Chapter 12
Building Games with pygame

12.1 Introduction

pygame is a cross-platform, free and Open Source Python library designed to make building multimedia applications such as games easy. Development of pygame started back in October 2000 with pygame version 1.0 being released six months later. The version of pygame discussed in this chapter is version 1.9.6. If you have a later version check to see what changes have been made to see if they have any impact on the examples presented here.

pygame is built on top of the SDL library. SDL (or Simple Directmedia Layer) is a cross platform development library designed to provide access to audio, keyboards, mouse, joystick and graphics hardware via OpenGL and Direct3D. To promote portability, pygame also supports a variety of additional backends including WinDIB, X11, Linux Frame Buffer etc.

SDL officially supports Windows, Mac OS X, Linux, iOS and Android (although other platforms are unofficially supported). SDL itself is written in C and pygame provides a wrapper around SDL. However, pygame adds functionality not found in SDL to make the creation of graphical or video games easier. These functions include vector maths, collision detection, 2D sprite scene graph management, MIDI support, camera, pixel array manipulation, transformations, filtering, advanced freetype font support and drawing.

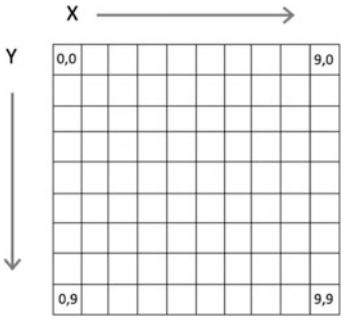
The remainder of this chapter introduces pygame, the key concepts; the key modules, classes and functions and a very simple first pygame application. The next chapter steps through the development of a simple arcade style video game which illustrates how a game can be created using pygame.

J. Hunt, *Advanced Guide to Python 3 Programming*,
Undergraduate Topics in Computer Science,
https://doi.org/10.1007/978-3-030-25943-3_12

12.2 The Display Surface

The Display Surface (aka the display) is the most important part of a pygame game. It is the main window display of your game and can be of any size, however you can only have one Display Surface.

In many ways the Display Surface is like a blank piece of paper on which you can draw. The surface itself is made up of pixels which are numbered from 0,0 in the top left hand corner with the pixel locations being indexed in the x axis and the y axis. This is shown below:



The above diagram illustrates how pixels within a Surface are indexed. Indeed a Surface can be used to draw lines, shapes (such as rectangles, squares, circles and elipses), display images, manipulate individual pixels etc. Lines are drawn from one pixel location to another (for example from location 0,0 to location 9,0 which would draw a line across the top of the above display surface). Images can be displayed within the display surface given a starting point such as 1, 1.

The Display Surface is created by the `pygame.display.set_mode()` function. This function takes a tuple that can be used to specify the size of the Display Surface to be returned. For example:

```
display_surface = pygame.display.set_mode((400, 300))
```

This will create a Display Surface (window) of 400 by 300 pixels.

Once you have the Display Surface you can fill it with an appropriate background colour (the default is black) however if you want a different background colour or want to clear everything that has previously been drawn on the surface, then you can use the surface's `fill()` method:

```
WHITE = (255, 255, 255)
display_surface.fill(WHITE)
```

The fill method takes a tuple that is used to define a colour in terms of Red, Green and Blue (or RGB) colours. Although the above examples uses a meaningful name for the tuple representing the RGB values used for white; there is of course no requirement to do this (although it is considered good practice).

To aid in performance any changes you make to the Display Surface actually happen in the background and will not be rendered onto the actual display that the user sees until you call the `update()` or `flip()` methods on the surface. For example:

- `pygame.display.update()`
- `pygame.display.flip()`

The `update()` method will redraw the display with all changes made to the display in the background. It has an optional parameter that allows you to specify just a region of the display to update (this is defined using a `Rect` which represents a rectangular area on the screen). The `flip()` method always refreshes the whole of the display (and as such does exactly the same as the `update()` method with no parameters).

Another method, which is not specifically a Display Surface method, but which is often used when the display surface is created, provides a caption or title for the top level window. This is the `pygame.display.set_caption()` function. For example:

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

This will give the top level window the caption (or title) 'Hello World'.

12.3 Events

Just as the Graphical User Interface systems described in earlier chapters have an event loop that allows the programmer to work out what the user is doing (in those cases this is typically selecting a menu item, clicking a button or entering data etc.); pygame has an event loop that allows the game to work out what the player is doing. For example, the user may press the left or right arrow key. This is represented by an event.

12.3.1 Event Types

Each event that occurs has associated information such as the type of that event. For example:

- Pressing a key will result in a `KEYDOWN` type of event, while releasing a key will result in a KEYUP event type.
- Selecting the window close button will generate a `QUIT` event type etc.
- Using the mouse can generate `MOUSEMOTION` events as well as `MOUSEBUTTONDOWN` and `MOUSEBUTTONUP` event types.

- Using a Joystick can generate several different types of event including JOYAXISMOTION, JOYBALLMOTION, JOYBUTTONDOWN and JOYBUTTONUP.

These event types tell you what occurred to generate the event. This means that you can choose which types of events you want to deal with and ignore other events.

12.3.2 Event Information

Each type of event object provides information associated with that event. For example a Key oriented event object will provide the actual key pressed while a mouse oriented event object will provide information on the position of the mouse, which button was pressed etc. If you try an access an attribute on an event that does not support that attribute, then an error will be generated.

The following lists some of the attributes available for different event types:

- KEYDOWN and KEYUP, the event has a key attribute and a mod attribute (indicating if any other modifying keys such as Shift are also being pressed).
- MOUSEBUTTONUP and MOUSEBUTTONDOWN has an attribute pos that holds a tuple indicating the mouse location in terms of x and y coordinates on the underlying surface. It also has a button attribute indicating which mouse was pressed.
- MOUSEMOTION has pos, rel and buttons attributes. The pos is a tuple indicating the x and y location of mouse cursor. The real attribute indicates the amount of mouse movement and buttons indicates the state of the mouse buttons.

As an example if we want to check for a keyboard event type and then check that the key pressed was the space bar, then we can write:

```
if event.type == pygame.KEYDOWN:
    # Check to see which key is pressed
    if event.key == pygame.K_SPACE:
        print('space')
```

This indicates that if it is a key pressed event and that the actual key was the space bar; then print the string ‘space’.

There are many keyboard constants that are used to represent the keys on the keyboard and pygame.K_SPACE constant used above is just one of them.

All the keyboard constants are prefixed with ‘K_’ followed by the key or the name of the key, for example:

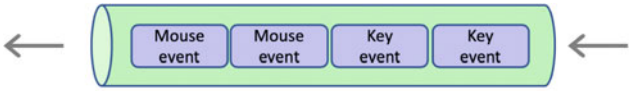
- K_TAB, K_SPACE, K_PLUS, K_0, K_1, K_AT, K_a, K_b, K_z, K_DELTE, K_DOWN, K_LEFT, K_RIGHT, K_LEFT etc.

Further keyboard constants are provided for modifier states that can be combined with the above such as `KMOD_SHIFT`, `KMOD_CAPS`, `KMOD_CTRL` and `KMOD_ALT`.

12.3.3 The Event Queue

Events are supplied to a pygame application via the Event Queue.

The Event Queue is used to collect together events as they happen. For example, let us assume that a user clicks on the mouse twice and a key twice before a program has a chance to process them; then there will be four events in the Event Queue as shown below:



The application can then obtain an iterable from the event queue and process through the events in turn. While the program is processing these events further events may occur and will be added to the Event Queue. When the program has finished processing the initial collection of events it can obtain the next set of events to process.

One significant advantage of this approach is that no events are ever lost; that is if the user clicks the mouse twice while the program is processing a previous set of events; they will be recorded and added to the event queue. Another advantage is that the events will be presented to the program in the order that they occurred.

The `pygame.event.get()` function will read all the events currently on the Event Queue (removing them from the event queue). The method returns an `EventList` which is an iterable list of the events read. Each event can then be processed in turn. For example:

```
for event in pygame.event.get():
    if event.type == pygame.QUIT:
        print('Received Quit Event:')
    elif event.type == pygame.MOUSEBUTTONDOWN:
        print('Received Mouse Event')
    elif event.type == pygame.KEYDOWN:
        print('Received KeyDown Event')
```

In the above code snippet an `EventList` is obtained from the Event Queue containing the current set of events. The for loop then processes each event in turn checking the type and printing an appropriate message.

You can use this approach to trigger appropriate behaviour such as moving an image around the screen or calculating the players score etc. However, be aware that if this behaviour takes too long it can make the game difficult to play (although the examples in this chapter and the next are simple enough that this is not a problem).

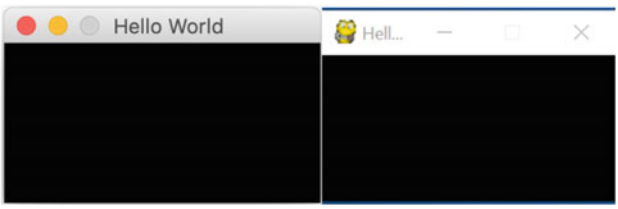
12.4 A First pygame Application

We are now at the point where we can put together what we have looked at so far and create a simple pygame application.

It is common to create a hello world style program when using a new programming language or using a new application framework etc. The intention is that the core elements of the language or framework are explored in order to generate the most basic form of an application using the language or framework. We will therefore implement the most basic application possible using pygame.

The application we will create will display a pygame window, with a 'Hello World' title. We will then be able to quit the game. Although technically speaking this isn't a game, it does possess the basic architecture of a pygame application.

The simple `HelloWorld` game will initialise pygame and the graphical display. It will then have a main game playing loop that will continue until the user selects to quit the application. It will then shut down pygame. The display created by the program is shown below for both Mac and Windows operating systems:



To quit the program click on the exit button for the windowing system you are using.

The simple HelloWorld *game* is given below:

```
import pygame

def main():
    print('Starting Game')

    print('Initialising pygame')
    pygame.init()  # Required by every pygame application

    print('Initialising HelloWorldGame')
    pygame.display.set_mode((200, 100))
    pygame.display.set_caption('Hello World')

    print('Update display')
    pygame.display.update()

    print('Starting main Game Playing Loop')
    running = True
    while running:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                print('Received Quit Event:', event)
                running = False

    print('Game Over')
    pygame.quit()

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

There are several key steps highlighted by this example, these steps are:

1. **Import pygame**. pygame is of course not one of the default modules available within Python. You must first import pygame into you code. The `import pygame` statement imports the `pygame` module into your code and makes the functions and classes in pygame available to you (note the capitalisation - pygame is not the same module name as `PyGame`). It is also common to find that programs import
   - `from pygame.locals import *`
   - This adds several constants and functions into the namespace of your program. In this very simple example we have not needed to do this.
2. **Initialise pygame**. Almost every pygame module needs to be initialised in some way and the simplest way to do this is to call `pygame.init()`. This will do what is required to set the pygame environment up for use. If you forget to call this function you will typically get an error message such as `pygame.error: video system not initialised` (or something similar). If you get such a

method check to see that you have called `pygame.init()`. Note that you can initialise individual pygame modules (for example the `pygame.font` module can be initialised using `pygame.font.init())` if required. However `pygame.init()` is the most commonly used approach to setting up pygame.

3. **Set up the display**. Once you have initialised the pygame framework you can setup the display. In the above code example, the display is set up using the `pygame.display.set_mode()` function. This function takes a tuple specifying the size of the window to be created (in this case 200 pixels wide by 100 pixels high). Note that if you try and invoke this function by passing in two parameters instead of a tuple, then you will get an error. This function returns the drawing surface or screen/window that can be used to display items within the game such as icons, messages, shapes etc. As our example is so simple we do not bother saving it into a variable. However, anything more complex than this will need to do so. We also set the window/frame's caption (or title). This is displayed in the title bar of the window.
4. **Render the display**. We now call the `pygame.display.update()` function. This function causes the current details of the display to be drawn. At the moment this is a blank window. However, it is common in games to perform a series of updates to the display in the background and then when the program is ready to update the display to call this function. This batches a series of updates and the causes the display to be refreshed. In a complex display it is possible to indicate which parts of the display need to be redrawn rather than redrawing the whole window. This is done by passing a parameter into the `update()` function to indicate the rectangle to be redrawn. However, our example is so simple we are ok with redrawing the whole window and therefore we do not need to pass any parameters to the function.
5. **Main game playing loop**. It is common to have a main game playing loop that drives the processing of user inputs, modifies the state of the game and updates the display. This is represented above by the `while running:` loop. The local variable `running` is initialised to `True`. This means that the `while` loop ensures that the game continues until the user selects to quit the game at which point the `running` variable is set to `False` which causes the loop to exit. In many cases this loop will call `update()` to refresh the display. The above example does not do this as nothing is changed in the display. However the example developed later in this chapter will illustrate this idea.
6. **Monitor for events that drive the game**. As mentioned earlier the event queue is used to allow user inputs to be queued and then processed by the game. In the simple example shown above this is represented by a for loop that receives events using `pygame.event.get()` and then checking to see if the event is a pygame.`QUIT` event. If it is, then it sets the `running` flag to `False`. Which will cause the main `while` loop of the game to terminate.
7. **Quit pygame once finished**. In pygame any module that has an `init()` function also has an equivalent `quit()` function that can be used to perform any cleanup operations. As we called `init()` on the `pygame` module at the

start of our program we will therefore need to call `pygame.quit()` at the end of the program to ensure everything is tidied up appropriately.

The output generated from a sample run of this program is given below:

```
pygame 1.9.6
Hello from the pygame community.
https://www.pygame.org/contribute.html
Starting Game
Initialising pygame
Initialising HelloWorldGame
Update display
Starting main Game Playing Loop
Received Quit Event: <Event(12-Quit {})>
Game Over
```

12.5 Further Concepts

There are very many facilities in pygame that go beyond what we can cover in this book, however a few of the more common are discussed below.

Surfaces are a hierarchy. The top level Display Surface may contain other surfaces that may be used to draw images or text. In turn containers such as Panels may render surfaces to display images or text etc.

Other types of surface. The primary Display Surface is not the only surface in pygame. For example, when an image, such as a PNG or JPEG image is loaded into a game then it is rendered onto a surface. This surface can then be displayed within another surface such as the Display Surface. This means that anything you can do to the Display Surface you can do with any other surface such as draw on it, put text on it, colour it, add another icon onto the surface etc.

Fonts. The `pygame.font.Font` object is used to create a `Font` that can be used to render text onto a surface. The render method returns a surface with the text rendered on it that can be displayed within another surface such as the Display Surface. Note that you cannot write text onto an existing surface you must always obtain a new surface (using render) and then add that to an existing surface. The text can only be displayed in a single line and the surface holding the text will be of the dimensions required to render the text. For example:

```
text_font = pygame.font.Font('freesansbold.ttf', 18)
text_surface = text_font.render('Hello World', antialias=True,
color=BLUE)
```

This creates a new `Font` object using the specified font with the specified font size (in this case 18). It will then render the string ‘Hello World’ on to a new surface using the specified font and font size in Blue. Specifying that antialias is True indicates that we would like to smooth the edges of the text on the screen.

Rectangles (or Rects). The `pygame.Rect` class is an object used to represent rectangular coordinates. A `Rect` can be created from a combination of the top left corner coordinates plus a width and height. For flexibility many functions that expect a `Rect` object can also be given a *Rectlike* list; this is a list that contains the data necessary to create a `Rect` object. Rects are very useful in a pygame `Game` as they can be used to define the borders of a game object. This means that they can be used within games to detect if two objects have collided. This is made particularly easy because the `Rect` class provides several collision detection methods:

- `pygame.Rect.contains()` test if one rectangle is inside another
- `pygame.Rect.collidepoint()` test if a point is inside a rectangle
- `pygame.Rect.colliderect()` test if two rectangles overlap
- `pygame.Rect.collidelist()` test if one rectangle in a list intersects
- `pygame.Rect.collidelistall()` test if all rectangles in a list intersect
- `pygame.Rect.collidedict()` test if one rectangle in a dictionary intersects
- `pygame.Rect.collidedictall()` test if all rectangles in a dictionary intersect

The class also provides several other utility methods such as `move()` which moves the rectangle and `inflate()` which can grow or shrink the rectangles size.

Drawing shapes. The `pygame.draw` module has numerous functions that can be used to draw lines and shapes onto a surface, for example:

```
pygame.draw.rect(display_surface, BLUE, [x, y, WIDTH, HEIGHT])
```

This will draw a filled blue rectangle (the default) onto the display surface. The rectangle will be located at the location indicated by x and y (on the surface). This indicates the top left hand corner of the rectangle. The width and height of the rectangle indicate its size. Note that these dimensions are defined within a list which is a structure referred to as being rect like (see below). If you do not want a filled rectangle (i.e. You just want the outline) then you can use the optional width parameter to indicate the thickness of the outer edge. Other methods available include:

- `pygame.draw.polygon()` draw a shape with any number of sides
- `pygame.draw.circle()` draw a circle around a point
- `pygame.draw.ellipse()` draw a round shape inside a rectangle
- `pygame.draw.arc()` draw a partial section of an ellipse
- `pygame.draw.line()` draw a straight line segment
- `pygame.draw.lines()` draw multiple contiguous line segments
- `pygame.draw.aaline()` draw fine antialiased lines
- `pygame.draw.aalines()` draw a connected sequence of antialiased lines

Images. The `pygame.image` module contains functions for loading, saving and transforming images. When an image is loaded into pygame, it is represented by a Surface object. This means that it is possible to draw, manipulate and process an image in exactly the same way as any other surface which provides a great deal of flexibility.

At a minimum the module only supports loading uncompressed BMP images but usually also supports JPEG, PNG, GIF (non-animated), BMP, TIFF as well as other formats. However, it only supports a limited set of formats when saving images; these are BMP, TGA, PNG and JPEG.

An image can be loaded from a file using:

```
image_surface = pygame.image.load(filename).convert()
```

This will load the image from the specified file onto a surface.

One thing you might wonder at is the use of the `convert()` method on the object returned from the `pygame.image.load()` function. This function returns a Surface that is used to display the image contained in the file. We call the method `convert()` on this `Surface`, not to convert the image from a particular file format (such as PNG, or JPEG) instead this method is used to convert the pixel format used by the Surface. If the pixel format used by the `Surface` is not the same as the display format, then it will need to be converted on the fly each time the image is displayed on the screen; this can be a fairly time consuming (and unnecessary) process. We therefore do this once when the image is loaded which means that it should not hinder runtime performance and may improve performance significantly on some systems.

Once you have a surface containing an image it can be rendered onto another surface, such as the display surface using the `Surface.blit()` method. For example:

```
display_surface.blit(image_surface, (x, y))
```

Note that the position argument is a tuple specifying the x and y coordinates to the image on the display surface.

Strictly speaking the `blit()` method draws one surface (the source surface) onto another surface at the destination coordinates. Thus the target surface does not beed to be the top level display surface.

Clock. A `Clock` object is an object that can be used to track time. In particular it can be used to define the frame rate for the game. That is the number of frames rendered per second. This is done using the `Clock.tick()` method. This method should be called once (and only once) per frame. If you pass the optional `framerate` argument to the `tick()` the function, then pygame will ensure that

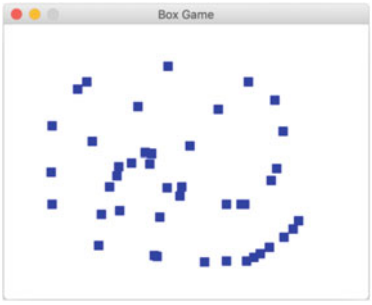
the games refresh rate is slower then the the given ticks per second. This can be used to help limit the runtime speed of a game. By calling `clock.tick(30)` once per frame, the program will never run at more than 30 frames per second.

12.6 A More Interactive pygame Application

The first pygame application we looked at earlier just displayed a window with the caption 'Hello World'. We can now extend this a little by playing with some of the features we have looked at above.

The new application will add some mouse event handling. This will allow us to pick up the location of the mouse when the user clicked on the window and draw a small blue box at that point.

If the user clicks the mouse multiple times we will get multiple blue boxes being drawn. This is shown below.



This is still not much of a game but does make the pygame application more interactive.

The program used to generate this application is presented below:

```
import pygame

FRAME_REFRESH_RATE = 30
BLUE = (0, 0, 255)
BACKGROUND = (255, 255, 255) # White
WIDTH = 10
HEIGHT = 10

def main():
    print('Initialising PyGame')
    pygame.init()  # Required by every PyGame application
```

```
    print('Initialising Box Game')
    display_surface = pygame.display.set_mode((400, 300))
    pygame.display.set_caption('Box Game')
    print('Update display')
    pygame.display.update()
    print('Setup the Clock')
    clock = pygame.time.Clock()
    # Clear the screen of current contents
    display_surface.fill(BACKGROUND)

    print('Starting main Game Playing Loop')
    running = True
    while running:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                print('Received Quit Event:', event)
                running = False
            elif event.type == pygame.MOUSEBUTTONDOWN:
                print('Received Mouse Event', event)
                x, y = event.pos
                pygame.draw.rect(display_surface, BLUE, [x, y,
WIDTH, HEIGHT])

        # Update the display
        pygame.display.update()
        # Defines the frame rate - the number of frames per
second
        # Should be called once per frame (but only once)
        clock.tick(FRAME_REFRESH_RATE)

    print('Game Over')
    # Now tidy up and quit Python
    pygame.quit()

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

Note that we now need to record the display surface in a local variable so that we can use it to draw the blue rectangles. We also need to call the `pygame.display.update()` function each time round the main while loop so that the new rectangles we have drawn as part of the event processing for loop are displayed to the user.

We also set the frame rate each time round the main while loop. This should happen once per frame (but only once) and uses the clock object initialised at the start of the program.

12.7 Alternative Approach to Processing Input Devices

There are actually two ways in which inputs from a device such as a mouse, joystick or the keyboard can be processed. One approach is the *Event* based model described earlier. The other approach is the *State* based approach.

Although the Event based approach has many advantages is has two disadvantages:

- Each event represents a single action and continuous actions are not explicitly represented. Thus if the user presses both the X key and the Z key then this will generate two events and it will be up to the program to determine that they have been pressed at the same time.
- It is also up to the program to determine that the user is still pressing a key (by noting that no `KEYUP` event has occurred).
- Both of these are possible but can be error prone.

An alternative approach is to use the *State* based approach. In the state based approach the program can directly check the state of a input device (such as a key or mouse or keyboard). For example, you can use `pygame.key.get_pressed()` which returns the state of all the keys. This can be used to determine if a specific key is being pressed at this moment in time. For example, `pygame.key.get_pressed()[pygame.K_SPACE]` can be used to check to see if the space bar is being pressed.

This can be used to determine what action to take. If you keep checking that the key is pressed then you can keep performing the associated action. This can be very useful for continues actions in a game such as moving an object etc.

However, if the user presses a key and then releases it before the program checks the state of the keyboard then that input will be missed.

12.8 pygame Modules

There are numerous modules provided as part of pygame as well as associated libraries. Some of the core modules are listed below:

- `pygame.display` This module is used to control the display window or screen. It provides facilities to initialise and shutdown the display module. It can be used to initialise a window or screen. It can also be used to cause a window or screen to refresh etc.

- `pygame.event` This module manages events and the event queue. For example
  `pygame.event.get()` retrieves events from the event queue,
  `pygame.event.poll()` gets a single event from the queue and
  `pygame.event.peek()` tests to see if there are any event types on the queue.

- `pygame.draw` The draw module is used to draw simple shapes onto a `Surface`. For example, it provides functions for drawing a rectangle (`pygame.draw.rect`), a polygon, a circle, an ellipse, a line etc.
- `pygame.font` The font module is used to create and render TrueType fonts into a new Surface object. Most of the features associated with fonts are supported by the `pygame.font.Font` class. Free standing module functions allow the module to be initialised and shutdown, plus functions to access fonts such as `pygame.font.get_fonts()` which provides a list of the currently available fonts.
- `pygame.image` This module allows images to be saved and loaded. Note that images are loaded into a Surface object (there is no Image class unlike many other GUI oriented frameworks).
- `pygame.joystick` The joystick module provides the Joystick object and several supporting functions. These can be used for interacting with joysticks, gamepads and trackballs.
- `pygame.key` This module provides support for working with inputs from the keyboard. This allows the input keys to be obtained and modifier keys (such as Control and Shift) to be identified. It also allows the approach to repeating keys to be specified.
- `pygame.mouse` This module provides facilities for working with mouse input such as obtaining the current mouse position, the state of mouse buttons as well as the image to use for the mouse.
- `pygame.time` This is the pygame module for managing timing within a game. It provides the `pygame.time.Clock` class that can be used to track time.

12.9 Online Resources

There is a great deal of information available on pygame including:

- https://www.pygame.org The pygame home page.
- http://www.libsdl.org/ SDL (Simple Directmedia Layer) documentation.
- news://gmane.comp.python.pygame The official pygame news group.