Python: Collisions

Computer Science ICS20

Collisions

Detecting and handling collisions between various objects is a basic requirement of most games and simulations. However, it can also be very challenging to program properly. In this section we will explore the basics of collision detection and how to handle them if they occur.

As you will recall, our basic template for a general animated program that runs at 60 fps is shown below:

```
import pygame
pygame.init()     # initializes the graphics module
window = pygame.display.set_mode((800,600))    # define window size
pygame.display.set_caption('PyGame Collisions')# title of program that
                                               # appears on window frame
clock = pygame.time.Clock()     # used to track time within the game (FPS)
quit = False
while not quit:                              # main program loop
      for event in pygame.event.get():       # check if there were any events
            if event.type == pygame.QUIT:    # check if user clicked the upper
                  quit = True                # right quit button

      # your code that draws to the window goes here
      pygame.draw.circle(window, red, (100, 100), 50)


      pygame.display.update()       # refresh your display
      clock.tick(60)                # wait a certain amount of time that
                                    # ensures a frame rate of 60 fps
pygame.quit()                       # shutdown module
```

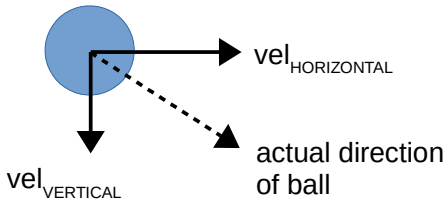
For this example, we will use two circle objects that have random initial positions and velocities. Each also have their own colour and size. For these characteristics we can use the following lists to store all of our variables:

ball_pos = [[x_1, y_1], [x_2, y_2]] ball_vel = [[x_1, y_1]), [x_2, y_2]]

ball_colour = [(R_1, G_1, B_1), (R_2, G_2, B_2)] ball_radius = [R_1, R_2]

Colliding a Circle with the Window Border

We already saw how to handle collisions with the window borders. To review, each ball has a velocity made up of 2 **components**: vertical and horizontal. Together, these components define not only **speed** but **direction** as well. For example, if both the horizontal and vertical speeds are 5, then the ball will move in a -45° angle. We will use the term **velocity** to encapsulate both the direction and the speed.



All movement is made up of 2 components: horizontal and vertical

As the ball moves around the window at a certain velocity, it will eventually collide with a window boundary. This type of collision is not too difficult to detect and handle because the edges of the window are horizontal or vertical, making the math straight forward. For example, if the ball moves horizontally, we could write an "if statement" to check if its position is the same as the right edge (which should be the WINDOW_WIDTH):

BALL 1

```
ball_pos[0][0] = ball_pos[0][0] + ball_vel[0][0]
ball_pos[0][1] = ball_pos[0][1] + ball_vel[0][1]
if ball_pos[0][0] + ball_radius[0] >= WINDOW_WIDTH:
      ball_vel[0][0] = - ball_vel[0][0]
```

Let's examine this code in a bit more detail. Notice that the first index is zero for ball1 (it would be one for ball2.) For either ball, if its position is equal to the right edge then we have **detected** a collision and we must then decide how to **handle** the collision. In the above case we decide to bounce the ball in the opposite direction by making the horizontal velocity negative. Notice that we compare the position of the ball + radius to see if it is greater than or equal to the right edge. We cannot just compare if they are equal because, depending on the speed of the ball, the position of the ball may not land exactly on the boundary.

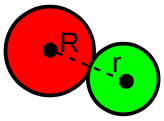
We could do the same thing for the other edge on the left as well as the top and bottom edges:

```
if ball_pos[0][0] - ball_radius[0]<= 0:  # check for collision with left edge
      ball_vel[0][0] = - ball_vel[0][0]
if ball_pos[0][1] - ball_radius[0] <= 0: # check for collision with top edge
      ball_vel[0][1] = - ball_vel[0][1]
elif ball_pos[0][1] + ball_radius[0] >= WINDOW_HEIGHT: # check for collision with
                                                       # bottom edge
      ball_vel[0][1] = - ball_vel[0][1]
```

Collisions Between Circles

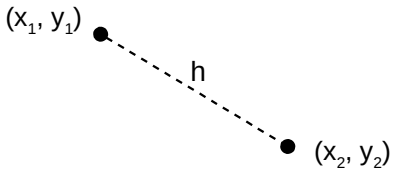
The next type of collision that we will look at are collisions between 2 (or more) balls. It is easy to see how a collision can be detected by drawing the two balls:

before collision

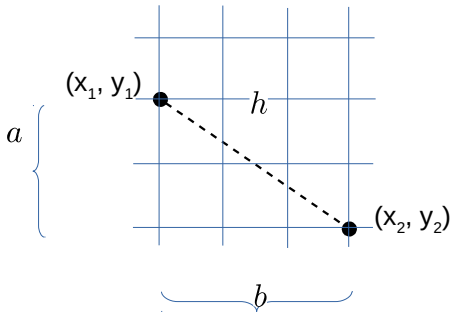


during collision

Notice that at the point of the collision, the distance between the centre of the red circle and the centre of the green circle is a minimum value of R + r. So all that is needed to detect a collision is to check whether the distance between the two centres is less than or equal to R + r. So we need to be able to determine this distance. If we focus only on the line between the two centres we get the following:



We need to determine what the distance between the two points is. In order to do this, we note that the (x, y) coordinates are on a Cartesian plane. So let's draw that in as well:



It is clear now that the distance, h, is simply the hypotenuse of a right angle triangle. So we can use the Pythagorean Theorem if only we knew the side lengths. But we can obtain the side lengths as follows:

vertical side: $a = y_2 - y_1$

horizontal side: $b = x_2 - x_1$

So, finally we can write an equation for h:

$$h^2 = a^2 + b^2$$

$$h = \sqrt{a^2 + b^2} = \sqrt{(y_2 - y_1)^2 + (x_2 - x_1)^2}$$

And now we can write our collision code for two balls:

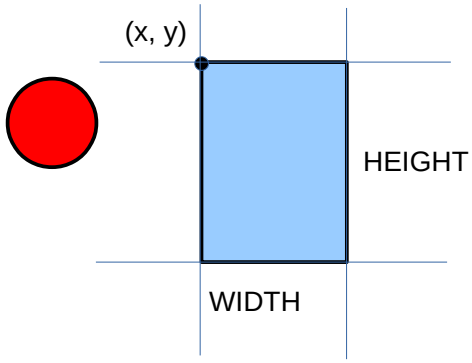
```
# distance() function returns the distance between the 2 given points
def distance(pos1, pos2):
   return math.sqrt((pos2[0] -  pos1[0])**2 + (pos2[1] -  pos1[1])**2)

ball1_pos = (ball_pos[0][0], ball_pos[0][1])
ball2_pos = (ball_pos[1][0], ball_pos[1][1])

if distance(ball1_pos, ball2_pos) <= ball_radius[0] + ball_radius[1]: # collision!
     # handle collision code goes here...how would you bounce them off each other?
```

Collisions with a Circle and a Rectangle

The idea that we used to bounce the ball off the window edges can be extended to any axis-aligned rectangular object. Suppose for example that we have a rectangular object of WIDTH and HEIGHT located at position (x, y) as defined by the top left corner as shown below:



If the ball is on the left side of the rectangle and is moving to the right, we check if the right side of the ball touches the left side of the rectangle by the following code:

```
if ball_pos[0][0] + radius[0] < rect_pos[0] and ball_vel[0][0] > 0:
       if ball_pos[0][1] + radius[0] >= rect_pos[1] and
                                    ball_pos[0][1] - radius[0] <= rect_pos[1] + HEIGHT:
              # collision occurred!
```

Similar code would exist for the left, top and right sides of the rectangle.

Adding Sound to the Collision

When two objects collide there is usually a sound generated. PyGame can play sounds by following 2 steps:

1. Load a sound, which is either an ogg or wav file.
2. Play the sound

Here is the code to perform the two previous steps:

```
sound = pygame.mixer.Sound("crash.wav")   # load an wav file
sound.play()                              # play the sound once
```

As an aside, if you want to play a song in the background while the game is playing the code is similar:

```
pygame.mixer.music.load("song.mp3")       # load an mp3 file
pygame.mixer.music.play()                 # play the song once
```

Passing a -1 to the above play() function will play the song endlessly. If you want to stop the song from playing type the following:

```
pygame.mixer.music.stop()
```

Exercises

1. Write a Python program that draws a ball at a random starting location and random velocity. The ball should bounce of each of the four window edges.
2. Modify your Python program from question 1 by adding a second ball of a different size and colour.
3. Modify your Python program from question 2 by implementing collision handling between the two balls.
4. Modify your Python program from question 3 by drawing a third ball of radius 20 that is always drawn where your mouse cursor is located.
5. Add collision detection between the mouse-controlled ball and the other balls. If the mouse-controlled ball collides with any of the other balls the game is over.
6. Modify your program so that the speed of the ball increases by one unit every 5 seconds.
7. Modify your Python program again so that every 10 seconds an extra ball is randomly placed in the game.
8. Finally modify your program so that a score appears on the top of the screen showing the elapsed time. See how long you can play the game!