into a new file, more_video_control_cb.py, in which the threaded callback interrupt routines are used for button presses. The code for more_video_control_cb.py can be found in the appendix.

After testing the functionality of the program, we further verified that it operates correctly in a bash script, similar to the operation of start_video from lab1. We named this new bash script Start_video_cb.

1.3 Performance measurement with ‘perf’ utilities

1.3.1 Install Perf

We first installed the “Perf” with the following commands:

sudo apt-get install linux-tools
sudo apt-get install linux-perf-4.18

1.3.2 Test Perf

After installation of the Perf, we test whether perf is operating as designed with the following commands:

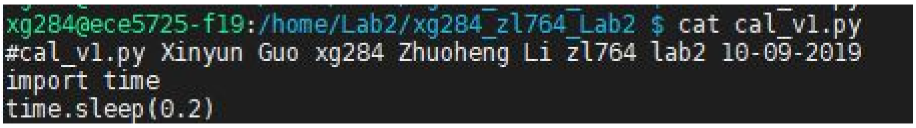
perf_4.18 --help
perf_4.18 list
perf_4.18 --version

1.3.3 Create cal_v1.py

We created a python file named cal_v1.py containing the following statements:

import time

time.sleep(0.2) # sleep



```
xg284@ece5725-f19:/home/Lab2/xg284_zl764_Lab2 $ cat cal_v1.py
#cal_v1.py Xinyun Guo xg284 Zhuoheng Li zl764 lab2 10-09-2019
import time
time.sleep(0.2)
```

Figure 4 code for cal_v1.py

1.3.4 Run cal_v1.py

sudo perf_4.18 stat -e

task-clock,context-switches,cpu-migrations,page-faults,cycles,instructions python cal_v1.py

This test gives a baseline set of statistics for simple python code.

1.3.5 Performance measurement of video_control.py

In this step, we implemented a series of tests to measure the performance of video_control.py and video_control_cb.py

1.3.5.1
We first modified more_video_control.py to introduce fixed timing prior to the perf run. Rename this function ‘more_video_control_perf.py’. Run the perf tools for more_video_control_perf.py, using a perf call:

sudo perf_4.18 stat -e
task-clock,context-switches,cpu-migrations,page-faults,cycles,instructions python
more_video_contol_perf.py

In the polling loop, we need to make sure to start with a ‘sleep’ value of 200 milliseconds (time.sleep(0.2)).

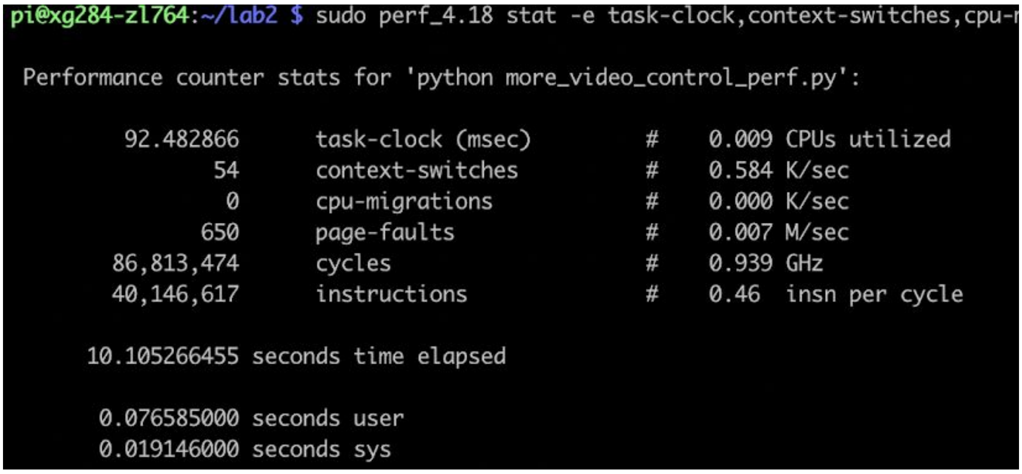
1.3.5.2
Then, we modified more_video_control_cb.py to introduce a fixed run-time prior to running the perf tool. Once modified, run the perf tools, for more_video_control_cb_perf.py.

1.3.5.3

We implemented successive runs of more_video_control_perf.py, changing polling loop times to 20 milliseconds, 2 milliseconds, 200 microseconds, 20 microseconds, and finally, with no sleep statement at all.

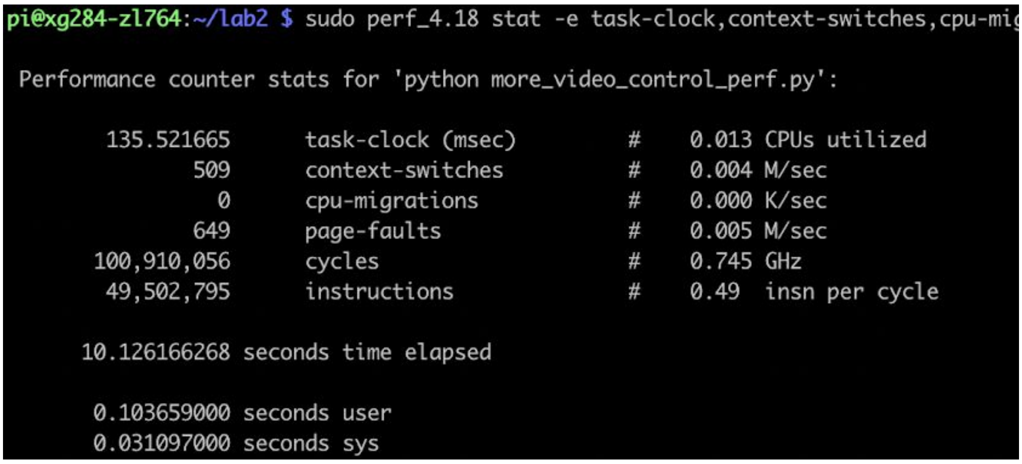
1.3.5.4

Note changes to the perf measurements and deduce impacts of changes and compare the results between successive runs of more_video_control_perf.py and with more_video_control_cb_perf.py.



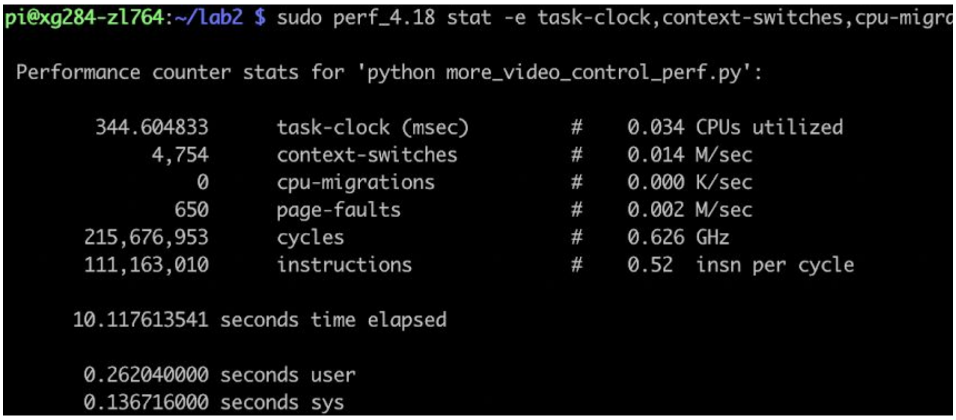
```
pi@xg284-zl764:~/lab2 $ sudo perf_4.18 stat -e task-clock,context-switches,cpu-r

 Performance counter stats for 'python more_video_control_perf.py':

         92.482866      task-clock (msec)         #    0.009 CPUs utilized
                54      context-switches          #    0.584 K/sec
                 0      cpu-migrations            #    0.000 K/sec
               650      page-faults               #    0.007 M/sec
        86,813,474      cycles                    #    0.939 GHz
        40,146,617      instructions              #    0.46  insn per cycle

      10.105266455 seconds time elapsed

       0.076585000 seconds user
       0.019146000 seconds sys
```

Figure 5.1 Result of running perf with 200ms sleep time



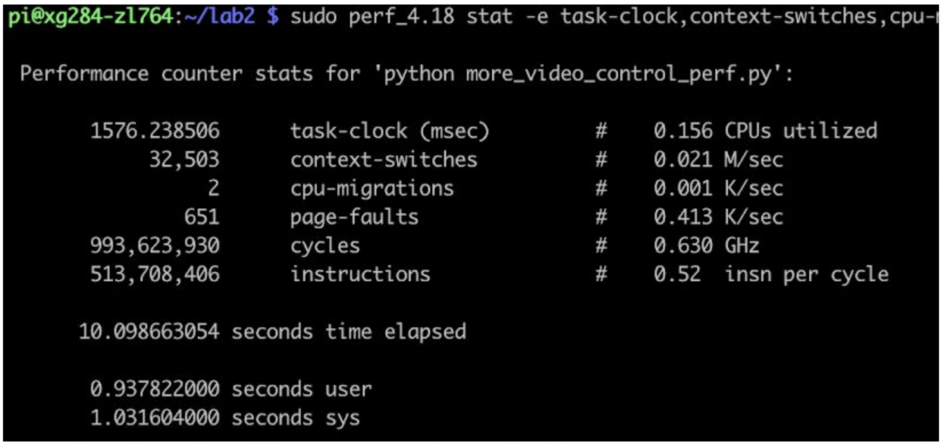
```
pi@xg284-zl764:~/lab2 $ sudo perf_4.18 stat -e task-clock,context-switches,cpu-mi

 Performance counter stats for 'python more_video_control_perf.py':

        135.521665      task-clock (msec)         #    0.013 CPUs utilized
               509      context-switches          #    0.004 M/sec
                 0      cpu-migrations            #    0.000 K/sec
               649      page-faults               #    0.005 M/sec
       100,910,056      cycles                    #    0.745 GHz
        49,502,795      instructions              #    0.49  insn per cycle

      10.126166268 seconds time elapsed

       0.103659000 seconds user
       0.031097000 seconds sys
```

Figure 5.2 Result of running perf with 20s sleep time



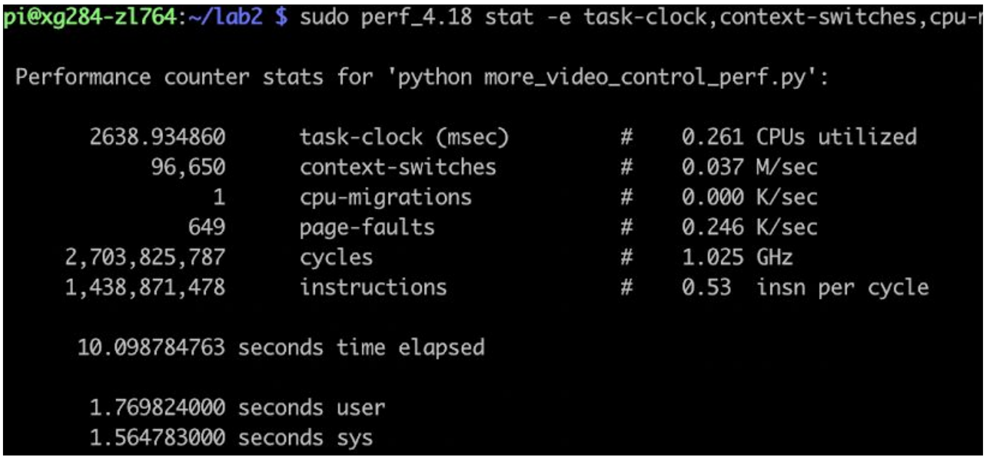
```
pi@xg284-zl764:~/lab2 $ sudo perf_4.18 stat -e task-clock,context-switches,cpu-migra

 Performance counter stats for 'python more_video_control_perf.py':

        344.604833      task-clock (msec)         #    0.034 CPUs utilized
             4,754      context-switches          #    0.014 M/sec
                 0      cpu-migrations            #    0.000 K/sec
               650      page-faults               #    0.002 M/sec
       215,676,953      cycles                    #    0.626 GHz
       111,163,010      instructions              #    0.52  insn per cycle

      10.117613541 seconds time elapsed

       0.262040000 seconds user
       0.136716000 seconds sys
```

Figure 5.3 Result of running perf with 2ms sleep time



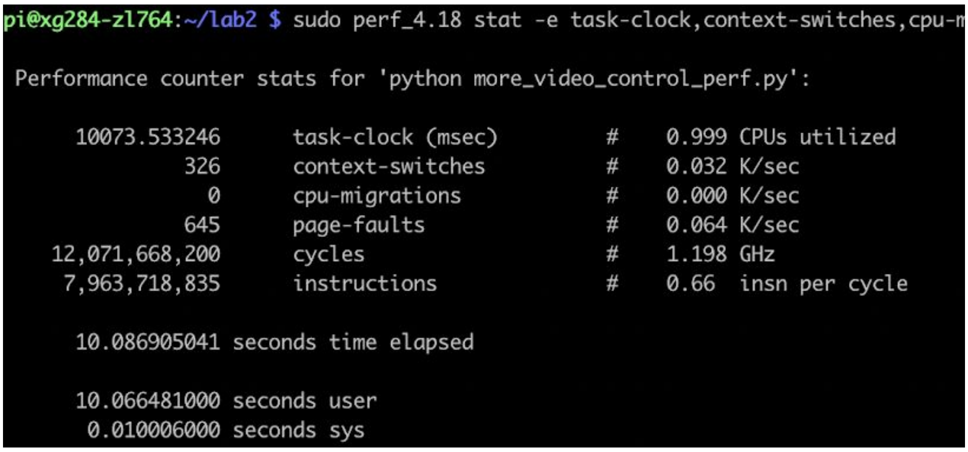
```
pi@xg284-zl764:~/lab2 $ sudo perf_4.18 stat -e task-clock,context-switches,cpu-

 Performance counter stats for 'python more_video_control_perf.py':

       1576.238506      task-clock (msec)         #    0.156 CPUs utilized
            32,503      context-switches          #    0.021 M/sec
                 2      cpu-migrations            #    0.001 K/sec
               651      page-faults               #    0.413 K/sec
       993,623,930      cycles                    #    0.630 GHz
       513,708,406      instructions              #    0.52  insn per cycle

      10.098663054 seconds time elapsed

       0.937822000 seconds user
       1.031604000 seconds sys
```

Figure 5.4 Result of running perf with 0.2ms sleep time



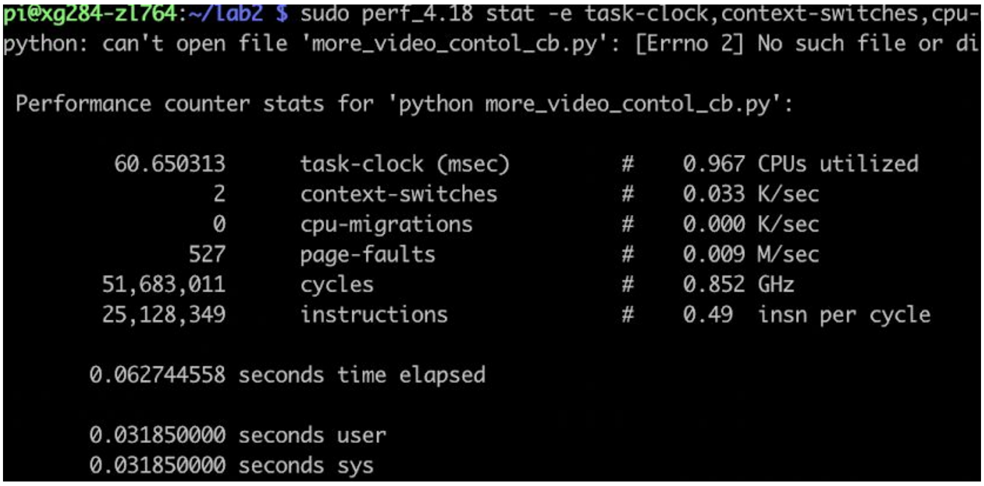
```
pi@xg284-zl764:~/lab2 $ sudo perf_4.18 stat -e task-clock,context-switches,cpu-

 Performance counter stats for 'python more_video_control_perf.py':

       2638.934860      task-clock (msec)         #    0.261 CPUs utilized
            96,650      context-switches          #    0.037 M/sec
                 1      cpu-migrations            #    0.000 K/sec
               649      page-faults               #    0.246 K/sec
     2,703,825,787      cycles                    #    1.025 GHz
     1,438,871,478      instructions              #    0.53  insn per cycle

      10.098784763 seconds time elapsed

       1.769824000 seconds user
       1.564783000 seconds sys
```

Figure 5.5 Result of running perf with 0.02ms sleep time



```
pi@xg284-zl764:~/lab2 $ sudo perf_4.18 stat -e task-clock,context-switches,cpu-m

 Performance counter stats for 'python more_video_control_perf.py':

      10073.533246      task-clock (msec)         #    0.999 CPUs utilized
               326      context-switches          #    0.032 K/sec
                 0      cpu-migrations            #    0.000 K/sec
               645      page-faults               #    0.064 K/sec
    12,071,668,200      cycles                    #    1.198 GHz
     7,963,718,835      instructions              #    0.66  insn per cycle

      10.086905041 seconds time elapsed

      10.066481000 seconds user
       0.010006000 seconds sys
```

Result 5.6 Result of running perf with no sleep time



```
pi@xg284-zl764:~/lab2 $ sudo perf_4.18 stat -e task-clock,context-switches,cpu-
python: can't open file 'more_video_contol_cb.py': [Errno 2] No such file or di

 Performance counter stats for 'python more_video_contol_cb.py':

         60.650313      task-clock (msec)         #    0.967 CPUs utilized
                 2      context-switches          #    0.033 K/sec
                 0      cpu-migrations            #    0.000 K/sec
               527      page-faults               #    0.009 M/sec
        51,683,011      cycles                    #    0.852 GHz
        25,128,349      instructions              #    0.49  insn per cycle

       0.062744558 seconds time elapsed

       0.031850000 seconds user
       0.031850000 seconds sys
```

Figure 6.1 Result of running cb_perf

Results:

From the test results, we can see that for more_video_control_perf, more polling loop time will take more CPU resources and more time which results in the CPUs fully utilized. In addition, the number of cycles and context-switches will also increase with the decrease of sleep time. While in more_video_control_perf, more polling loop time does not affect the number of page-fault and cpu-migration . Also, from the performance of measurement of more_video_control_perf.py and more_video_control_perf_cb.py, we can find that even though the CPU utilization rate are almost the same, the more_video_control_perf_cb.py run much less cycles and instructions. This means that the callback function is more effective.

2. Part 2

In this part, we implemented a PyGame: Bounce Program. In this section, we also implemented a physical quit button and a timeout for the following codes. Also, in order to

enable the functionality of PiTFT, we set the following environment variables when running on the piTFT:
import os os.putenv('SDL_VIDEODRIVER', 'fbcon')
os.putenv('SDL_FBDEV', '/dev/fb1')

2.1 Bounce program for one ball

We first created surface in Pygame and a rect for the object ball. By using the flip() function, the object ball can be displayed on the screen. Further, we set the starting coordinates and speed of the ball to make it moving. Then, we implemented a while loop to detect whether the ball hits the edges of the screen and change it speed in that case. Then, the python code bounce.py (see appendix) can achieve the movement of the ball and the bounce of the ball when it hits the edges.

2.2 Bounce program for two balls

We further extended the code to include 2 balls, two_bounce.py (see appendix), where each ball moves on the screen at a different speed. Meanwhile, two balls will bounce back if they reach the edges of the screen.

2.3 Bounce and collide program for two balls

We expanded two_bounce.py to create two_collide.py (see appendix), so that the balls alter their trajectories as they collide with one another. To achieve this, we use the colliderect() to detect whether two balls have collided or not. When a collision has been detected, two balls will bounce back to oppose direction which is achieved by setting the original speed to the opposite direction.

3. Part 3

3.1 Enable piTFT touch screen controls

We first need to enable the functionality of piTFT touch screen controls

3.1.1 Enable wheezy package sources by editing the new file:

sudo vim /etc/apt/sources.list.d/wheezy.list

and adding the line:

deb http://legacy.raspbian.org/raspbian wheezy main Save and close the file

3.1.2 Set stable as default package source (for the wheezy changes) by editing the new file:

sudo vim /etc/apt/apt.conf.d/10defaultRelease and adding the line:

APT::Default-release “stable”; Save and close the file

3.1.3 Set the priority for libsdl from wheezy higher than the Buster package by editing the

new file: *sudo vim /etc/apt/preferences.d/libsdl*

and adding the lines:

```
Package: libsdl1.2debian
Pin: release n=buster
Pin-Priority: -10
Package:libsdl1.2debian
Pin: release n=wheezy
Pin-Priority: 900
```

3.1.4 Install the changes by running the commands:

```
sudo apt-get update
sudo apt-get –y —allow-downgrades install libsdl1.2debian/wheezy
```

3.2 Set piTFT environment

Control for monitor use is determined by the environment variables:

```
os.putenv('SDL_VIDEODRIVER', 'fbcon') # Display on piTFT
os.putenv('SDL_FBDEV', '/dev/fb1') #
os.putenv('SDL_MOUSEDRV', 'TSLIB') # Track mouse clicks on piTFT
os.putenv('SDL_MOUSEDEV', '/dev/input/touchscreen')
```

Also we need to disable the mouse when running the program on piTFT, this can be achieved by the command: *pygame.mouse.set_visible(False)*

3.3 Design a quit button

In this step, we designed a python application, quit_button.py to display a single ‘quit’ button on the lower edge of the screen. The program is designed so that touching the ‘quit’ button ends the program and returns to the Linux console screen. To achieve this, we use the event.get() to detect hits and pygame.mouse.get_pos() to detect the coordinates of the hits. If the hits has been detected within the quit button position, the system will quit.

3.4 Display the screen coordinates of a hit

We expanded quit_button.py into a second python application, screen_coordinates.py. The program will display a single quit button at the bottom of the screen, and tapping the ‘quit’ button will exit the program. Similarly, we detect the event of hits and display the coordinates after the hit (mouse up state in the code in appenx). Therefore, when user taps any location, the screen will display ‘Hit at x, y’ where x, y show the screen coordinates of the hit, and the coordinates of the hit will keep displaying on the screen until the next hit detected.

3.5 Design control panel to run two_collide

We designed a python program 'two_button.py' with the following functions:
- Two, on-screen buttons are displayed 'start' and 'quit'
- Hitting 'start' begins playback of two_collide.py
- Hitting 'quit' ends the program and returns to the Linux console screen.
- Hitting any other location on the screen displays screen coordinates.
- The start and quit buttons should be displayed on the screen, and operate whenever they are displayed, during the entire time the program is running (including while the animation is playing)

To implement this, we need to create more button on the screen as we did in the previous steps. Only difference is when the hit on 'start' button has been detected, the two_collide.py will be called.

3.6 Design two-level control panel to run two_collide

We further designed a python program control_two_collide.py with the following functions:
- Start and quit are implemented with identical functions. Screen coordinates should be displayed if hits occur outside of start and quit buttons. This is the 'level 1' menu.
- Once the animation begins to play, a second level of button controls , shown above on the 'level 2' menu should be displayed including the following buttons and associated functions:
- Pause/restart: pause a running animation. Restart a paused animation
- Faster: speed up the animation by a fixed amount
- Slower: slow the animation by a fixed amount
- Back: stop the animation and return to the 'top' menu screen which implements the start and quit buttons.

Similarly, we created new touch buttons for the 'level 2' panel. We use a flag (aniRunning in the code in appendix) to help us to record whether the 'start ' button has been pressed. When the flag is true, which indicates, 'start button has been pressed and we need to display the 'level 2' panel, versa vise.

Result：

Following the lab procedure, we implemented and tested callback functions on Raspberry Pi. Comparing the perf measurements results of using callback function will reduce the cycles during the running time. The number of context-switches is lower as well. We learnt that using callback function is a better choice when listening to some events from the GPIO.

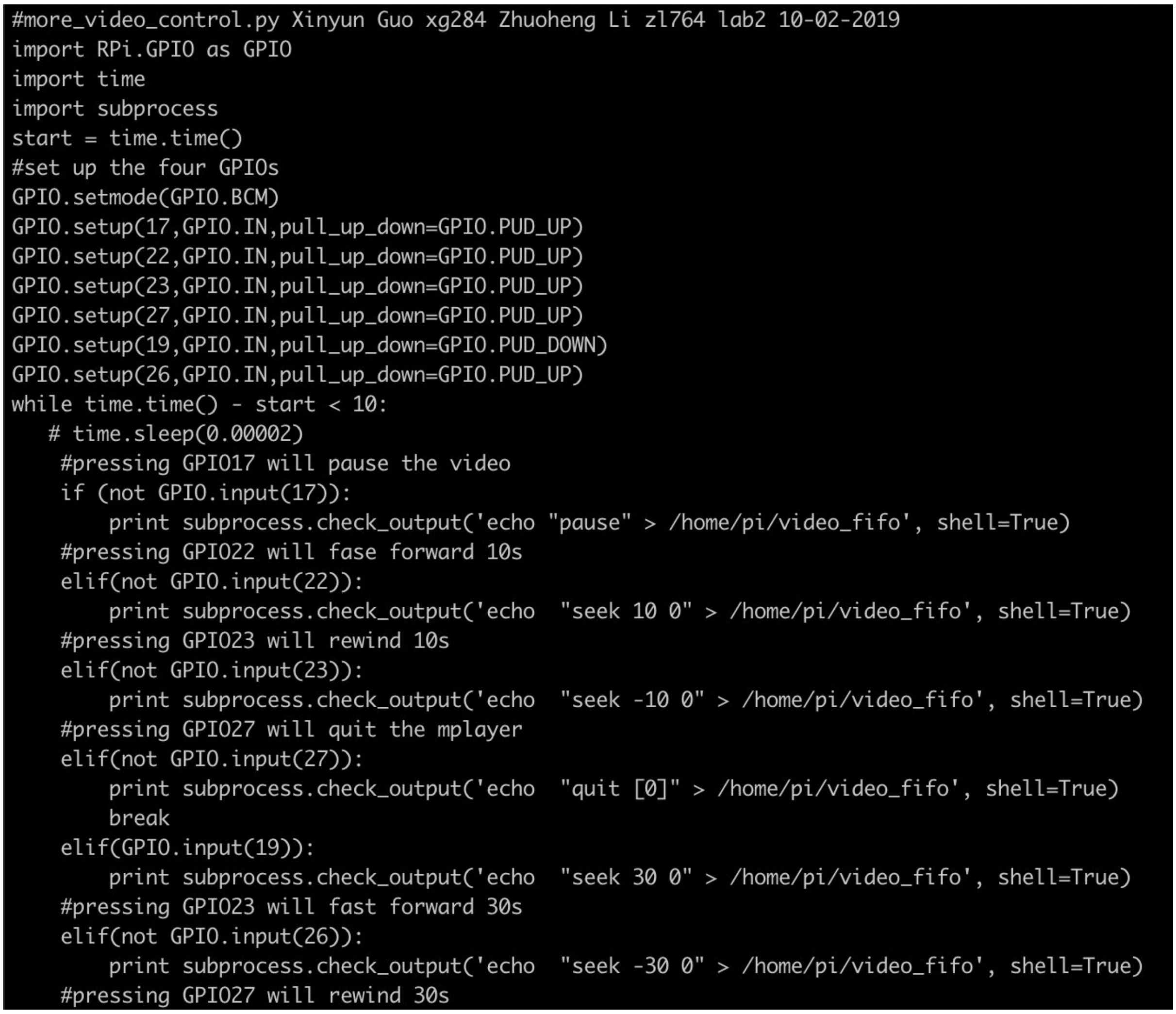
Furthermore, through the implementation of the Pygame, we learned more about how the piTFT touch screen control works and how to use the libraries of the Pygame to achieve performance we want, such as displaying objects, detecting hits and displaying the corresponding coordinates, illustrating ball bounce, and creating control panel. By using the correct library functions, the piTFT touch screen perform as we expected for those tasks.

Conclusion

In lab 2, we completed most steps smoothly, and most tasks performed well as we expected. During the lab, we encountered a problem that the two balls in the two_collide.py will "collide" when there are still a distance between them. At first, we think that there were some problems with our code. However, after checking the code carefully, we didn't find any problems with that. Then, we thought about the real shape of the rect objects of the two balls. Actually, the two objects are rectangles so that there are margins around the balls are not covered by the ball picture. Since we can't change the shape of rect, we are going to resize the rect so that the empty margin area can be smaller. By doing that, there is a still a short distance between the two balls when they collide, but the short distance is hard to notice by naked eye.

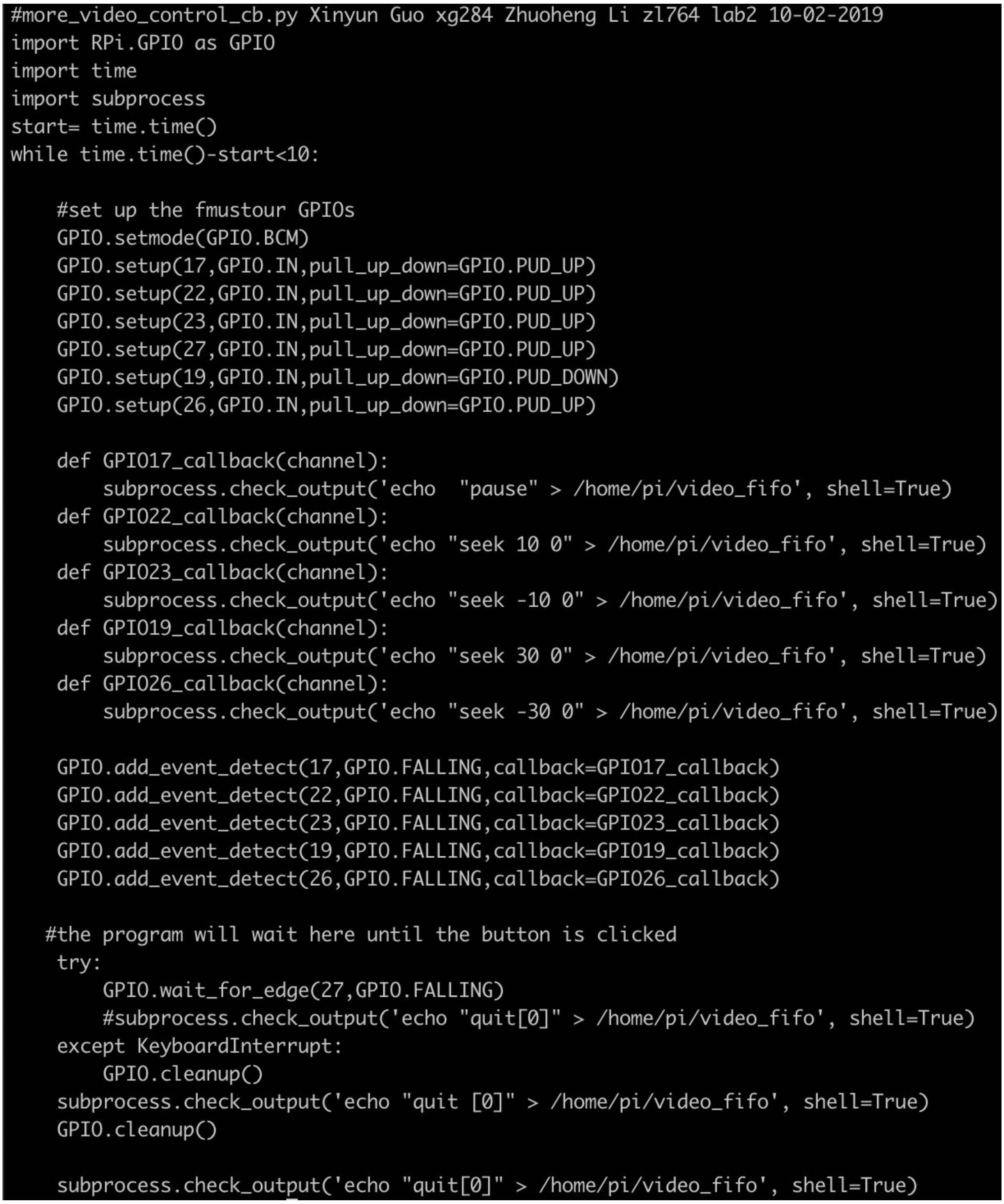
We think the lab description is clear and organized, and we are able to follow each step during the lab. However, we did not check out the lab within the lab section. We think it would be nice to have more time to work through all the tasks.

Appendix



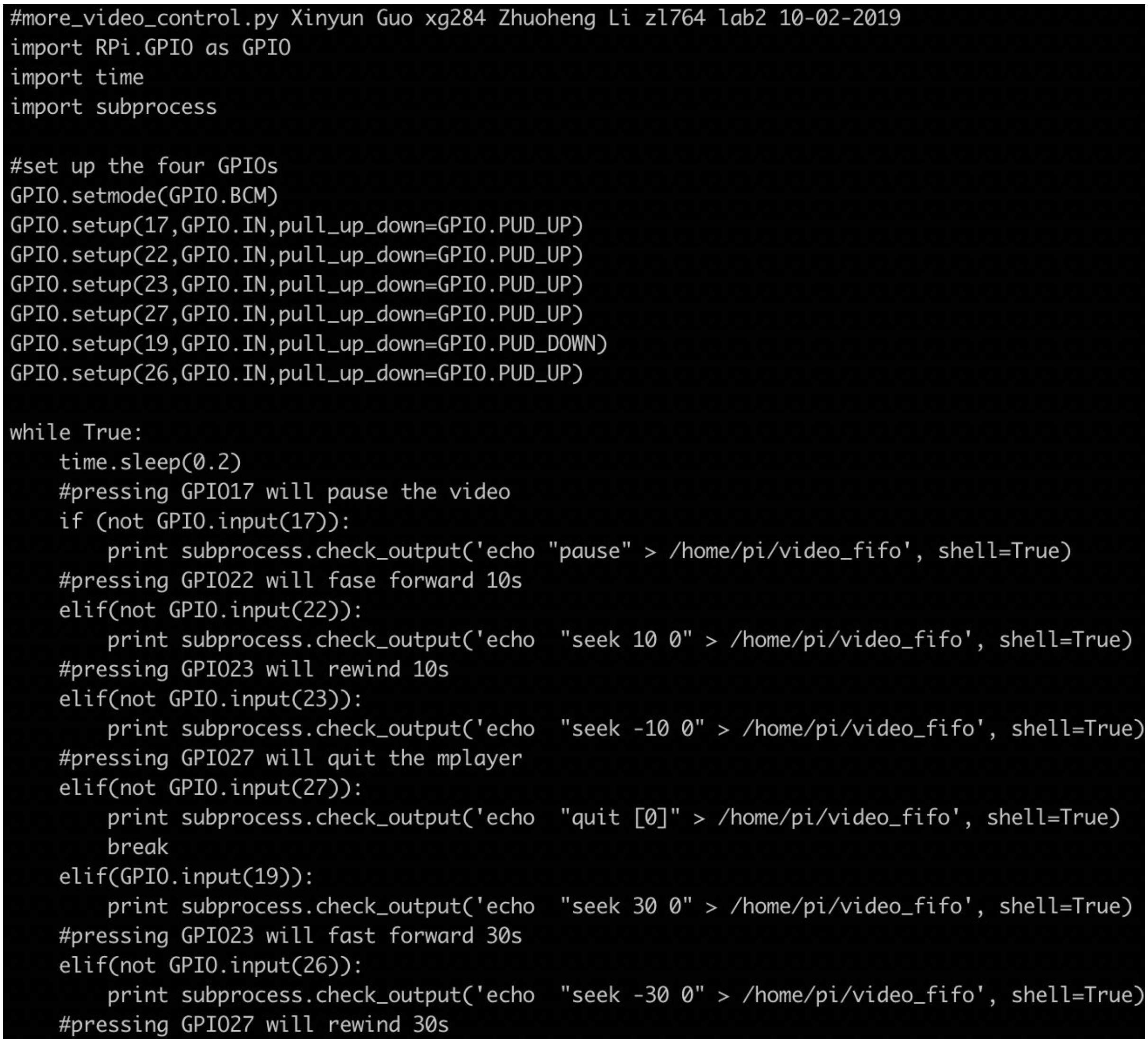
```
#more_video_control.py Xinyun Guo xg284 Zhuoheng Li zl764 lab2 10-02-2019
import RPi.GPIO as GPIO
import time
import subprocess
start = time.time()
#set up the four GPIOs
GPIO.setmode(GPIO.BCM)
GPIO.setup(17,GPIO.IN,pull_up_down=GPIO.PUD_UP)
GPIO.setup(22,GPIO.IN,pull_up_down=GPIO.PUD_UP)
GPIO.setup(23,GPIO.IN,pull_up_down=GPIO.PUD_UP)
GPIO.setup(27,GPIO.IN,pull_up_down=GPIO.PUD_UP)
GPIO.setup(19,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
GPIO.setup(26,GPIO.IN,pull_up_down=GPIO.PUD_UP)
while time.time() - start < 10:
   # time.sleep(0.00002)
    #pressing GPIO17 will pause the video
    if (not GPIO.input(17)):
        print subprocess.check_output('echo "pause" > /home/pi/video_fifo', shell=True)
    #pressing GPIO22 will fase forward 10s
    elif(not GPIO.input(22)):
        print subprocess.check_output('echo  "seek 10 0" > /home/pi/video_fifo', shell=True)
    #pressing GPIO23 will rewind 10s
    elif(not GPIO.input(23)):
        print subprocess.check_output('echo  "seek -10 0" > /home/pi/video_fifo', shell=True)
    #pressing GPIO27 will quit the mplayer
    elif(not GPIO.input(27)):
        print subprocess.check_output('echo  "quit [0]" > /home/pi/video_fifo', shell=True)
        break
    elif(GPIO.input(19)):
        print subprocess.check_output('echo  "seek 30 0" > /home/pi/video_fifo', shell=True)
    #pressing GPIO23 will fast forward 30s
    elif(not GPIO.input(26)):
        print subprocess.check_output('echo  "seek -30 0" > /home/pi/video_fifo', shell=True)
    #pressing GPIO27 will rewind 30s
```

Figure 1Python code for more_video_control.py



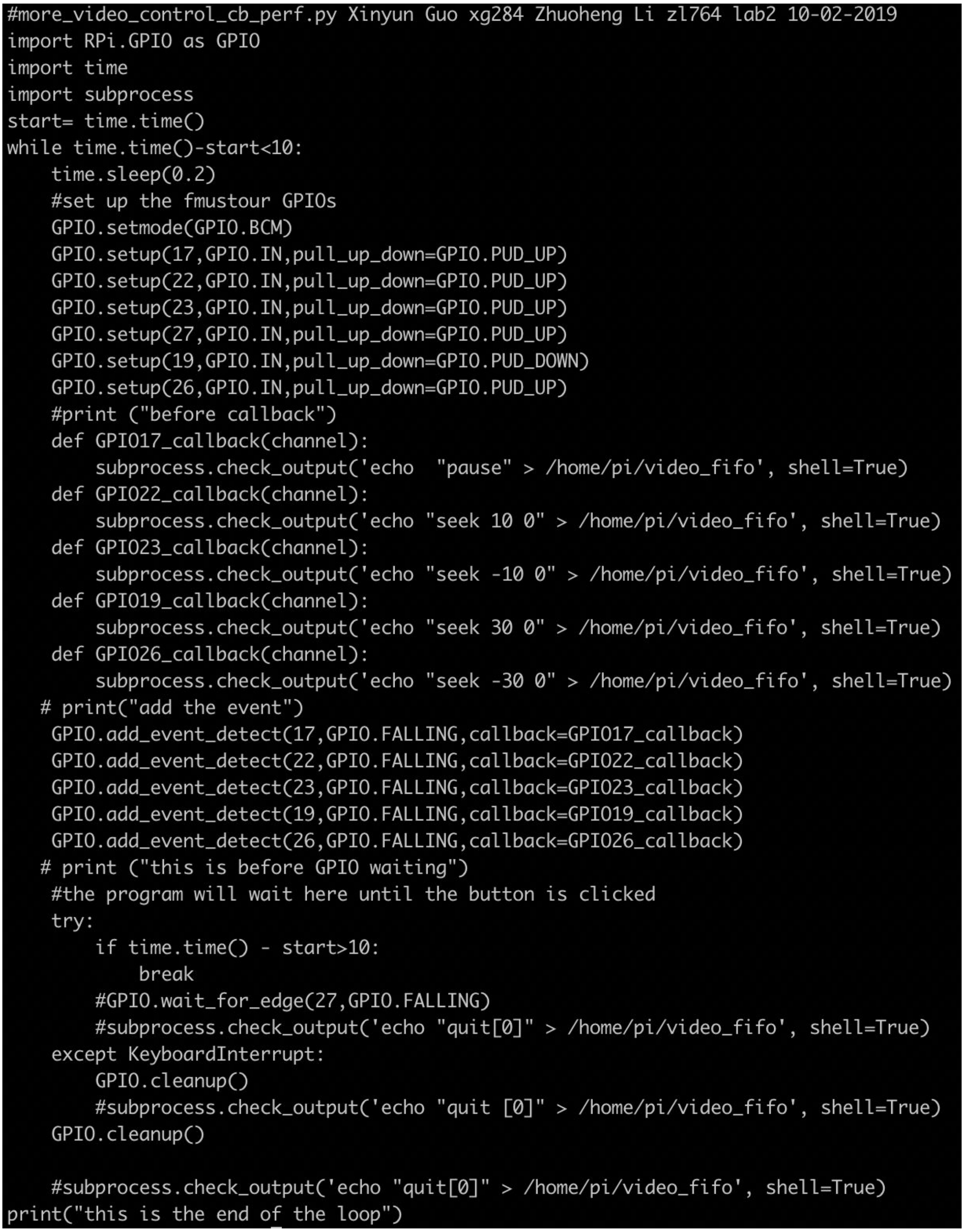
```
#more_video_control_cb.py Xinyun Guo xg284 Zhuoheng Li zl764 lab2 10-02-2019
import RPi.GPIO as GPIO
import time
import subprocess
start= time.time()
while time.time()-start<10:

    #set up the fmustour GPIOs
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(17,GPIO.IN,pull_up_down=GPIO.PUD_UP)
    GPIO.setup(22,GPIO.IN,pull_up_down=GPIO.PUD_UP)
    GPIO.setup(23,GPIO.IN,pull_up_down=GPIO.PUD_UP)
    GPIO.setup(27,GPIO.IN,pull_up_down=GPIO.PUD_UP)
    GPIO.setup(19,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
    GPIO.setup(26,GPIO.IN,pull_up_down=GPIO.PUD_UP)

    def GPIO17_callback(channel):
        subprocess.check_output('echo  "pause" > /home/pi/video_fifo', shell=True)
    def GPIO22_callback(channel):
        subprocess.check_output('echo "seek 10 0" > /home/pi/video_fifo', shell=True)
    def GPIO23_callback(channel):
        subprocess.check_output('echo "seek -10 0" > /home/pi/video_fifo', shell=True)
    def GPIO19_callback(channel):
        subprocess.check_output('echo "seek 30 0" > /home/pi/video_fifo', shell=True)
    def GPIO26_callback(channel):
        subprocess.check_output('echo "seek -30 0" > /home/pi/video_fifo', shell=True)

    GPIO.add_event_detect(17,GPIO.FALLING,callback=GPIO17_callback)
    GPIO.add_event_detect(22,GPIO.FALLING,callback=GPIO22_callback)
    GPIO.add_event_detect(23,GPIO.FALLING,callback=GPIO23_callback)
    GPIO.add_event_detect(19,GPIO.FALLING,callback=GPIO19_callback)
    GPIO.add_event_detect(26,GPIO.FALLING,callback=GPIO26_callback)

   #the program will wait here until the button is clicked
    try:
        GPIO.wait_for_edge(27,GPIO.FALLING)
        #subprocess.check_output('echo "quit[0]" > /home/pi/video_fifo', shell=True)
    except KeyboardInterrupt:
        GPIO.cleanup()
    subprocess.check_output('echo "quit [0]" > /home/pi/video_fifo', shell=True)
    GPIO.cleanup()

    subprocess.check_output('echo "quit[0]" > /home/pi/video_fifo', shell=True)
```

Figure 2 Python code for more_video_control_cb.py



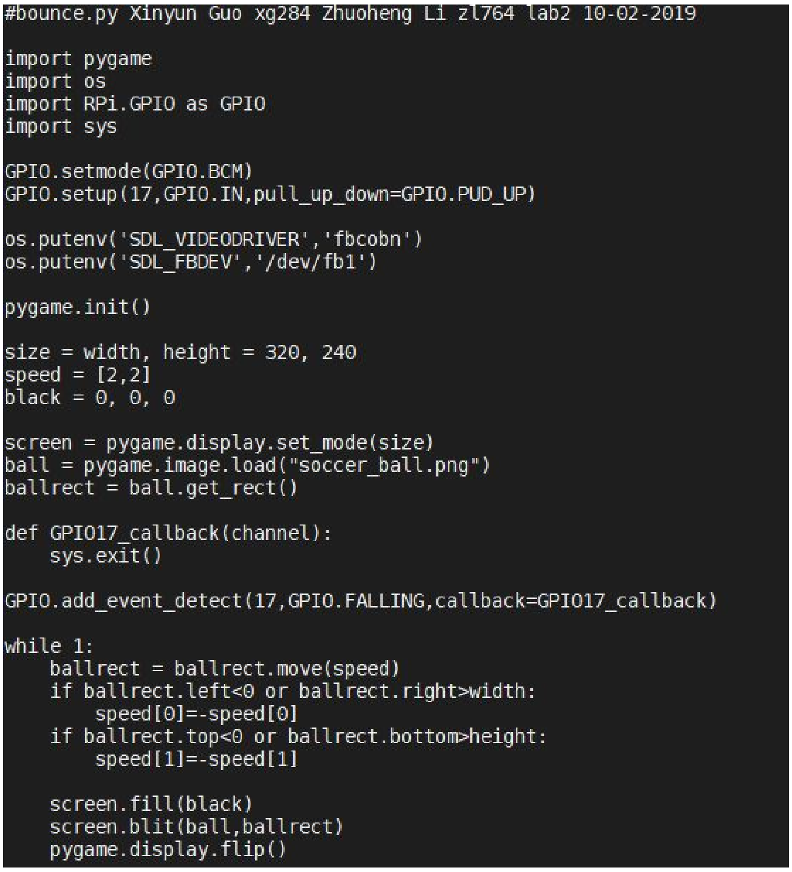
```
#more_video_control.py Xinyun Guo xg284 Zhuoheng Li zl764 lab2 10-02-2019
import RPi.GPIO as GPIO
import time
import subprocess

#set up the four GPIOs
GPIO.setmode(GPIO.BCM)
GPIO.setup(17,GPIO.IN,pull_up_down=GPIO.PUD_UP)
GPIO.setup(22,GPIO.IN,pull_up_down=GPIO.PUD_UP)
GPIO.setup(23,GPIO.IN,pull_up_down=GPIO.PUD_UP)
GPIO.setup(27,GPIO.IN,pull_up_down=GPIO.PUD_UP)
GPIO.setup(19,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
GPIO.setup(26,GPIO.IN,pull_up_down=GPIO.PUD_UP)

while True:
    time.sleep(0.2)
    #pressing GPIO17 will pause the video
    if (not GPIO.input(17)):
        print subprocess.check_output('echo "pause" > /home/pi/video_fifo', shell=True)
    #pressing GPIO22 will fase forward 10s
    elif(not GPIO.input(22)):
        print subprocess.check_output('echo  "seek 10 0" > /home/pi/video_fifo', shell=True)
    #pressing GPIO23 will rewind 10s
    elif(not GPIO.input(23)):
        print subprocess.check_output('echo  "seek -10 0" > /home/pi/video_fifo', shell=True)
    #pressing GPIO27 will quit the mplayer
    elif(not GPIO.input(27)):
        print subprocess.check_output('echo  "quit [0]" > /home/pi/video_fifo', shell=True)
        break
    elif(GPIO.input(19)):
        print subprocess.check_output('echo  "seek 30 0" > /home/pi/video_fifo', shell=True)
    #pressing GPIO23 will fast forward 30s
    elif(not GPIO.input(26)):
        print subprocess.check_output('echo  "seek -30 0" > /home/pi/video_fifo', shell=True)
    #pressing GPIO27 will rewind 30s
```

Figure 3 Python code for more_video_control_perf.py



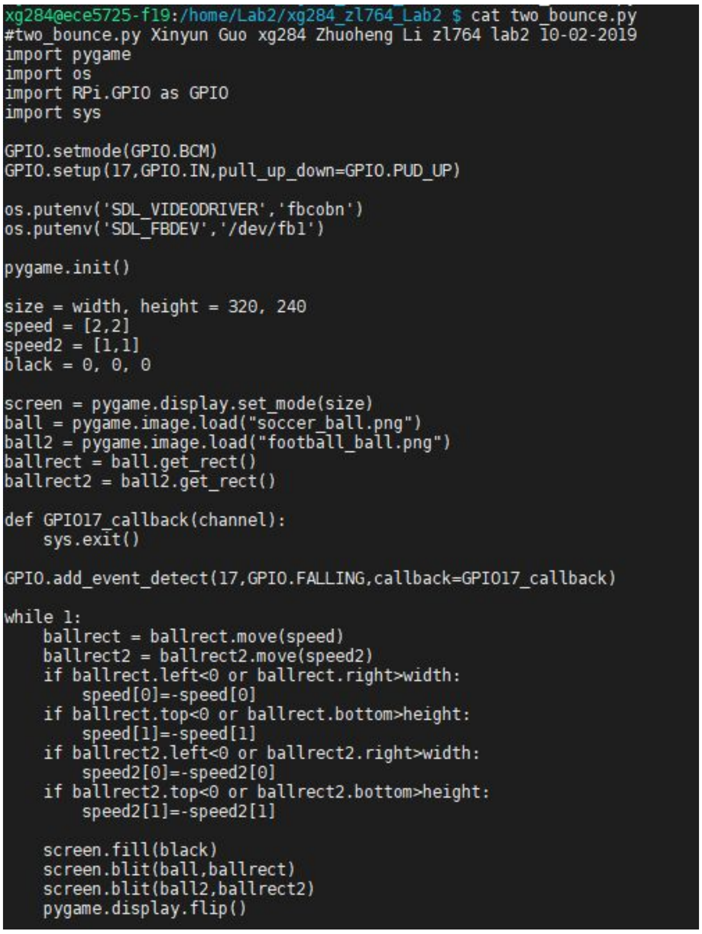
```
#more_video_control_cb_perf.py Xinyun Guo xg284 Zhuoheng Li zl764 lab2 10-02-2019
import RPi.GPIO as GPIO
import time
import subprocess
start= time.time()
while time.time()-start<10:
    time.sleep(0.2)
    #set up the fmustour GPIOs
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(17,GPIO.IN,pull_up_down=GPIO.PUD_UP)
    GPIO.setup(22,GPIO.IN,pull_up_down=GPIO.PUD_UP)
    GPIO.setup(23,GPIO.IN,pull_up_down=GPIO.PUD_UP)
    GPIO.setup(27,GPIO.IN,pull_up_down=GPIO.PUD_UP)
    GPIO.setup(19,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
    GPIO.setup(26,GPIO.IN,pull_up_down=GPIO.PUD_UP)
    #print ("before callback")
    def GPIO17_callback(channel):
        subprocess.check_output('echo  "pause" > /home/pi/video_fifo', shell=True)
    def GPIO22_callback(channel):
        subprocess.check_output('echo "seek 10 0" > /home/pi/video_fifo', shell=True)
    def GPIO23_callback(channel):
        subprocess.check_output('echo "seek -10 0" > /home/pi/video_fifo', shell=True)
    def GPIO19_callback(channel):
        subprocess.check_output('echo "seek 30 0" > /home/pi/video_fifo', shell=True)
    def GPIO26_callback(channel):
        subprocess.check_output('echo "seek -30 0" > /home/pi/video_fifo', shell=True)
   # print("add the event")
    GPIO.add_event_detect(17,GPIO.FALLING,callback=GPIO17_callback)
    GPIO.add_event_detect(22,GPIO.FALLING,callback=GPIO22_callback)
    GPIO.add_event_detect(23,GPIO.FALLING,callback=GPIO23_callback)
    GPIO.add_event_detect(19,GPIO.FALLING,callback=GPIO19_callback)
    GPIO.add_event_detect(26,GPIO.FALLING,callback=GPIO26_callback)
   # print ("this is before GPIO waiting")
    #the program will wait here until the button is clicked
    try:
        if time.time() - start>10:
            break
        #GPIO.wait_for_edge(27,GPIO.FALLING)
        #subprocess.check_output('echo "quit[0]" > /home/pi/video_fifo', shell=True)
    except KeyboardInterrupt:
        GPIO.cleanup()
        #subprocess.check_output('echo "quit [0]" > /home/pi/video_fifo', shell=True)
    GPIO.cleanup()

    #subprocess.check_output('echo "quit[0]" > /home/pi/video_fifo', shell=True)
print("this is the end of the loop")
```

Figure 4 Python code for more_video_control_cb_perf.py



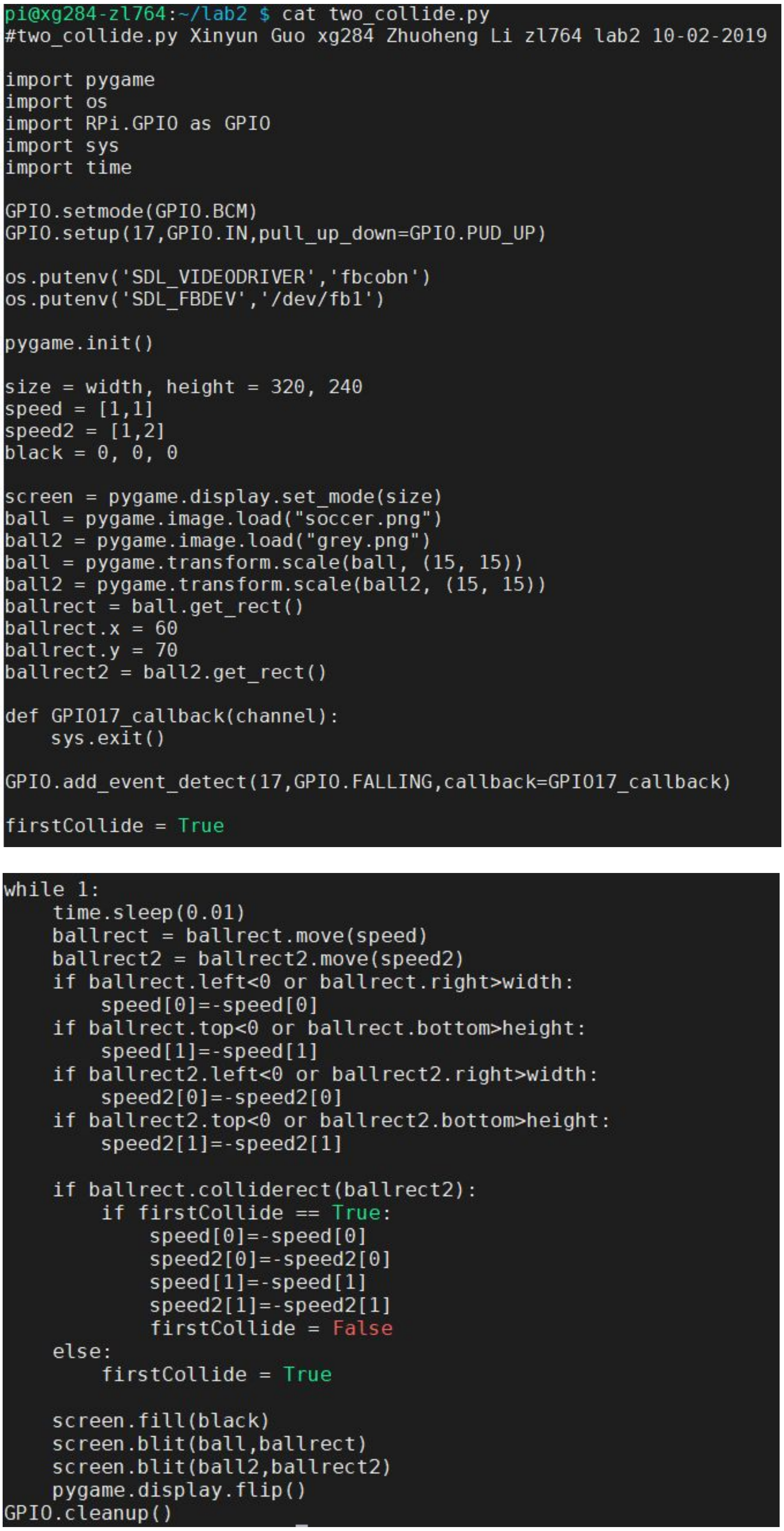
```
#bounce.py Xinyun Guo xg284 Zhuoheng Li zl764 lab2 10-02-2019

import pygame
import os
import RPi.GPIO as GPIO
import sys

GPIO.setmode(GPIO.BCM)
GPIO.setup(17,GPIO.IN,pull_up_down=GPIO.PUD_UP)

os.putenv('SDL_VIDEODRIVER','fbcobn')
os.putenv('SDL_FBDEV','/dev/fb1')

pygame.init()

size = width, height = 320, 240
speed = [2,2]
black = 0, 0, 0

screen = pygame.display.set_mode(size)
ball = pygame.image.load("soccer_ball.png")
ballrect = ball.get_rect()

def GPIO17_callback(channel):
    sys.exit()

GPIO.add_event_detect(17,GPIO.FALLING,callback=GPIO17_callback)

while 1:
    ballrect = ballrect.move(speed)
    if ballrect.left<0 or ballrect.right>width:
        speed[0]=-speed[0]
    if ballrect.top<0 or ballrect.bottom>height:
        speed[1]=-speed[1]

    screen.fill(black)
    screen.blit(ball,ballrect)
    pygame.display.flip()
```

Figure 5 code for bounce.py



```
xg284@ece5725-f19:/home/Lab2/xg284_zl764_Lab2 $ cat two_bounce.py
#two_bounce.py Xinyun Guo xg284 Zhuoheng Li zl764 lab2 10-02-2019
import pygame
import os
import RPi.GPIO as GPIO
import sys

GPIO.setmode(GPIO.BCM)
GPIO.setup(17,GPIO.IN,pull_up_down=GPIO.PUD_UP)

os.putenv('SDL_VIDEODRIVER','fbcobn')
os.putenv('SDL_FBDEV','/dev/fb1')

pygame.init()

size = width, height = 320, 240
speed = [2,2]
speed2 = [1,1]
black = 0, 0, 0

screen = pygame.display.set_mode(size)
ball = pygame.image.load("soccer_ball.png")
ball2 = pygame.image.load("football_ball.png")
ballrect = ball.get_rect()
ballrect2 = ball2.get_rect()

def GPIO17_callback(channel):
    sys.exit()

GPIO.add_event_detect(17,GPIO.FALLING,callback=GPIO17_callback)

while 1:
    ballrect = ballrect.move(speed)
    ballrect2 = ballrect2.move(speed2)
    if ballrect.left<0 or ballrect.right>width:
        speed[0]=-speed[0]
    if ballrect.top<0 or ballrect.bottom>height:
        speed[1]=-speed[1]
    if ballrect2.left<0 or ballrect2.right>width:
        speed2[0]=-speed2[0]
    if ballrect2.top<0 or ballrect2.bottom>height:
        speed2[1]=-speed2[1]

    screen.fill(black)
    screen.blit(ball,ballrect)
    screen.blit(ball2,ballrect2)
    pygame.display.flip()
```

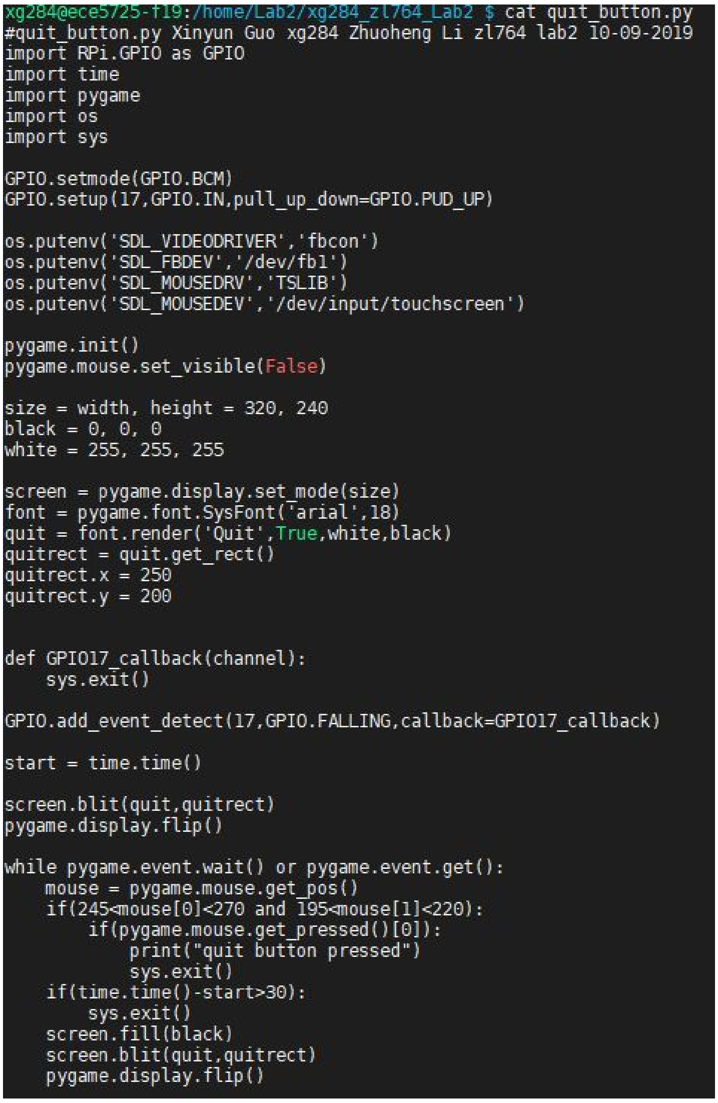
Figure 6 two_bounce.py



```
pi@xg284-zl764:~/lab2 $ cat two_collide.py
#two_collide.py Xinyun Guo xg284 Zhuoheng Li zl764 lab2 10-02-2019

import pygame
import os
import RPi.GPIO as GPIO
import sys
import time

GPIO.setmode(GPIO.BCM)
GPIO.setup(17,GPIO.IN,pull_up_down=GPIO.PUD_UP)

os.putenv('SDL_VIDEODRIVER','fbcobn')
os.putenv('SDL_FBDEV','/dev/fb1')

pygame.init()

size = width, height = 320, 240
speed = [1,1]
speed2 = [1,2]
black = 0, 0, 0

screen = pygame.display.set_mode(size)
ball = pygame.image.load("soccer.png")
ball2 = pygame.image.load("grey.png")
ball = pygame.transform.scale(ball, (15, 15))
ball2 = pygame.transform.scale(ball2, (15, 15))
ballrect = ball.get_rect()
ballrect.x = 60
ballrect.y = 70
ballrect2 = ball2.get_rect()

def GPIO17_callback(channel):
    sys.exit()

GPIO.add_event_detect(17,GPIO.FALLING,callback=GPIO17_callback)

firstCollide = True
```

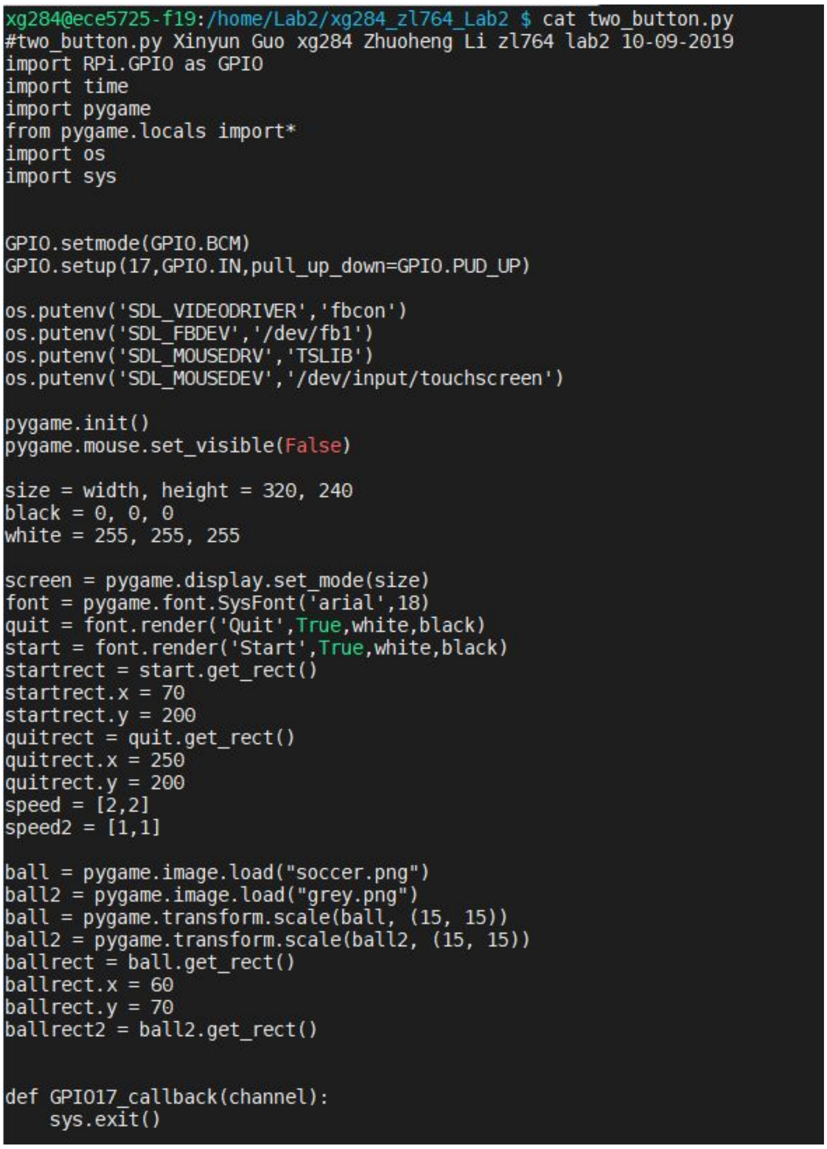
```
while 1:
    time.sleep(0.01)
    ballrect = ballrect.move(speed)
    ballrect2 = ballrect2.move(speed2)
    if ballrect.left<0 or ballrect.right>width:
        speed[0]=-speed[0]
    if ballrect.top<0 or ballrect.bottom>height:
        speed[1]=-speed[1]
    if ballrect2.left<0 or ballrect2.right>width:
        speed2[0]=-speed2[0]
    if ballrect2.top<0 or ballrect2.bottom>height:
        speed2[1]=-speed2[1]

    if ballrect.colliderect(ballrect2):
        if firstCollide == True:
            speed[0]=-speed[0]
            speed2[0]=-speed2[0]
            speed[1]=-speed[1]
            speed2[1]=-speed2[1]
            firstCollide = False
    else:
        firstCollide = True

    screen.fill(black)
    screen.blit(ball,ballrect)
    screen.blit(ball2,ballrect2)
    pygame.display.flip()
GPIO.cleanup()
```

Figure 7 two_collide.py

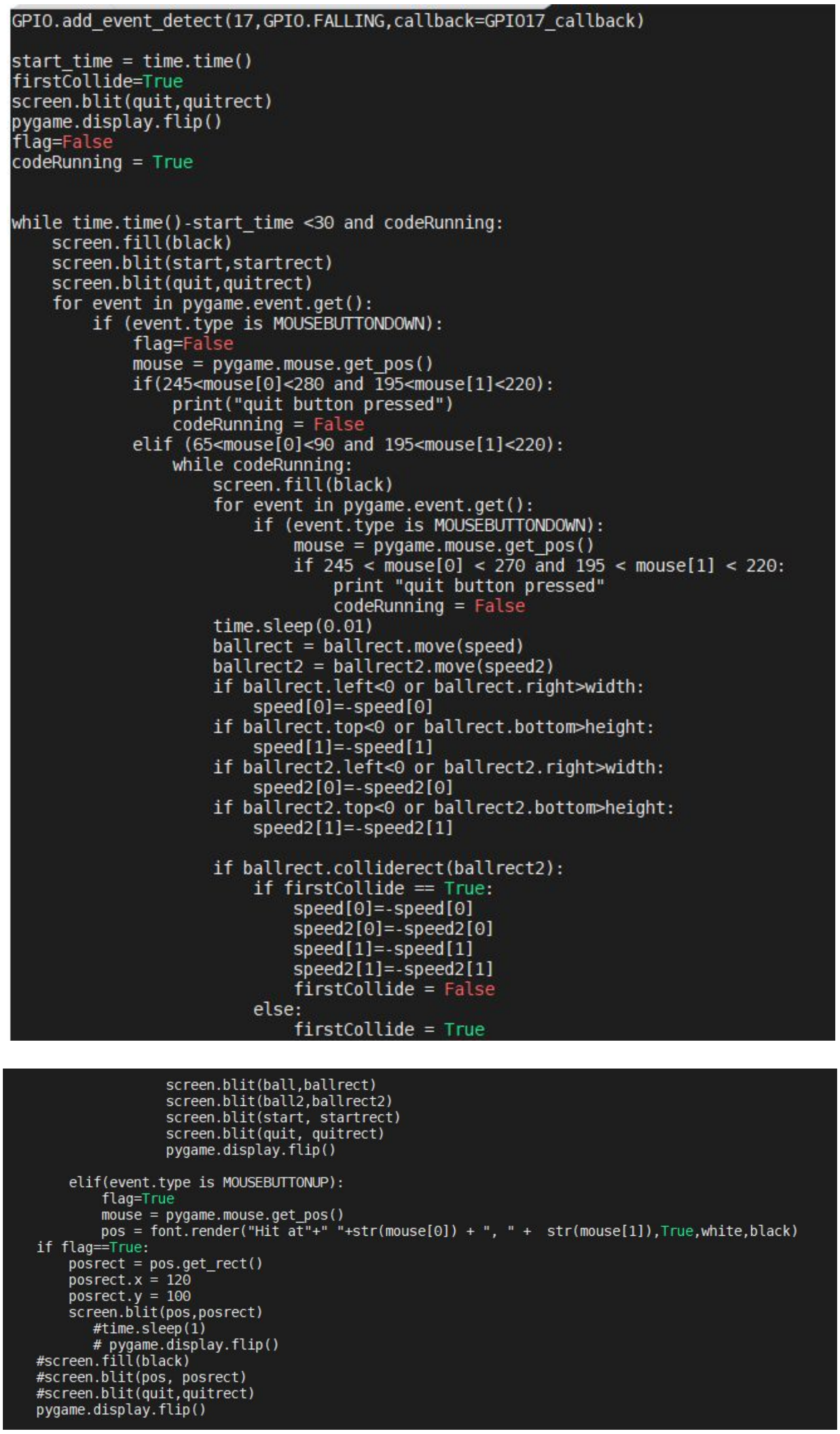


```
xg284@ece5725-f19:/home/Lab2/xg284_zl764_Lab2 $ cat quit_button.py
#quit_button.py Xinyun Guo xg284 Zhuoheng Li zl764 lab2 10-09-2019
import RPi.GPIO as GPIO
import time
import pygame
import os
import sys

GPIO.setmode(GPIO.BCM)
GPIO.setup(17,GPIO.IN,pull_up_down=GPIO.PUD_UP)

os.putenv('SDL_VIDEODRIVER','fbcon')
os.putenv('SDL_FBDEV','/dev/fb1')
os.putenv('SDL_MOUSEDRV','TSLIB')
os.putenv('SDL_MOUSEDEV','/dev/input/touchscreen')

pygame.init()
pygame.mouse.set_visible(False)

size = width, height = 320, 240
black = 0, 0, 0
white = 255, 255, 255

screen = pygame.display.set_mode(size)
font = pygame.font.SysFont('arial',18)
quit = font.render('Quit',True,white,black)
quitrect = quit.get_rect()
quitrect.x = 250
quitrect.y = 200


def GPIO17_callback(channel):
    sys.exit()

GPIO.add_event_detect(17,GPIO.FALLING,callback=GPIO17_callback)

start = time.time()

screen.blit(quit,quitrect)
pygame.display.flip()

while pygame.event.wait() or pygame.event.get():
    mouse = pygame.mouse.get_pos()
    if(245<mouse[0]<270 and 195<mouse[1]<220):
        if(pygame.mouse.get_pressed()[0]):
            print("quit button pressed")
            sys.exit()
    if(time.time()-start>30):
        sys.exit()
    screen.fill(black)
    screen.blit(quit,quitrect)
    pygame.display.flip()
```

Figure 8 quit_button.py



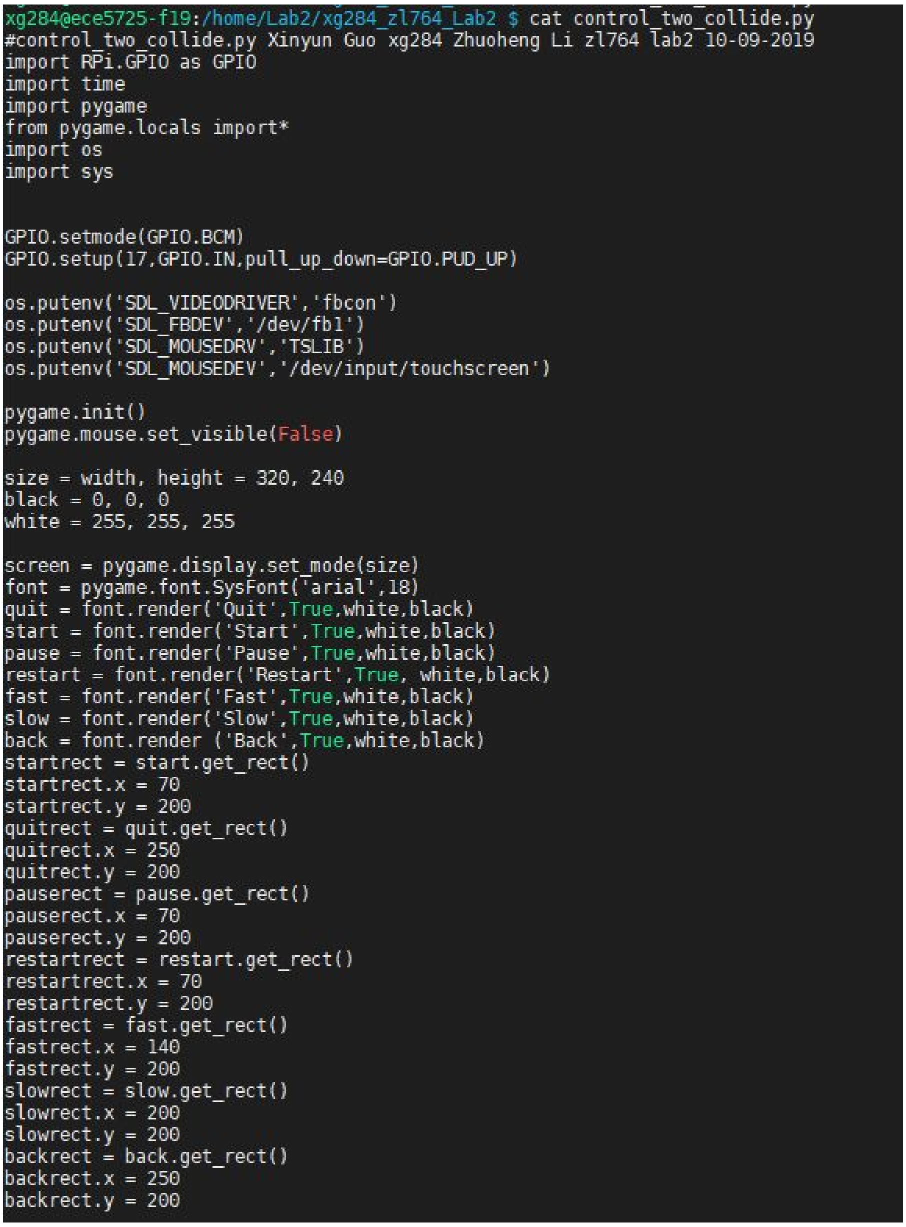
```
xg284@ece5725-f19:/home/Lab2/xg284_zl764_Lab2 $ cat two_button.py
#two_button.py Xinyun Guo xg284 Zhuoheng Li zl764 lab2 10-09-2019
import RPi.GPIO as GPIO
import time
import pygame
from pygame.locals import*
import os
import sys


GPIO.setmode(GPIO.BCM)
GPIO.setup(17,GPIO.IN,pull_up_down=GPIO.PUD_UP)

os.putenv('SDL_VIDEODRIVER','fbcon')
os.putenv('SDL_FBDEV','/dev/fb1')
os.putenv('SDL_MOUSEDRV','TSLIB')
os.putenv('SDL_MOUSEDEV','/dev/input/touchscreen')

pygame.init()
pygame.mouse.set_visible(False)

size = width, height = 320, 240
black = 0, 0, 0
white = 255, 255, 255

screen = pygame.display.set_mode(size)
font = pygame.font.SysFont('arial',18)
quit = font.render('Quit',True,white,black)
start = font.render('Start',True,white,black)
startrect = start.get_rect()
startrect.x = 70
startrect.y = 200
quitrect = quit.get_rect()
quitrect.x = 250
quitrect.y = 200
speed = [2,2]
speed2 = [1,1]

ball = pygame.image.load("soccer.png")
ball2 = pygame.image.load("grey.png")
ball = pygame.transform.scale(ball, (15, 15))
ball2 = pygame.transform.scale(ball2, (15, 15))
ballrect = ball.get_rect()
ballrect.x = 60
ballrect.y = 70
ballrect2 = ball2.get_rect()


def GPIO17_callback(channel):
    sys.exit()
```



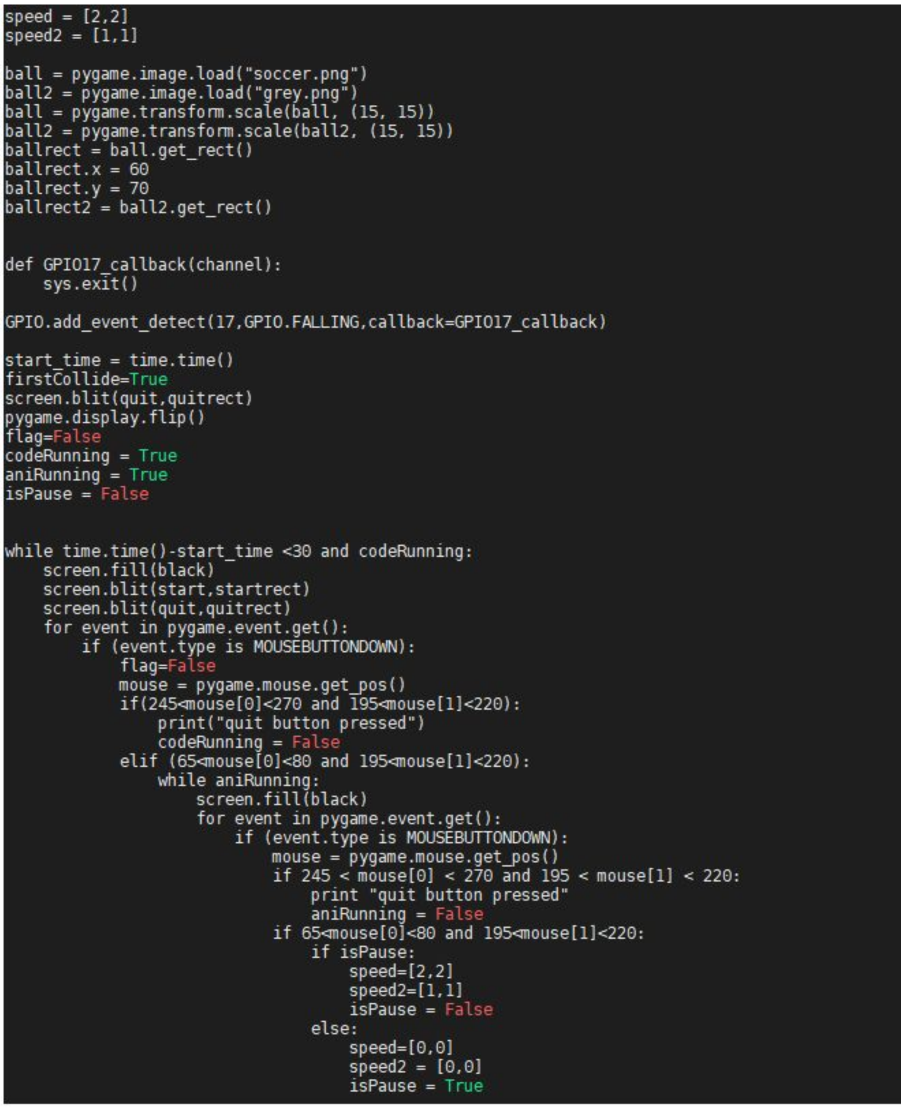
```
GPIO.add_event_detect(17,GPIO.FALLING,callback=GPIO17_callback)

start_time = time.time()
firstCollide=True
screen.blit(quit,quitrect)
pygame.display.flip()
flag=False
codeRunning = True


while time.time()-start_time <30 and codeRunning:
    screen.fill(black)
    screen.blit(start,startrect)
    screen.blit(quit,quitrect)
    for event in pygame.event.get():
        if (event.type is MOUSEBUTTONDOWN):
            flag=False
            mouse = pygame.mouse.get_pos()
            if(245<mouse[0]<280 and 195<mouse[1]<220):
                print("quit button pressed")
                codeRunning = False
            elif (65<mouse[0]<90 and 195<mouse[1]<220):
                while codeRunning:
                    screen.fill(black)
                    for event in pygame.event.get():
                        if (event.type is MOUSEBUTTONDOWN):
                            mouse = pygame.mouse.get_pos()
                            if 245 < mouse[0] < 270 and 195 < mouse[1] < 220:
                                print "quit button pressed"
                                codeRunning = False
                    time.sleep(0.01)
                    ballrect = ballrect.move(speed)
                    ballrect2 = ballrect2.move(speed2)
                    if ballrect.left<0 or ballrect.right>width:
                        speed[0]=-speed[0]
                    if ballrect.top<0 or ballrect.bottom>height:
                        speed[1]=-speed[1]
                    if ballrect2.left<0 or ballrect2.right>width:
                        speed2[0]=-speed2[0]
                    if ballrect2.top<0 or ballrect2.bottom>height:
                        speed2[1]=-speed2[1]

                    if ballrect.colliderect(ballrect2):
                        if firstCollide == True:
                            speed[0]=-speed[0]
                            speed2[0]=-speed2[0]
                            speed[1]=-speed[1]
                            speed2[1]=-speed2[1]
                            firstCollide = False
                        else:
                            firstCollide = True
```

```
                screen.blit(ball,ballrect)
                screen.blit(ball2,ballrect2)
                screen.blit(start, startrect)
                screen.blit(quit, quitrect)
                pygame.display.flip()

    elif(event.type is MOUSEBUTTONUP):
        flag=True
        mouse = pygame.mouse.get_pos()
        pos = font.render("Hit at"+" "+str(mouse[0]) + ", " +  str(mouse[1]),True,white,black)
if flag==True:
    posrect = pos.get_rect()
    posrect.x = 120
    posrect.y = 100
    screen.blit(pos,posrect)
       #time.sleep(1)
       # pygame.display.flip()
#screen.fill(black)
#screen.blit(pos, posrect)
#screen.blit(quit,quitrect)
pygame.display.flip()
```

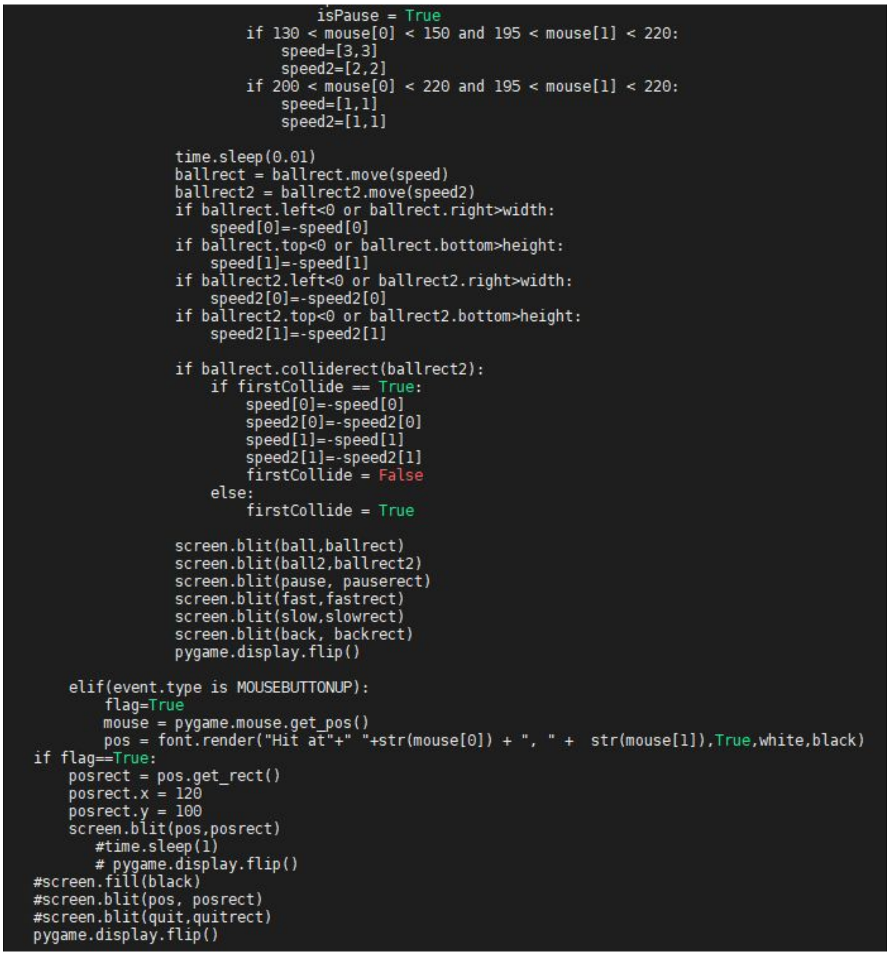
Figure 9 two_button.py



```
xg284@ece5725-f19:/home/Lab2/xg284_zl764_Lab2 $ cat control_two_collide.py
#control_two_collide.py Xinyun Guo xg284 Zhuoheng Li zl764 lab2 10-09-2019
import RPi.GPIO as GPIO
import time
import pygame
from pygame.locals import*
import os
import sys


GPIO.setmode(GPIO.BCM)
GPIO.setup(17,GPIO.IN,pull_up_down=GPIO.PUD_UP)

os.putenv('SDL_VIDEODRIVER','fbcon')
os.putenv('SDL_FBDEV','/dev/fb1')
os.putenv('SDL_MOUSEDRV','TSLIB')
os.putenv('SDL_MOUSEDEV','/dev/input/touchscreen')

pygame.init()
pygame.mouse.set_visible(False)

size = width, height = 320, 240
black = 0, 0, 0
white = 255, 255, 255

screen = pygame.display.set_mode(size)
font = pygame.font.SysFont('arial',18)
quit = font.render('Quit',True,white,black)
start = font.render('Start',True,white,black)
pause = font.render('Pause',True,white,black)
restart = font.render('Restart',True, white,black)
fast = font.render('Fast',True,white,black)
slow = font.render('Slow',True,white,black)
back = font.render ('Back',True,white,black)
startrect = start.get_rect()
startrect.x = 70
startrect.y = 200
quitrect = quit.get_rect()
quitrect.x = 250
quitrect.y = 200
pauserect = pause.get_rect()
pauserect.x = 70
pauserect.y = 200
restartrect = restart.get_rect()
restartrect.x = 70
restartrect.y = 200
fastrect = fast.get_rect()
fastrect.x = 140
fastrect.y = 200
slowrect = slow.get_rect()
slowrect.x = 200
slowrect.y = 200
backrect = back.get_rect()
backrect.x = 250
backrect.y = 200
```



```
speed = [2,2]
speed2 = [1,1]

ball = pygame.image.load("soccer.png")
ball2 = pygame.image.load("grey.png")
ball = pygame.transform.scale(ball, (15, 15))
ball2 = pygame.transform.scale(ball2, (15, 15))
ballrect = ball.get_rect()
ballrect.x = 60
ballrect.y = 70
ballrect2 = ball2.get_rect()


def GPIO17_callback(channel):
    sys.exit()

GPIO.add_event_detect(17,GPIO.FALLING,callback=GPIO17_callback)

start_time = time.time()
firstCollide=True
screen.blit(quit,quitrect)
pygame.display.flip()
flag=False
codeRunning = True
aniRunning = True
isPause = False


while time.time()-start_time <30 and codeRunning:
    screen.fill(black)
    screen.blit(start,startrect)
    screen.blit(quit,quitrect)
    for event in pygame.event.get():
        if (event.type is MOUSEBUTTONDOWN):
            flag=False
            mouse = pygame.mouse.get_pos()
            if(245<mouse[0]<270 and 195<mouse[1]<220):
                print("quit button pressed")
                codeRunning = False
            elif (65<mouse[0]<80 and 195<mouse[1]<220):
                while aniRunning:
                    screen.fill(black)
                    for event in pygame.event.get():
                        if (event.type is MOUSEBUTTONDOWN):
                            mouse = pygame.mouse.get_pos()
                            if 245 < mouse[0] < 270 and 195 < mouse[1] < 220:
                                print "quit button pressed"
                                aniRunning = False
                            if 65<mouse[0]<80 and 195<mouse[1]<220:
                                if isPause:
                                    speed=[2,2]
                                    speed2=[1,1]
                                    isPause = False
                                else:
                                    speed=[0,0]
                                    speed2 = [0,0]
                                    isPause = True
```



```
                                isPause = True
                        if 130 < mouse[0] < 150 and 195 < mouse[1] < 220:
                            speed=[3,3]
                            speed2=[2,2]
                        if 200 < mouse[0] < 220 and 195 < mouse[1] < 220:
                            speed=[1,1]
                            speed2=[1,1]

                time.sleep(0.01)
                ballrect = ballrect.move(speed)
                ballrect2 = ballrect2.move(speed2)
                if ballrect.left<0 or ballrect.right>width:
                    speed[0]=-speed[0]
                if ballrect.top<0 or ballrect.bottom>height:
                    speed[1]=-speed[1]
                if ballrect2.left<0 or ballrect2.right>width:
                    speed2[0]=-speed2[0]
                if ballrect2.top<0 or ballrect2.bottom>height:
                    speed2[1]=-speed2[1]

                if ballrect.colliderect(ballrect2):
                    if firstCollide == True:
                        speed[0]=-speed[0]
                        speed2[0]=-speed2[0]
                        speed[1]=-speed[1]
                        speed2[1]=-speed2[1]
                        firstCollide = False
                    else:
                        firstCollide = True

                screen.blit(ball,ballrect)
                screen.blit(ball2,ballrect2)
                screen.blit(pause, pauserect)
                screen.blit(fast,fastrect)
                screen.blit(slow,slowrect)
                screen.blit(back, backrect)
                pygame.display.flip()

    elif(event.type is MOUSEBUTTONUP):
        flag=True
        mouse = pygame.mouse.get_pos()
        pos = font.render("Hit at"+" "+str(mouse[0]) + ", " +  str(mouse[1]),True,white,black)
if flag==True:
    posrect = pos.get_rect()
    posrect.x = 120
    posrect.y = 100
    screen.blit(pos,posrect)
       #time.sleep(1)
       # pygame.display.flip()
#screen.fill(black)
#screen.blit(pos, posrect)
#screen.blit(quit,quitrect)
pygame.display.flip()
```

Figure 10 control_two_collide.py