Python Games

Session 7

By Declan Fox

Rules

"Above all, be cool."

General Information

Wi-Fi Name: CoderDojo

Password: coderdojowireless

Website: http://cdathenry.wordpress.com/

Useful Links

Recommended reading:

http://inventwithpython.com

Reference Guide

http://www.tutorialspoint.com/python/

Social Media

Like our new Facebook page at
www.facebook.com/CoderDojoAthenry

Or if you are on twitter follow us on
@coderdojoathenr

Installation

As we will be moving on to graphical games we will need to install both Python and Pygame*

* If you have Python 3.x.x and Pygame installed you can ignore the next slide

Installation

We are using version 3.2 of Python go to

https://www.python.org/download/releases/3.2.5/

Select Windows x86 MSI Installer (3.2.5)

To install Pygame go to

http://pygame.org/download.shtml

Select pygame-1.9.2a0.win32-py3.2.msi

Collision Detection

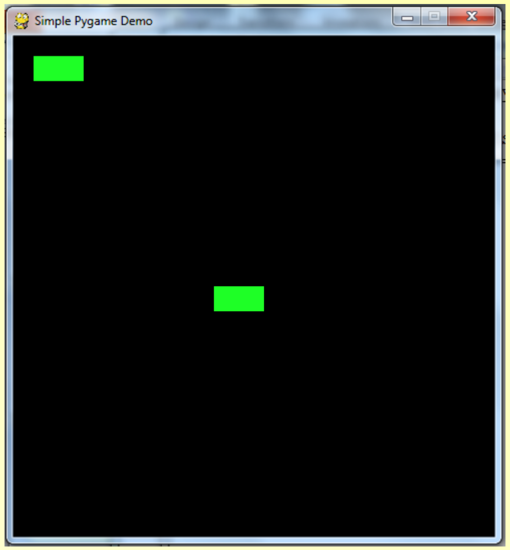
Collision detection is used to figure out if two sprites are touching each other.

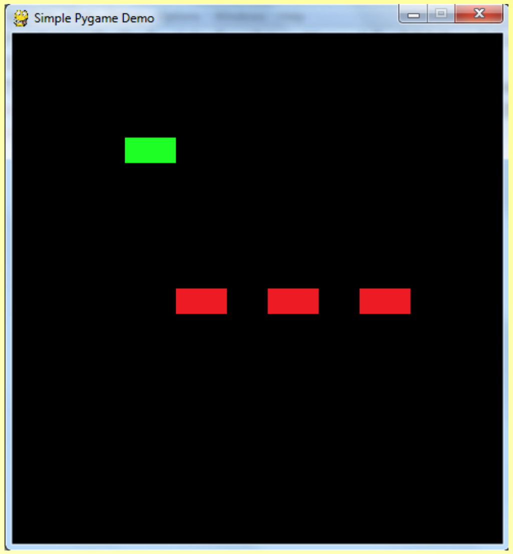
Some examples of collision Detection are

- Has the player touched an enemy
- Has the player been hit by a bullet
- Has the player touched a coin
- Is the player on solid ground

Collision Detection

Today we are going to look a two demo programs which use collision detection, lists and for loops. Our code is here.





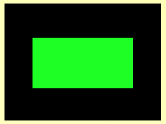
Collision Detection

Setting up our rect object

```
player_rect=pygame.Rect(player.get_rect())
player_rect.left = player_x
player_rect.top = player_y
```

Pygame uses Rect objects to store and manipulate rectangular areas. A Rect can be created from a combination of left, top, width, and height values.

Player in player.get_rect() is our player image which is the green block.



Collision Detection

```
if player_rect.colliderect(block_rect):
```

Colliderect() tests if two rectangles overlap it returns True if any portion of either rectangle overlap (except the top + bottom or left + right edges).

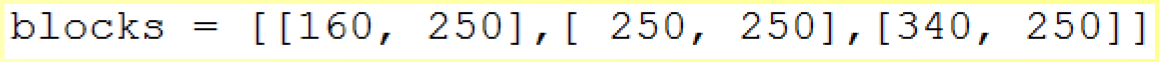
```
    player = red_block
    block = red_block
else:
    player = green_block
    block = green_block
```

If colloderect() returns a True value then the block's colour changes to red otherwise they are set to red.

Using lists and for loops

Lists are very useful in Pygame they allow us to create and destroy variables as the game is running.
Imagine we want to write a Tetris style game we could create all the blocks and position them outside the viewable area and move them in and out as we need them or we could use a list and add them as we need them and remove them when we are finished with them.

Lists in Pygame



```
blocks = [[160, 250],[ 250, 250],[340, 250]]
```

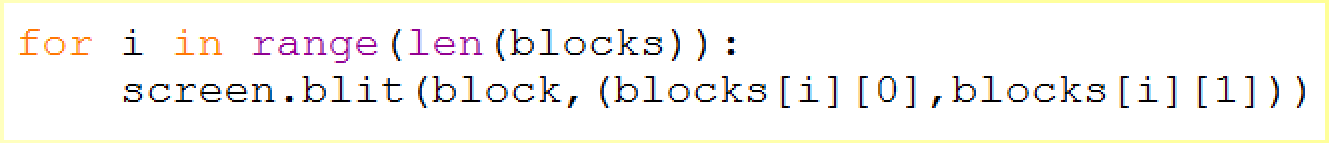
This list contains three lists of two elements we are going to use these two numbers as our x any coordinates for our blocks.

blocks[0][0] blocks[1][0] blocks[2][0] will be our x coordinates.

blocks[0][1] blocks[1][1] blocks[2][1] will be our y coordinates.

For loops in Pygame

As I’m sure you’ve started to notice when ever you have a list you nearly always have a for loop to manipulate it.

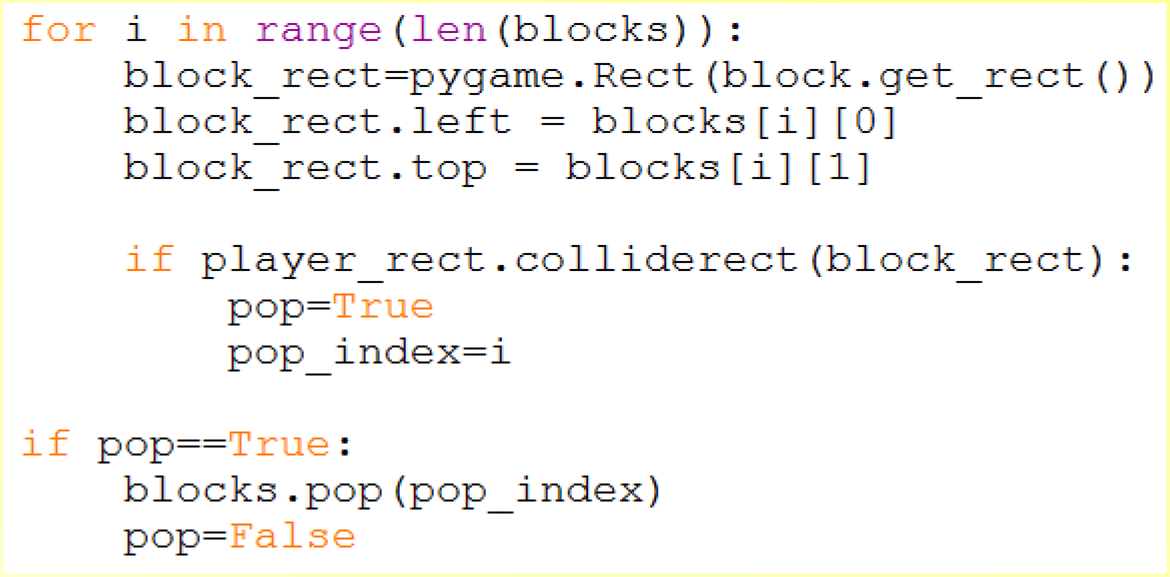


```
for i in range(len(blocks)):
    screen.blit(block,(blocks[i][0],blocks[i][1]))
```

This for loop uses the len() function to find out how many elements are in the list. Each time it runs through the loop it draws a block on the screen.

Collisions and for loops

If we wanted to remove a sprite every time the player touched it we could use the following



```
for i in range(len(blocks)):
    block_rect=pygame.Rect(block.get_rect())
    block_rect.left = blocks[i][0]
    block_rect.top = blocks[i][1]

    if player_rect.colliderect(block_rect):
        pop=True
        pop_index=i

if pop==True:
    blocks.pop(pop_index)
    pop=False
```

Collisions and for loops

First we set up our rect

```
for i in range(len(blocks)):
    block_rect=pygame.Rect(block.get_rect())
    block_rect.left = blocks[i][0]
    block_rect.top = blocks[i][1]
```

Then we use colliderect() to test for collisions

```
if player_rect.colliderect(block_rect):
    pop=True
    pop_index=i
```

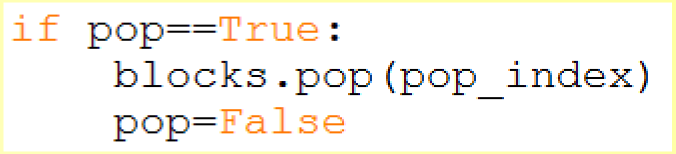
If colliderect() returns as True we set a variable called pop to True and a variable called pop_index to the amount of iterations the loop has completed.

Collisions and for loops

The .pop() method is a very handy way of removing a element from a list.
However we can’t use it inside a for loop as the for loop is expecting a set number of elements.
This is why we used the pop and pop_index variables.

Collisions and for loops

After we leave the loop we check if pop has been set to true.

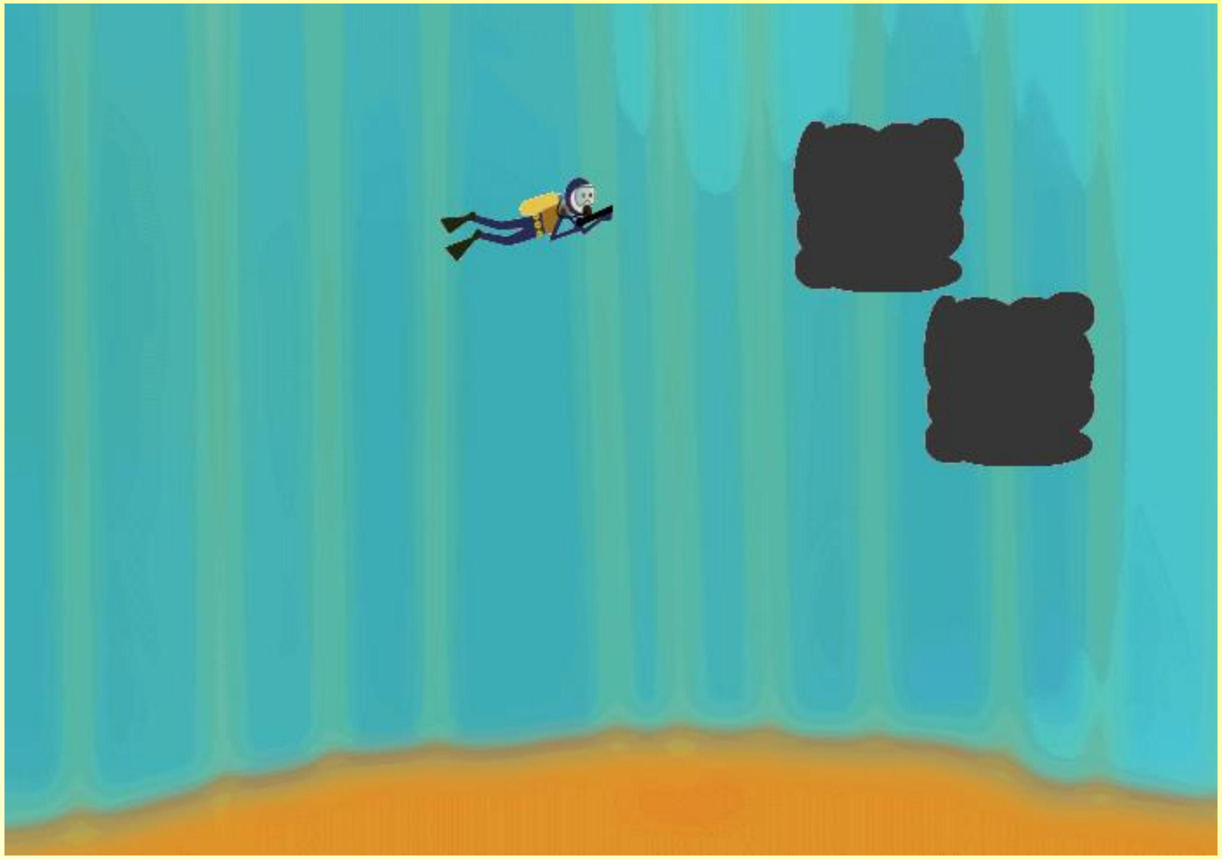


```
if pop==True:
    blocks.pop(pop_index)
    pop=False
```

If it has we use the .pop() method and pop_index to remove the block the player was touching.

This is not a game

We'll continue working on our diver game.



Next session

We're going to continue working on our ocean game, we'll also look at audio in Pygame.