Python Object-Oriented Program with Libraries

Unit 4: PyGame Tutorial

CHAPTER 1: PYGAME BASICS

DR. ERIC CHOU

IEEE SENIOR MEMBER

GUI and TUI

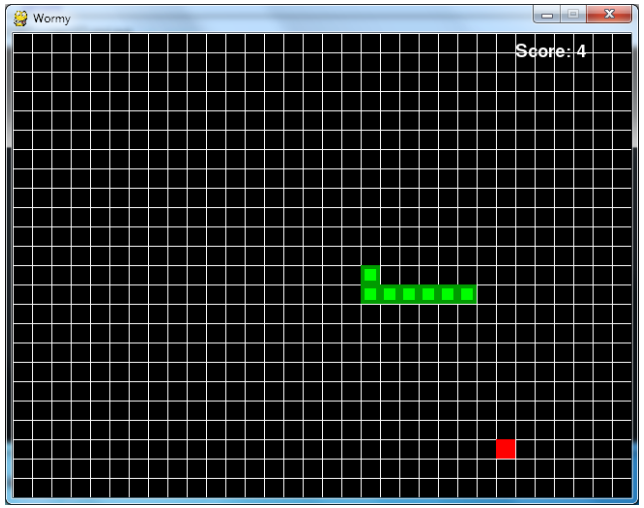
LECTURE 1

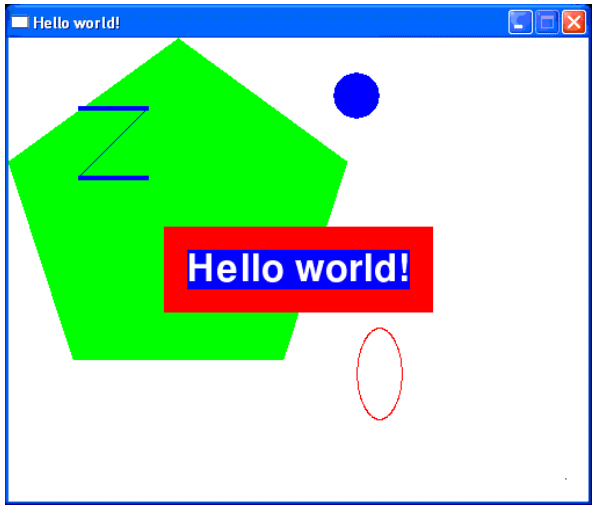
Pygame Basics

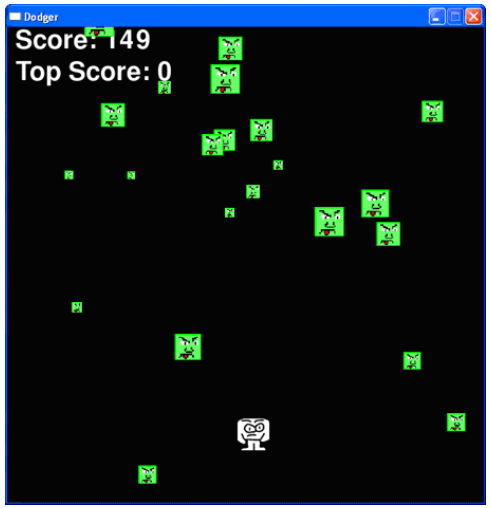
Just like how Python comes with several modules that provide additional functions for your programs (like math and random), the Pygame framework includes several modules with functions for:

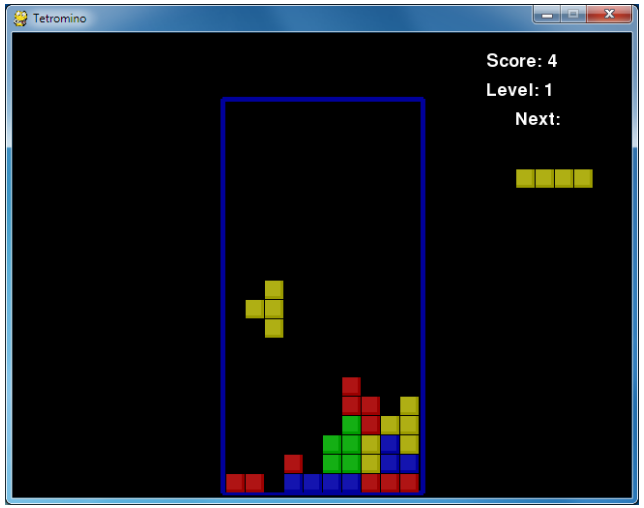
- drawing graphics (Graphics – GUI and Canvas)
- playing sounds (Audio)
- handling mouse input (Event Handling)
- And more!

Some Pygames









GUI vs. TUI

The Python programs that you can write with Python's built-in functions only deal with text through the print() and input() functions.

Your program can display text on the screen and let the user type in text from the keyboard. This type of program has a command line interface, or TUI (Text-mode User Interface).

GUI vs. TUI

These programs are somewhat limited because they can't display graphics, have colors, or use the mouse.

These CLI programs only get input from the keyboard with the input() function and even then user must press Enter before the program can respond to the input.

This means **real-time** (that is, continuing to run code without waiting for the user) action games are impossible to make.

GUI vs. TUI

Pygame provides functions for creating programs with a *graphical user interface*, or **GUI** (pronounced, “gooey”).

Programs with a graphics-based GUIs can show a window with images and colors.

PyGame Hello World

LECTURE 1

Hello World

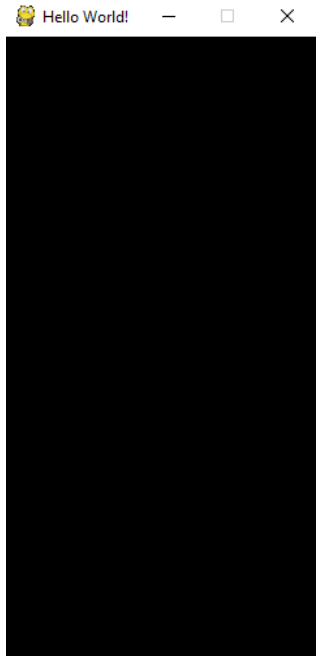
Our first program using Pygame will be a small program that makes a window that says

"**Hello World!**" appear on the screen.

Blank Game

Demo Program: blankgame.py

1. **Create a script called blankpygame.py**
2. **Type in the following program (do not include numbers or the periods at the beginning of the line – they are just for reference.)**



Event Loop started.

Get one Event object from the event Loop.

Event Loop stopped.

```
import pygame
import sys

width, height = 240, 480

def main():
    pygame.init()
    root = pygame.display.set_mode((width, height))
    pygame.display.set_caption('Hello World!')
    while True:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()
        pygame.display.update()
    pygame.quit()

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

Hello World

It’s just a blank window with “Hello World!” at the top of the window (in what is called the window’s **title bar**, which holds the **caption** text).

But creating a window is the first step to making graphical games. When you click on the X button in the corner of the window, the program will end and the window will disappear.

Hello World – A Closer Look

```
1. import pygame, sys
```

The first few lines of code are lines that will begin almost every program you write that uses Pygame.

Line 1 is a simple import statement that imports the pygame and sys modules so that our program can use the functions in them.

All of the Pygame functions dealing with graphics, sound, and other features that Pygame provides are in the pygame module.

Hello World

We can’t use the input() or print() functions in a Pygame GUI. We will learn more about input and output in Pygame later.

Let’s take a closer look to the “Hello World” program!

Hello World – A Closer Look

```
2. from pygame.locals import *
```

Line 2 is also an import statement. It uses the form **modulename** import * format.

Normally if you want to call a function that is in a module, you must use the **modulename.functionname()** format after importing the module.

However, with from **modulename** import *, you can skip the ***modulename.*** portion and simply use functionname() (just like Python's built-in functions).

Hello World – A Closer Look

```
4. pygame.init()
```

Line 4 is the pygame.init() function call, which always needs to be called after importing the pygame module and before calling any other Pygame function.

You don’t need to know what this function does, you just need to know that it needs to be called first in order for many Pygame functions to work.

If you ever see an error message like

pygame.error: font not initialized,

check to see if you forgot to call pygame.init() at the start of your program.

Hello World – A Closer Look

```
8       root = pygame.display.set_mode((width, height))
```

Line 5 is a call to the pygame.display.set_mode() function, which returns the pygame.Surface object for the window. *More on surface objects later…*

Notice that we pass a tuple value of two integers to the function: (240, 480). This tuple tells the set_mode() function how wide and how high to make the window in pixels. (240, 480) will make a window with a width of 240 pixels and height of 480 pixels.

NOTE: An error will occur if it is not a tuple.

Hello World – A Closer Look

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

Line 6 sets the caption text that will appear at the top of the window by calling the `pygame.display.set_caption()` function. The string value `'Hello World!'` is passed in this function call to make that text appear as the caption:

Hello World – A Closer Look - Game States

```
7. while True: # main game loop
8.     for event in pygame.event.get():
```

Most of the games we will be looking at have these *while True* loops in them along with a comment calling it the “main game loop”. A game loop (also called a main loop) is a loop where the code does three things:

1. Handles events.
2. Updates the game state.
3. Draws the game state to the screen.

The Game State

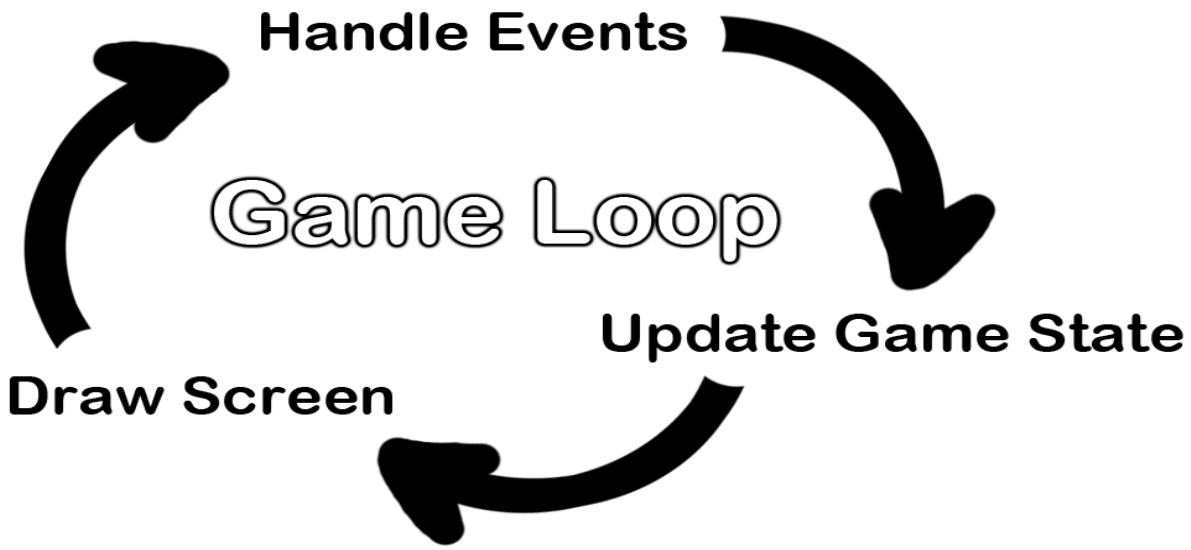
The **game state** is simply a way of referring to a set of values for all the variables in a game program. In many games, the game state includes the values in the variables that tracks the player's health and position, the health and position of any enemies, which marks have been made on a board, the score, or whose turn it is.

Whenever something happens like the player taking damage (which lowers their health value), or an enemy moves somewhere, or something happens in the game world we say that the game state has changed.

The Game State

If you've ever played a game that lets you save, the "save state" is the game state at the point that you've saved it. In most games, pausing the game will prevent the game state from changing.

The Game State



Since the game state is usually updated in response to events (such as mouse clicks or keyboard presses) or the passage of time, the game loop is constantly checking and re-checking many times a second for any new events that have happened. This is usually called event handling.

Hello World – The QUIT Event

```
 9.             if event.type == QUIT:
10.                 pygame.quit()
11.                 sys.exit()
```

Event objects have a member variable (also called attributes or properties – *remember ALICE?*) named type which tells us what kind of event the object represents.

Line 9 checks if the Event object's type is equal to the constant QUIT. Remember that since we used the from pygame.locals import * form of the import statement, we only have to type QUIT instead of pygame.locals.QUIT.

Hello World – A Closer Look

```
12.     pygame.display.update()
```

Line 12 calls the pygame.display.update() function, which draws the Surface object returned by pygame.display.set_mode() to the screen (remember we stored this object in the DISPLAYSURF variable).

Since the Surface object hasn't changed, the same black image is redrawn to the screen each time pygame.display.update() is called.

pygame.display.flip()

Update the full display Surface to the screen

```
flip() -> None
```

This will update the contents of the entire display.

pygame.display.update()

Update portions of the screen for software displays

```
update(rectangle=None) -> None
update(rectangle_list) -> None
```

This function is like an optimized version of pygame.display.flip() for software displays. It allows only a portion of the screen to updated, instead of the entire area. If no argument is passed it updates the entire Surface area like pygame.display.flip().

Hello World – That is IT!

That is the entire program!

After line 12 is done, the infinite while loop starts again from the beginning.

This program does nothing besides make a black window appear on the screen, constantly check for a QUIT event, and then redraws the unchanged black window to the screen over and over again.

What is Next?

Let’s learn how to make interesting things appear on this window instead of just blackness by learning about :

- Pixels
- Surface objects
- Color objects
- Rect objects
- Keyboard Mouse
- Pygame drawing functions.

PyGame Click Ticks

LECTURE 1

Separate the Data Unit and Control Unit

- The main function (also the view of the program) is separated from the game data unit (data_init()).
- In this way, we can deal with the graphic user interface and the data model much more efficiently.

Adding clock ticks

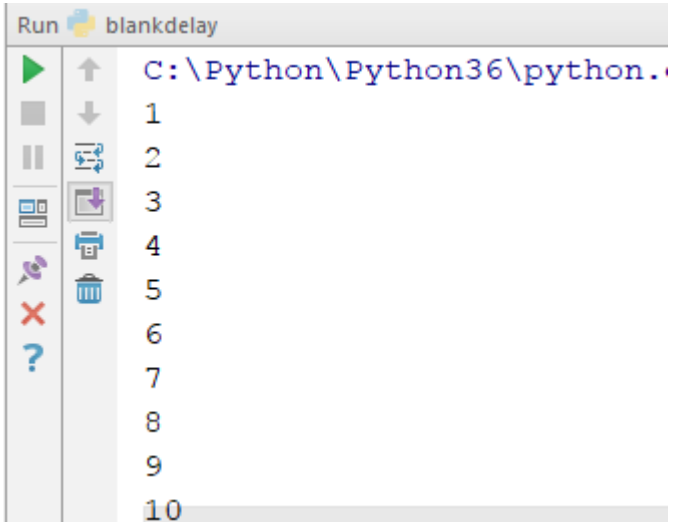
We set the game cycle time to be 300 ticks (milli-sec) by setting cycle_time = 300.

For every cycle_time, we redraw the game board once. We will use print out to show how the program update every one cycle time.

Cycle Time

Demo Program: blankdelay.py

1. Add cycle_time = 300
2. update pygame.display.update() every one cycle time.
3. When updated, we advance a global counter by 1.



Constants and Date Model

```
width, height = 240, 480
cycle_time = 300

# game data
count = 0

def data_init():
    global count
    count = 0
```

Controller and View Model

```
def main():
    global width, height, cycle_time, count
    # initialize game board and other data
    data_init()

    pygame.init()
    root = pygame.display.set_mode((width, height))
    pygame.display.set_caption('Hello World!')
    while True:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()
        pygame.display.update()
        pygame.time.delay(cycle_time)
        count = count + 1
        print(count)
```

Delay for one Cycle (300 ticks)

pygame.time.get_ticks()

get the time in milliseconds

get_ticks() -> milliseconds

Return the number of milliseconds since pygame.init() was called. Before pygame is initialized this will always be 0.

pygame.time.wait()

pause the program for an amount of time

wait(milliseconds) -> time

Will pause for a given number of milliseconds.

pygame.time.delay()

pause the program for an amount of time

delay(milliseconds) -> time

Will pause for a given number of milliseconds.

pygame.time.Clock

create an object to help track time

Clock() -> Clock

pygame.time.Clock.tick	—	update the clock
pygame.time.Clock.tick_busy_loop	—	update the clock
pygame.time.Clock.get_time	—	time used in the previous tick
pygame.time.Clock.get_rawtime	—	actual time used in the previous tick
pygame.time.Clock.get_fps	—	compute the clock framerate

Creates a new Clock object that can be used to track an amount of time. The clock also provides several functions to help control a game's framerate.

Note: This clock object works like a timer.

PyGame Adding Keyboard Events

LECTURE 1

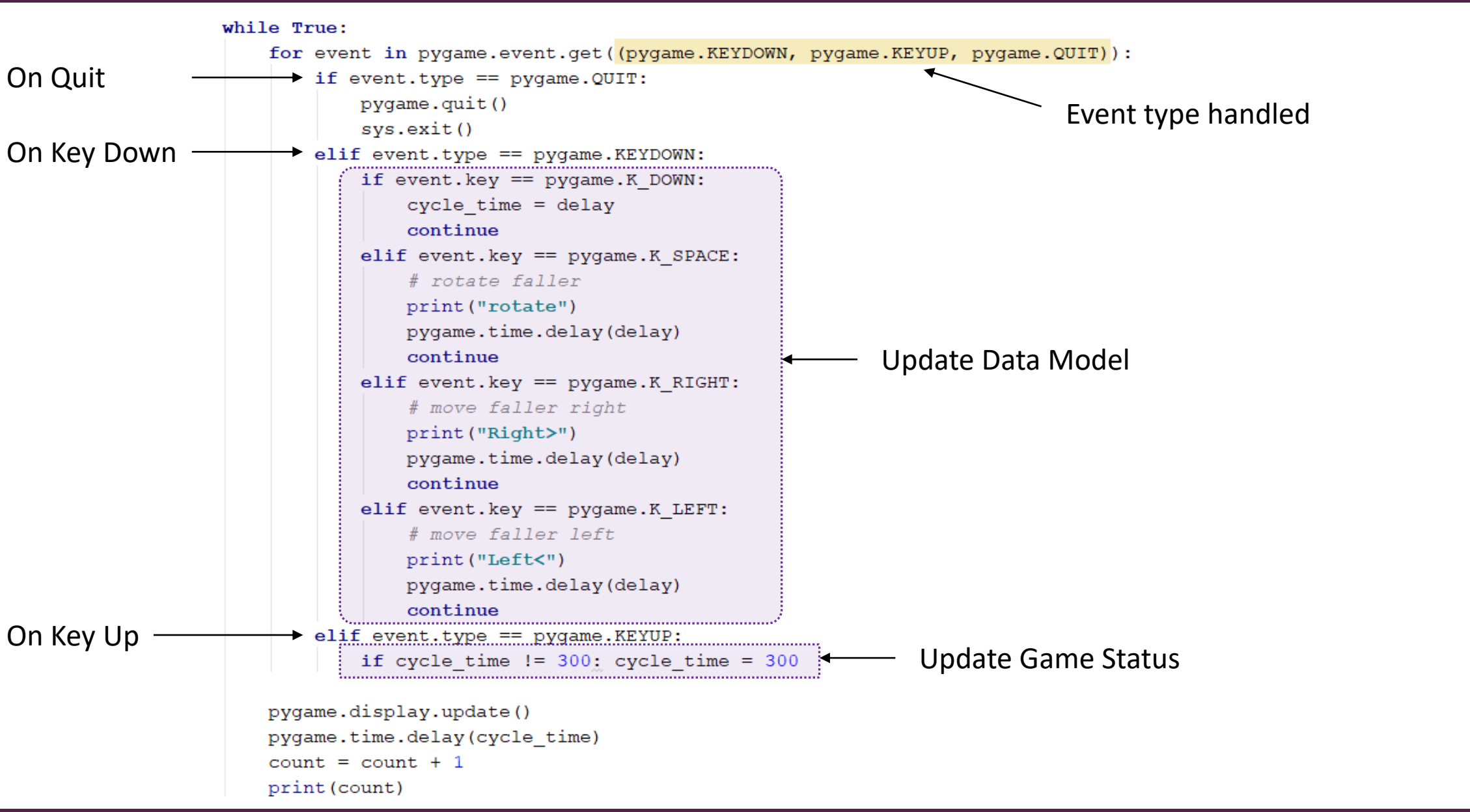
Event Type	Event Attributes
QUIT	none
ACTIVEEVENT	gain, state
KEYDOWN	unicode, key, mod
KEYUP	key, mod
MOUSEMOTION	pos, rel, buttons
MOUSEBUTTONUP	pos, button
MOUSEBUTTONDOWN	pos, button
JOYAXISMOTION	joy, axis, value
JOYBALLMOTION	joy, ball, rel
JOYHATMOTION	joy, hat, value
JOYBUTTONUP	joy, button
JOYBUTTONDOWN	joy, button
VIDEORESIZE	size, w, h
VIDEOEXPOSE	none
USEREVENT	code

Event Methods

pygame.event.pump	— internally process pygame event handlers
pygame.event.get	— get events from the queue
pygame.event.poll	— get a single event from the queue
pygame.event.wait	— wait for a single event from the queue
pygame.event.peek	— test if event types are waiting on the queue
pygame.event.clear	— remove all events from the queue
pygame.event.event_name	— get the string name from and event id
pygame.event.set_blocked	— control which events are allowed on the queue
pygame.event.set_allowed	— control which events are allowed on the queue
pygame.event.get_blocked	— test if a type of event is blocked from the queue
pygame.event.set_grab	— control the sharing of input devices with other applications
pygame.event.get_grab	— test if the program is sharing input devices
pygame.event.post	— place a new event on the queue
pygame.event.Event	— create a new event object

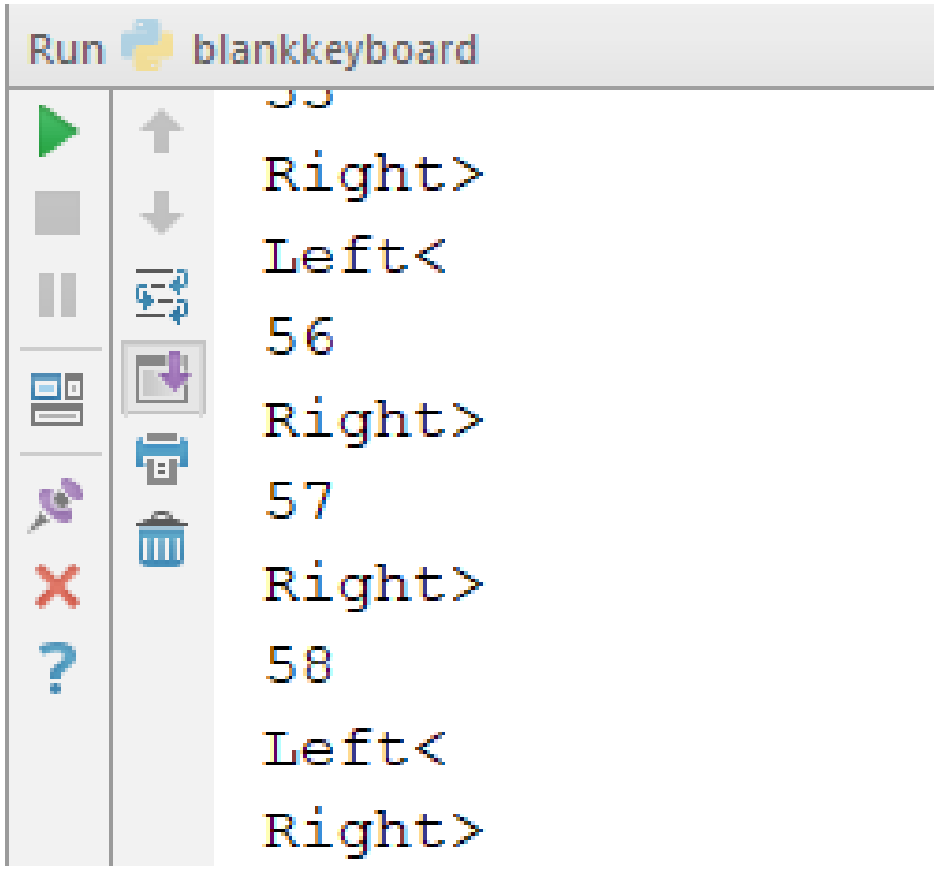
Key Down and Key Up

Quit	On Quit, the game program should be terminated.
Key down	On key down, we should accept the input by the user.
Key up	On key up, we should activate the update of data model and game board status.


On Quit
On Key Down
On Key Up
Event type handled
Update Data Model
Update Game Status
while True:
for event in pygame.event.get((pygame.KEYDOWN, pygame.KEYUP, pygame.QUIT)):
if event.type == pygame.QUIT:
pygame.quit()
sys.exit()
elif event.type == pygame.KEYDOWN:
if event.key == pygame.K_DOWN:
cycle_time = delay
continue
elif event.key == pygame.K_SPACE:
# rotate faller
print("rotate")
pygame.time.delay(delay)
continue
elif event.key == pygame.K_RIGHT:
# move faller right
print("Right>")
pygame.time.delay(delay)
continue
elif event.key == pygame.K_LEFT:
# move faller left
print("Left<")
pygame.time.delay(delay)
continue
elif event.type == pygame.KEYUP:
if cycle_time != 300: cycle_time = 300
pygame.display.update()
pygame.time.delay(cycle_time)
count = count + 1
print(count)

Capture Keyboard Events

Demo Program: blankkeyboard.py



PyGame Color

LECTURE 1

```python
import pygame
import sys

width, height = 240, 480

def main():
    pygame.init()
    root = pygame.display.set_mode((width, height))
    pygame.display.set_caption('Hello World!')
    root.fill(pygame.Color(64, 64, 64))                    # by Color(R, G, B)
    pygame.draw.rect(root, (255, 0, 0), (50, 50, 50, 50))  # by Color tuple
    pygame.display.update()
    while True:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()
        pygame.display.update()
    pygame.quit()

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

PyGame Rectangles

LECTURE 1

PyGame Color

- It is represented using the RGB (**Red**, **Green**, **Blue**) color model.
- In Python, we use tuples of 3 integers to represent the values of **red**, **green**, **and blue** each range from 0 to 255.
- In the following example, we just try to demonstrate how to draw simple rectangles, so we don't use Color Object from PyGame

13 Simple colors in a color dictionary

```
color = {'BLACK': (0, 0, 0),
         'WHITE': (255, 255, 255),
         'RED':   (255, 0, 0),
         'ORANGE':(255, 128, 0),
         'YELLOW':(255, 255, 0),
         'APPLE':(128, 255, 0),
         'GREEN':(0, 255, 0),
         'LAKE':(0, 255, 128),
         'CYAN':(0, 255, 255),
         'TEAL':(0, 128, 255),
         'PURPLE':(128, 0, 255),
         'MAGENTA':(255, 0, 255),
         'DEEPPINK':(255, 0, 128)
         }
```

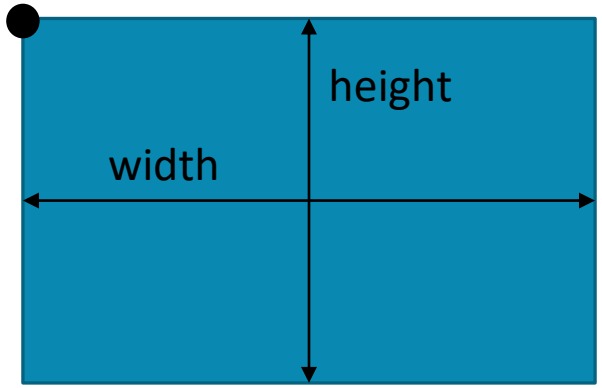
Draw Rectangle on a Surface

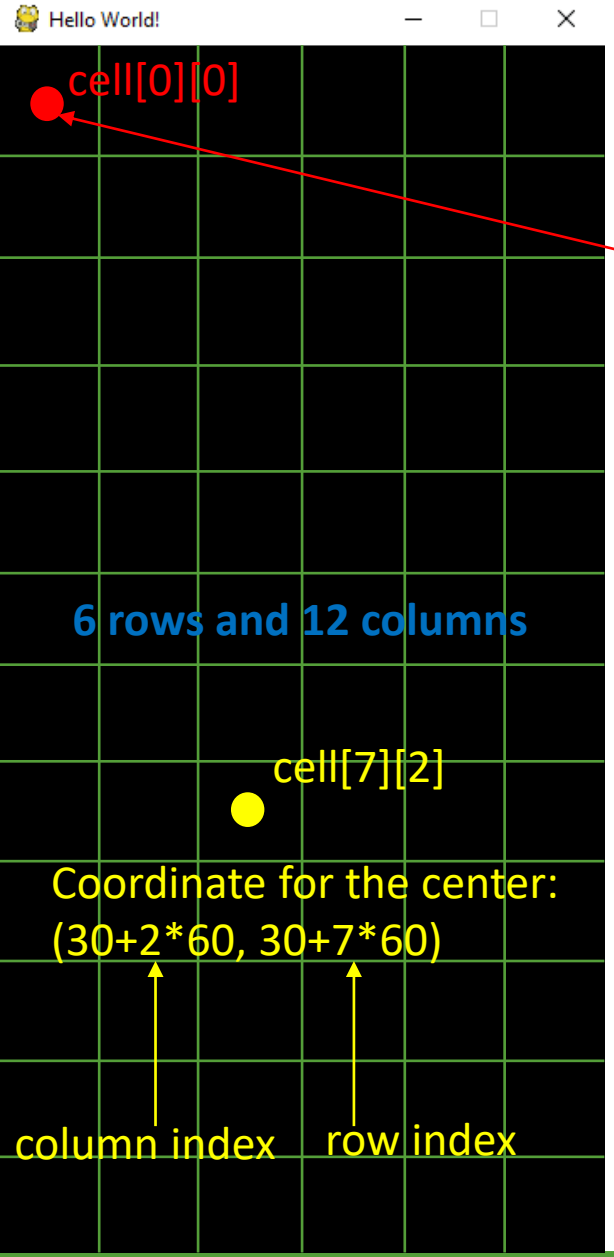
A 3 tuple data for a color

```
pygame.draw.rect(surface, color, (left, top, width, height))
```

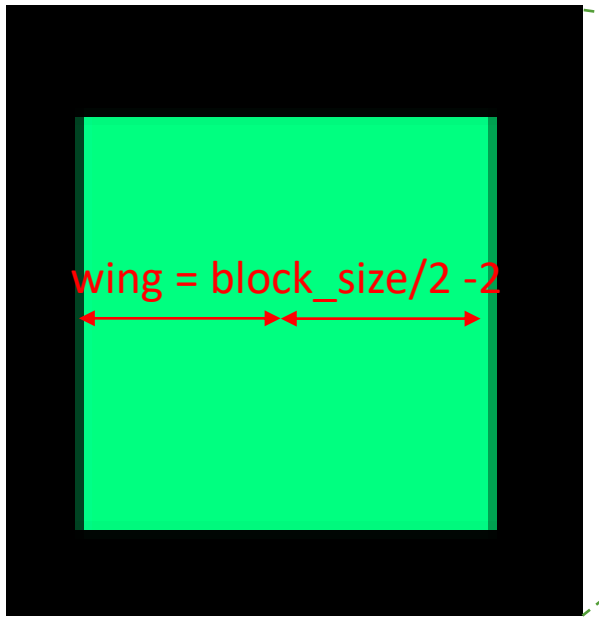
Surface is like Canvas in other tools.
You may draw your rectangle on it. It is the graphics data holder.

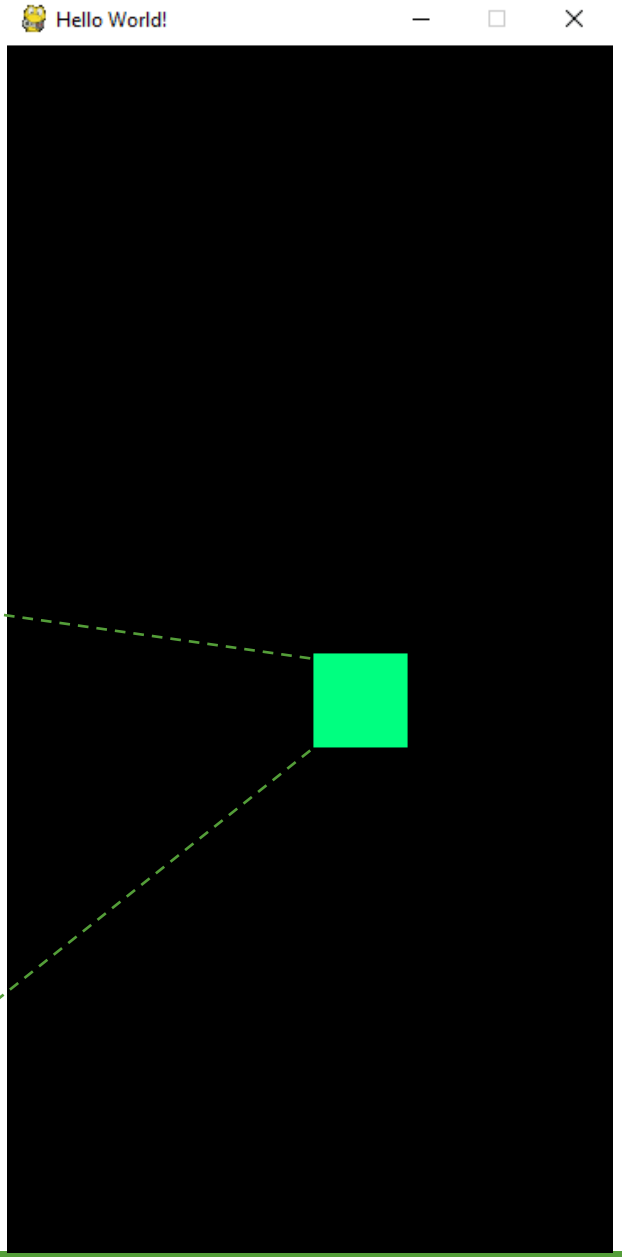
(x, y) = (left, top) Coordinates for the dot





Zero point (center point for
zero[0] = block_size/2
= 30 (in example program)
zero[1] = block_size/2
= 30 (in the example program)





draw_clear() function

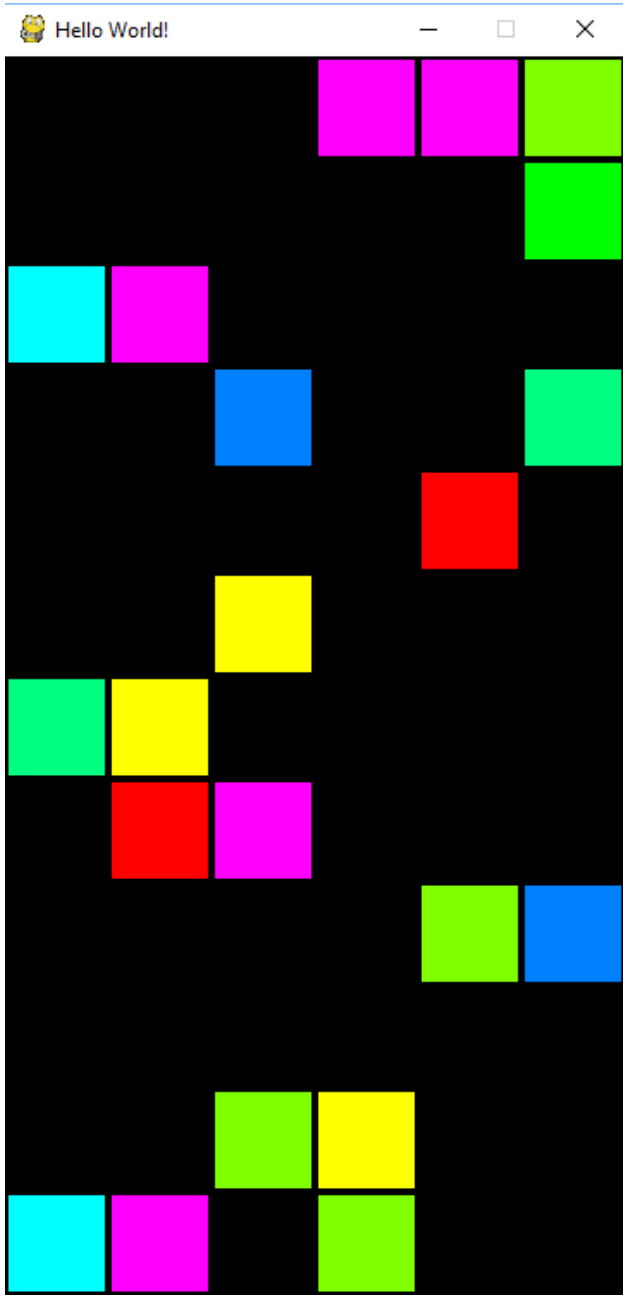
- To paint a black color block on a certain cell. It will work like removing the block.

Random Block Generator

Demo Problem: blankblock.py

1. When space key is hit, a block of random color (color_code[0] to color_code[9]) will be generated.

2. When p key is hit, all blocks will be cleared.


Hello World!

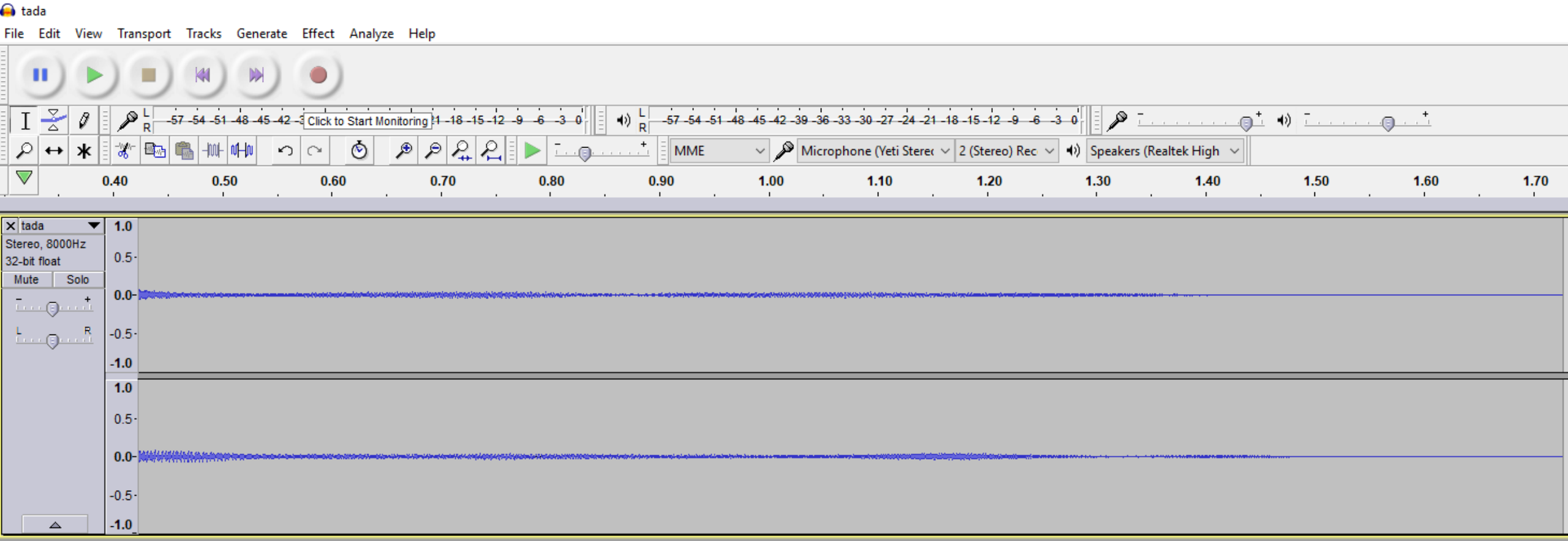
Simple Sound Effects

LECTURE 1

Adding Sound Effect to Your Code

```
# Loading and playing background music:
pygame.mixer.music.load(backgroundmusic.mp3')
pygame.mixer.music.play(-1, 0.0)
# ...some more of your code goes here...
pygame.mixer.music.stop()
```

Audio Effect tada.mp3 (1700+ ticks)



pygame.mixer.music.play()

- Start the playback of the music stream
- play(loops=0, start=0.0) -> None
- This will play the loaded music stream. If the music is already playing it will be restarted.
- The loops argument controls the number of repeats a music will play. play(5) will cause the music to played once, then repeated five times, for a total of six. If the loops is -1 then the music will repeat indefinitely.
- The starting position argument controls where in the music the song starts playing. The starting position is dependent on the format of music playing. MP3 and OGG use the position as time (in seconds). MOD music it is the pattern order number. Passing a startpos will raise a NotImplementedError if it cannot set the start position

```python
import pygame
import sys

width, height = 240, 480

def main():
    pygame.init()
    root = pygame.display.set_mode((width, height))
    pygame.display.set_caption('Hello World!')
    pygame.mixer.music.load('tada.mp3')
    pygame.mixer.music.play(-1, 0.0)
    pygame.time.delay(1500)
    pygame.mixer.music.stop()
    while True:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()
        pygame.display.update()

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

Load Sound Effect tada.mp3

Looping the Sound Effect

For 1.5 sec

Demo Program: blanksound.py

Go PyCharm!!!

Simple Title Page

LECTURE 1

Display Text Message

1. Choose Font

   myfont = pygame.font.SysFont(name, size, bold=False, italic=False)

2. Render Label

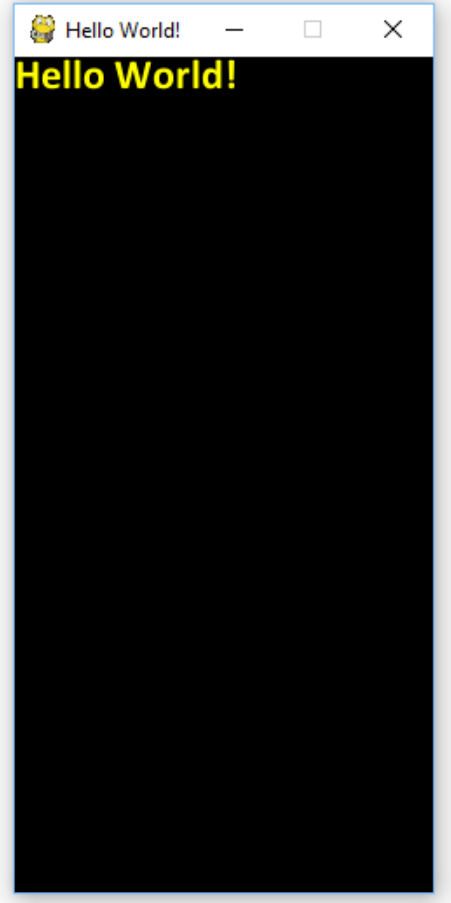
   labelobj = myfont.render(“Your Text”, 1, color)

   Note: 1 is for antialiasing

3. Attach Label on Surface (your canvas in PyGame)
   - surface.blit(labelobj, left, top)

Demo Program: blanktext.py

```
1   import pygame
2   import sys
3
4   width, height = 240, 480
5
6   def main():
7       pygame.init()
8       root = pygame.display.set_mode((width, height))
9       pygame.display.set_caption('Hello World!')
10
11      myfont = pygame.font.SysFont("Calibri", 24, True, False)
12      label = myfont.render("Hello World!", 1, (255, 255, 0))
13      root.blit(label, (0, 0))
14      while True:
15          for event in pygame.event.get():
16              if event.type == pygame.QUIT:
17                  pygame.quit()
18                  sys.exit()
19          pygame.display.update()
20
21  if __name__ == "__main__":
22      main()
```



blankflashingtext.py

```python
import pygame
import sys

width, height = 240, 480

def main():
    pygame.init()
    root = pygame.display.set_mode((width, height))
    pygame.display.set_caption('Hello World!')

    texton = True
    myfont = pygame.font.SysFont("Calibri", 24, True, False)
    for i in range(20):
        if (texton):
            label = myfont.render("Hello World!", 1, (255, 255, 0))
            root.blit(label, (50, 100))
            texton = False
        else:
            pygame.draw.rect(root, (0, 0, 0), ((0, 0, 240, 480)))
            texton = True
        pygame.display.update()
        pygame.time.delay(200)

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

Toggle Flag for Text Flashing

Label

Clear Label

Toggle Cycle Time for Text Flashing

pygame_basics.py

LECTURE 1

Demo Program: pygame_basics.py

Go PyCharm!!!