# Pygame tutorial Documentation

***Release 2019***

**Raphael Holzer**

**Jul 02, 2019**

# Contents:

This tutorial explains how to make interactive games and applications with Pygame. It teaches an **object-oriented programming** approach.

CHAPTER 1

# Making Apps with Pygame

In this tutorial we are going to create applications and games with Pygame. Pygame only allows one single window. Within one window, we can switch Scenes. Each scene has objects.

## 1.1 The App class

We are going to build an app based on pygame. So the first thing to do is to import the module, as well as a series of useful constants:

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

Then we create define the App class which initializes pygame and opens a the app window:

```
class App:
    """Create a single-window app with multiple scenes."""

    def __init__(self):
        """Initialize pygame and the application."""
        pygame.init()
        flags = RESIZABLE
        App.screen = pygame.display.set_mode((640, 240), flags)

        App.running = True
```

Further we have to define the main event loop:

```
def run(self):
    """Run the main event loop."""
    while App.running:
        for event in pygame.event.get():
            if event.type == QUIT:
```

(continues on next page)

(continued from previous page)

```
                App.running = False
    pygame.quit()
```

At the end of the module we run a demo, if the programm is run directly and not imported as a module:

```
if __name__ == '__main__':
    App().run()
```

## 1.2 Add text

Now we add some text to the screen. We create a Text class from which we can instantiate text objects:

```
class Text:
    """Create a text object which knows how to draw itself."""

    def __init__(self, text, pos, **options):
        """Instantiate and render the text object."""
        self.str = text
        self.pos = pos
        self.fontsize = 72
        self.fontcolor = Color('black')
        self.render()
```

The text needs to be rendered into a surface object, an image. This needs to be done only once, or whenever the text changes:

```
def render(self):
    """Render the string and create a surface object."""
    self.font = pygame.font.Font(None, self.fontsize)
    self.text = self.font.render(self.str, True, self.fontcolor)
    self.rect = self.text.get_rect()
```

Drawing the text means blitting it to the application screen:

```
def draw(self):
    """Draw the text surface on the screen."""
    App.screen.blit(self.text, self.pos)
```

## 1.3 Create scenes

Most applications or games have different scenes, such as an introduction screen, an intro, and different game levels. So we are going to define the Scene class:

```
class Scene:
    """Create a new scene (room, level, view)."""
    id = 0
    bg = Color('gray')
```

When creating a new scene, we append the scene to the applications scene list and make this scene the current scene:

```
def __init__(self, *args, **kwargs):
    # Append the new scene and make it the current scene
    App.scenes.append(self)
    App.scene = self
```

The we set a scene id, which is kept as class attribute of the Scene class. Then we set the nodes list to the empty list and set the background color:

```
# Set the instance id and increment the class id
self.id = Scene.id
Scene.id += 1
self.nodes = []
self.bg = Scene.bg
```

The scene object knows how to draw itself. It first fills the background with the background color, then draws each nodes and finally flips the display to update the screen:

```
def draw(self):
    """Draw all objects in the scene."""
    App.screen.fill(self.bg)
    for node in self.nodes:
        node.draw()
    pygame.display.flip()
```

The string representation of the scene is *Scene* followed by its ID number:

```
def __str__(self):
    return 'Scene {}'.format(self.id)
```


## 1.4 Shortcut keys

Key presses can be used to switch scenes, or to interact with the game, or to run commands. We add the following code inside the event loop to intercept the S key:

```
if event.type == KEYDOWN:
    if event.key == K_s:
        print('Key press S')
```

The easiest way to represent shortcuts is under the form of a dictionary, where the keys are associated with command strings. We add the following code inside the App init method:

```
self.shortcuts = {K_ESCAPE: 'App.running=False',
                  K_p: 'self.capture()',
                  K_w: 'self.where()',
                  K_s: 'self.next_scene()',
                  }
```

Inside the event loop we detect keydown events and call the key handler:

```
if event.type == KEYDOWN:
    self.do_shortcuts(event)
```

The following method handles the shortcuts for simple keys or combinations of keys and modifier keys:

```
def do_shortcuts(self, event):
    """Check if the key/mod combination is part of the shortcuts
    dictionary and execute it. More shortcuts can be added
    to the ``self.shortcuts`` dictionary by the program."""
    k = event.key
    m = event.mod

    if k in self.shortcuts and m == 0 :
        exec(self.shortcuts[k])
    elif (k, m) in self.shortcuts:
        exec(self.shortcuts[k, m])
```

CHAPTER 2

# Introduction to Pygame

Pygame is a multimedia library which a wrapper around the SDL (Simple DirectMedia Layer) library.

After an introduction of the basics without classes, each example is a subclass of the `Game` class.

## 2.1 Initialization

In order to access the classes and methodes defined in the pygame package, the module must first be imported:

```
import pygame
```

The import statement writes the version and the following texte to the console:

```
pygame 1.9.5
Hello from the pygame community. https://www.pygame.org/contribute.html
```

The pygame import statement is always placed at the beginning of the program. The effect is to import the classes and methods, which can be accessed via `pygame.method()`.

Then we initialize all submodules with the following line:

```
pygame.init()
```

Finally we set the screen size and assign the `Surface` object to the variable `screen`:

```
screen = pygame.display.set_mode((640, 240))
```

Running this program opens a window and closes it immediately.

## 2.2 The event loop

One of the essential parts of any game or user application is the event loop. Events are the things that can happen, such as a mouse click, a mouse movement, or a keyboard press. The following is an infinite loop which prints all events to the console:

```
while True:
    for event in pygame.event.get():
        print(event)
```

On the console you find something like this:

```
<Event(4-MouseMotion {'pos': (173, 192), 'rel': (173, 192), 'buttons': (0, 0, 0),
↪'window': None})>
<Event(2-KeyDown {'unicode': 'a', 'key': 97, 'mod': 0, 'scancode': 0, 'window': None}
↪)>
<Event(3-KeyUp {'key': 97, 'mod': 0, 'scancode': 0, 'window': None})>
<Event(12-Quit {})>
```

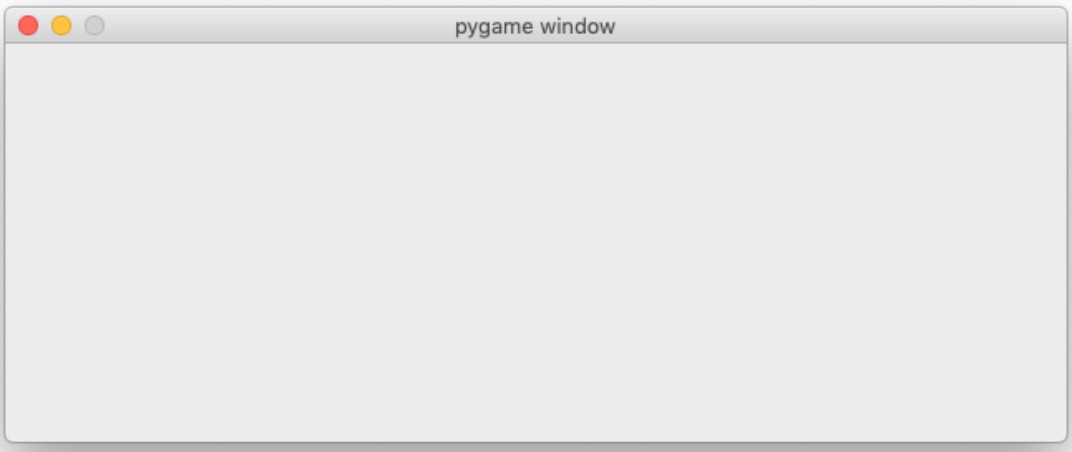
In order to quite the program, make the console the active window and type `ctrl-C`.

## 2.3 Quitting the event loop

In order to quit the game with the window close button (QUIT event) we modify the event loop:

```
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

pygame.quit()
```

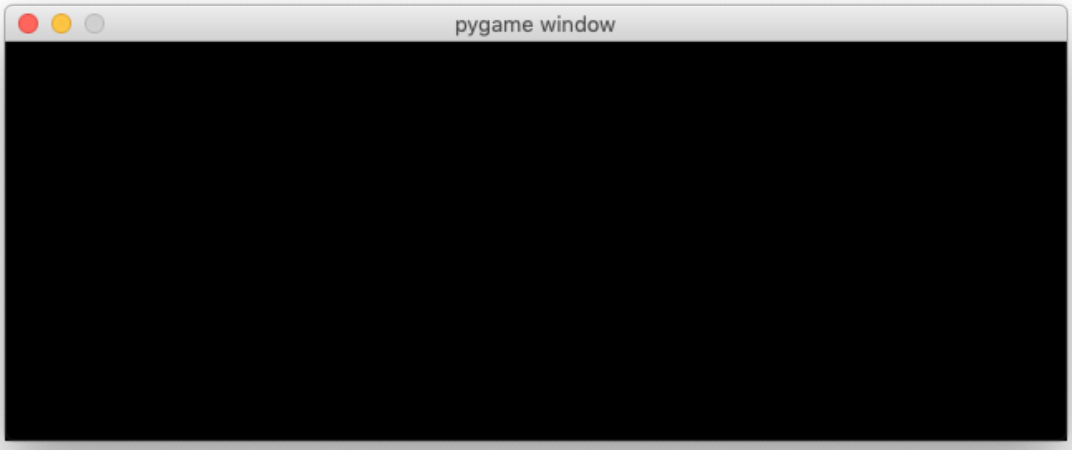


## 2.4 Object oriented Programming

From now on we will use **object-oriented programming (OOP)** style. We define a `Game` class which is main game object. It has a `run()` method which launches the event loop.

**class** intro3.**Game**
Is the main game object.

**run**()
Runs the main event loop.




## 2.5 Changing background color

Colors are defined as tuples of the three components red, green and blue. Each component is represented as an integer value from 0 to 255. Here are some color definitions:

```
BLACK = (0, 0, 0)
RED = (255, 0, 0)
GREEN = (0, 255, 0)
BLUE = (0, 0, 255)
```

In the event loop we decode the key down event:

```
if event.type == pygame.KEYDOWN:
```

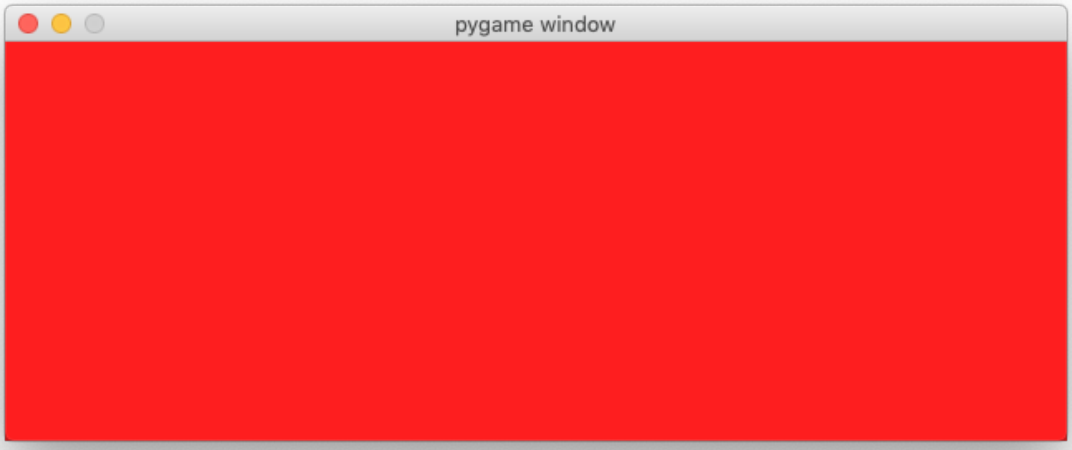
and within the all possible key down events we use the R, G, and B key to set the color:

```
if event.key == pygame.K_r:
    self.color = RED
elif event.key == pygame.K_g:
    self.color = GREEN
elif event.key == pygame.K_b:
    self.color = BLUE
```

This program changes the background color between red, green and blue

**class** intro4.**Game**
Is the main game object.

**run**()
Runs the main event loop.




## 2.6 Display text

In order to display text we need to create a `Font` object which defines a specific font and a specific size:

```
self.font = pygame.font.Font(None, 48)
```

The font object has `render()` method which creates a `Surface` object which is like an image:

```
s = 'Hello world.'
self.text = self.font.render(s, False, RED)
```

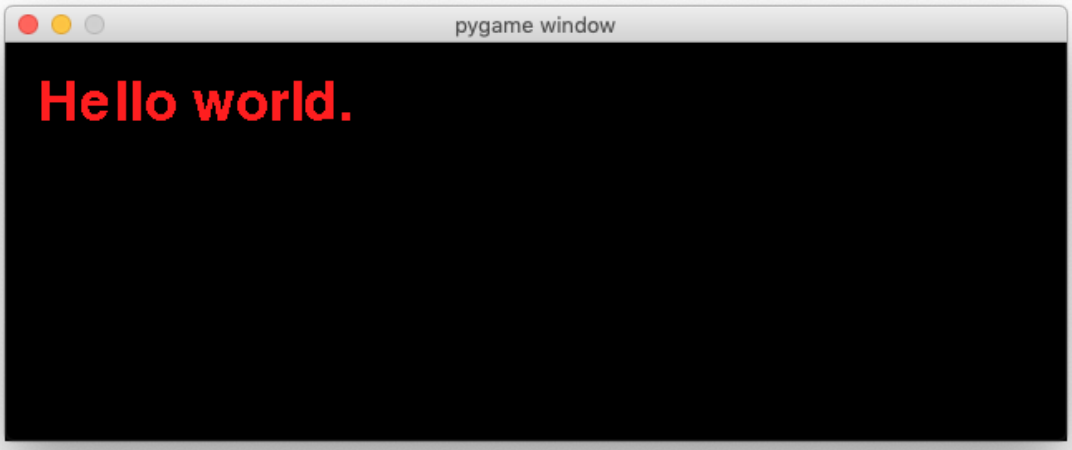
This `Surface` object can be placed on the screen like any image:

```
self.screen.blit(self.text, (20, 20))
```

Display a red text on the screen.

**class** intro5.**Game**
Define the main game object and its attributes.

**run**()
Run the main event loop.




## 2.7 Composing an RGB color

In the following program we use the R, G, and B key to compose the RGB value of the background color.

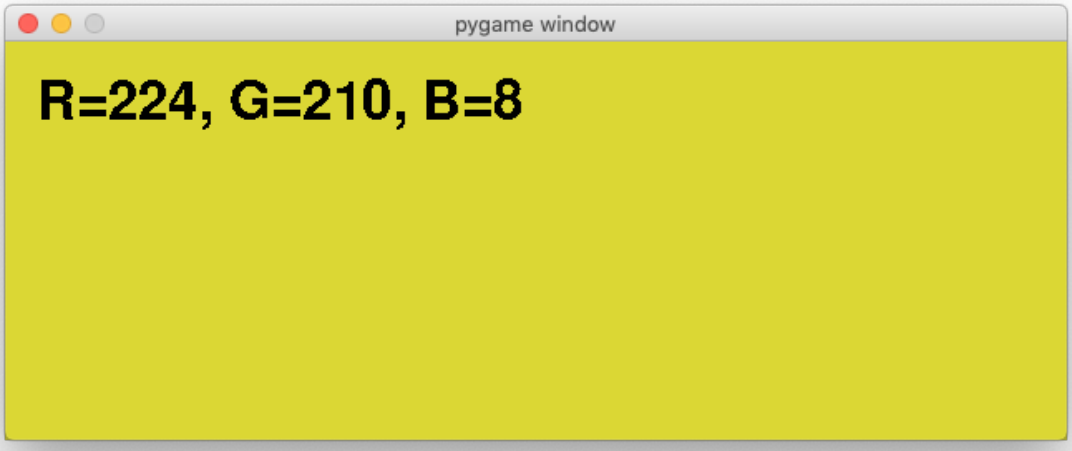
Choose the RGB components of the background color by pressing the R, G, and B keys .

**class** intro6.**Game**

Define the main game object and its attributes.

**run**()

Run the main event loop.

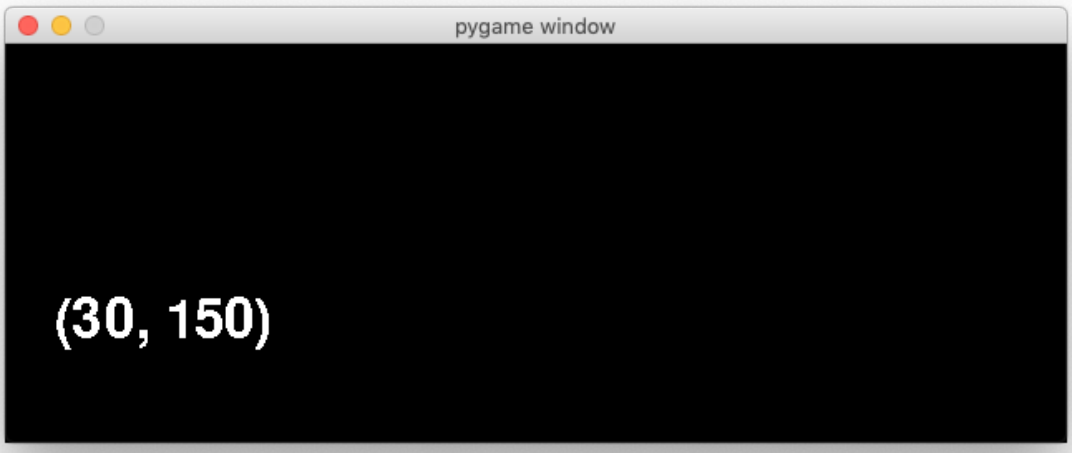


## 2.8 Display the mouse position

Display the mouse position as a label and move the label position with the mouse.

**class** intro7.**Game**
Define the main game object and its attributes.

**run**()
Run the main event loop.




## 2.9 Demo

To show what Pygame can do, here is a simple program that does a bouncing ball animation:

```
import sys, pygame
pygame.init()

size = width, height = 640, 480
speed = [2, 2]
black = 0, 0, 0

screen = pygame.display.set_mode(size)

ball = pygame.image.load("ball.gif")
ballrect = ball.get_rect()

while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            sys.exit()

    ballrect = ballrect.move(speed)
    if ballrect.left < 0 or ballrect.right > width:
```

(continues on next page)

(continued from previous page)

```
        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()
```

CHAPTER 3

# Drawing primitives

The `draw` module allows to draw simple shapes to a surface:

- rectangle
- polygon
- circle
- ellipse
- arc
- line
- lines

The functions have the following format:

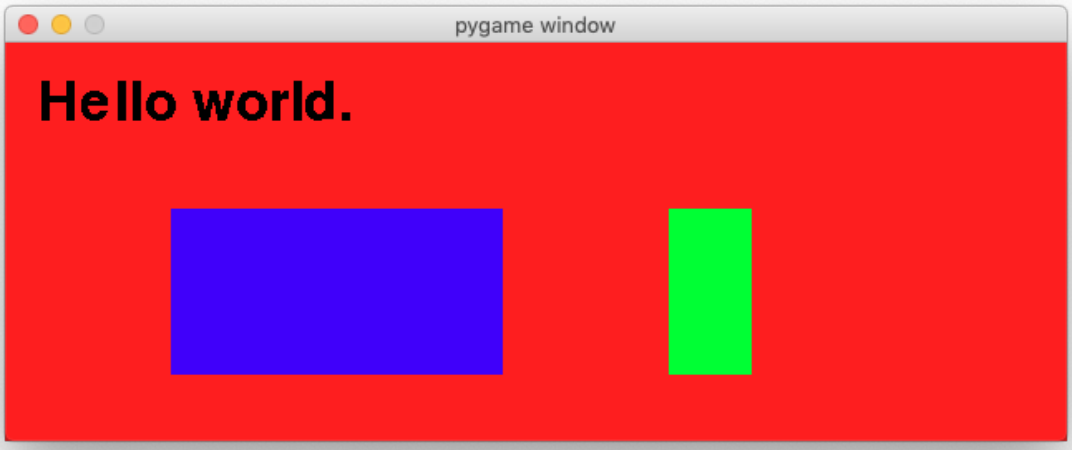
```
rect(Surface, color, Rect, width) -> Rect
polygon(Surface, color, pointlist, width) -> Rect
circle(Surface, color, center, radius, width) -> Rect
```

Most of the functions take a width argument. If the width is 0, the shape is filled.

Draw text, rectangles and a circle.

**class** draw1.**App**
Define the main game object and its attributes.

**run**()
Run the main event loop.




## 3.1 Draw a rectangle

In the next example we use the arrow keys to move the rectangle and alt+arrow to resize the rectangle. We define a dictionary which associates the 4 arrow keys with a displacement vector:

```
d = {K_UP:(0, -10), K_DOWN:(0, 10), K_LEFT:(-10, 0), K_RIGHT:(10, 0)}
```

The code for the movement becomes now very simple:

```
if event.type == pygame.KEYDOWN:
    if event.key in d:
        vec = d[event.key]
        if event.mod & KMOD_ALT:
            self.rect.inflate_ip(vec)
        else:
            self.rect.move_ip(vec)
```

We define also a color dictionary which associations a character key with a color:

```
color = {K_r:RED, K_b:BLUE, K_g:GREEN, K_m:MAGENTA,
            K_c:CYAN, K_y:YELLOW, K_k:BLACK, K_w:WHITE}
```

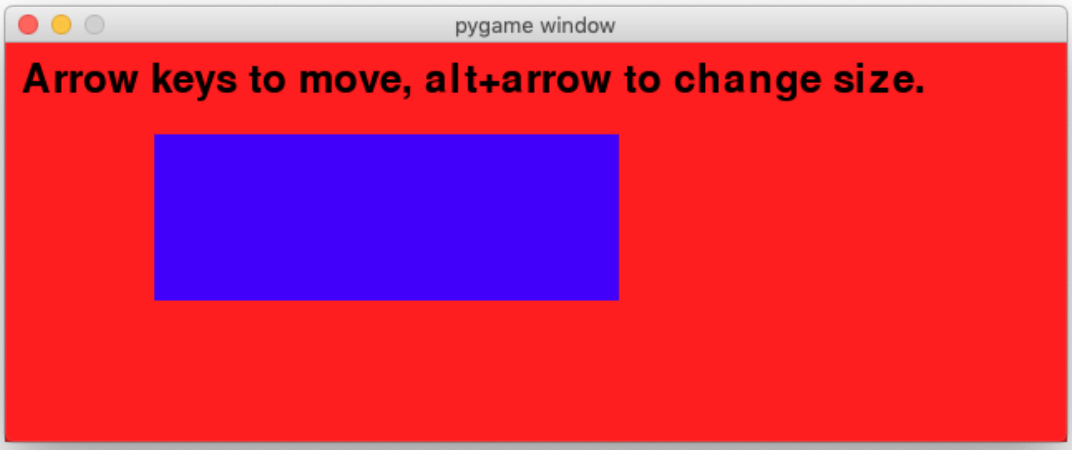
Again, when using dictionaries, the code becomes very short and simple:

```
if event.key in color:
    if event.mod & KMOD_ALT:
        self.bg = color[event.key]
    else:
        self.col = color[event.key]
```

Draw a blue rectangle. Use the arrow keys to move the rectangle. Use the alt+arrow keys to resize the rectangle.

**class** draw2.**App**
Define the main game object and its attributes.

**run**()
Run the main event loop.



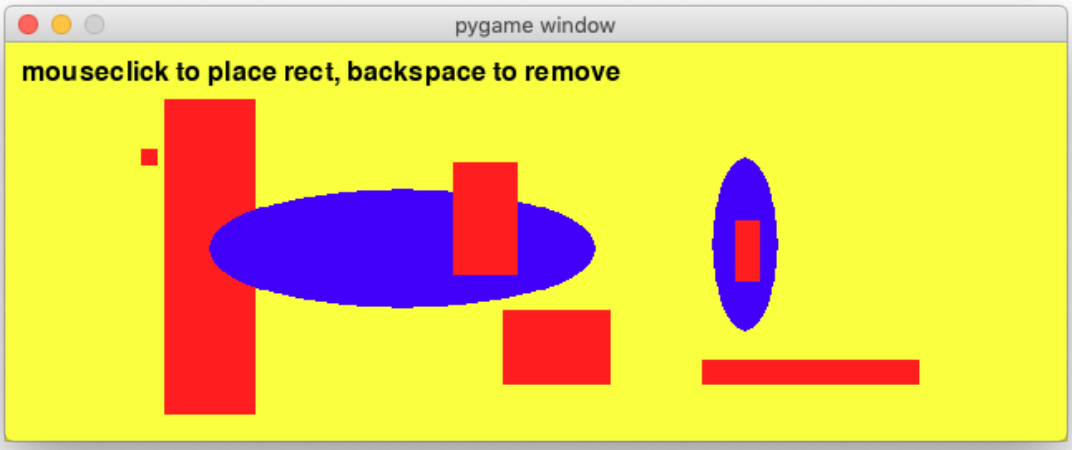

## 3.2 Place a rectangle with the mouse

In the next program we use the mouse button to draw a rectangle.

Place a rectangle or ellipse by clicking and dragging with the mouse. Press E to draw an ellipse and R to draw a rectangle.

**class** draw3.**App**
Define the main game object and its attributes.

**run**()
Run the main event loop.

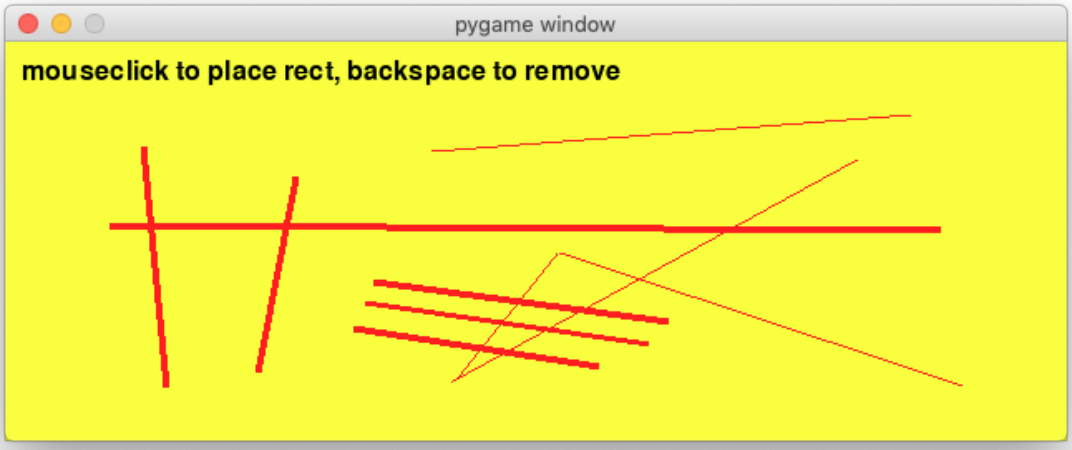



## 3.3 Draw lines

Place lines on the screen. * Text * TextLabel to present a list menu

**class** draw4.**App**
Define the main game object and its attributes.

**run**()
Run the main event loop.



## 3.4 The App class

The basic structure of a game is always the same. We create a `App` class from which we can sub-class our applications.

The constructor method

- initilizes the module
- creates a display window, stored as class variable `App.screen`
- defines a background color
- defines an empty `objects` list:

```
class App():
    """Define the main game object and its attributes."""
    def __init__(self):
        pygame.init()
        App.screen = pygame.display.set_mode((640, 240))
        self.bg_color = WHITE
        self.objects = []
```

The `run()` method enters the game loop. Only the QUIT event is handled. All other events are sent to the `on_event` function:

```
def run(self):
    """Run the main event loop.
    Handle the QUIT event and call ``on_event``. """
    running = True
    while running:
        for event in pygame.event.get():

            if event.type == QUIT:
                running = False
            else:
                self.on_event(event)
        self.draw()
```

If a game has events and interacts with the user then the `on_event`` method must be implemented:

```
def on_event(self, event):
    """Implement an event handler."""
    pass
```

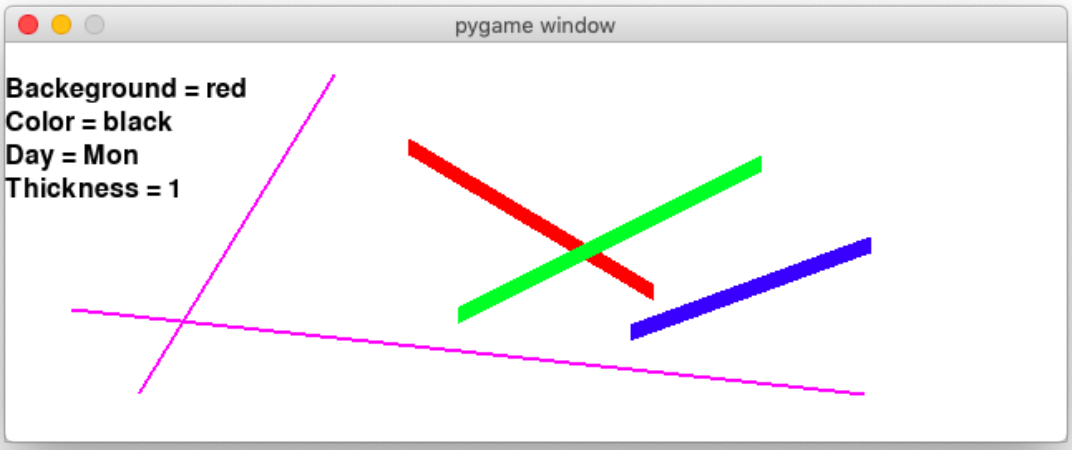
The `draw()` method

- draws the background
- draws all the objects in the `objects` list
- updates (flips) the display:

```
def draw(self):
    """Draw the game objects to the screen."""
    self.screen.fill(self.bg_color)
    for object in self.objects:
        object.draw()
    pygame.display.flip()
```

**class** draw5.**LineDemo**
Drawing lines with the mouse.

**on_event** (*event*)
React to mouseclicks and keydown events.




## 3.5 Text demo

The basic structure of a game is always the same. We create a `App` class from which we can sub-class.

Display example text on the screen. Vary color, size and position. Add new text objects with a mouse click. Type to add text.

**class** draw6.**Text**(*str=''*, *size=None*, *color=None*, *bgcolor=None*, *font=None*, ***kwargs*)
Draw a line of text on the screen.

**draw**()
Draw the text on the screen.

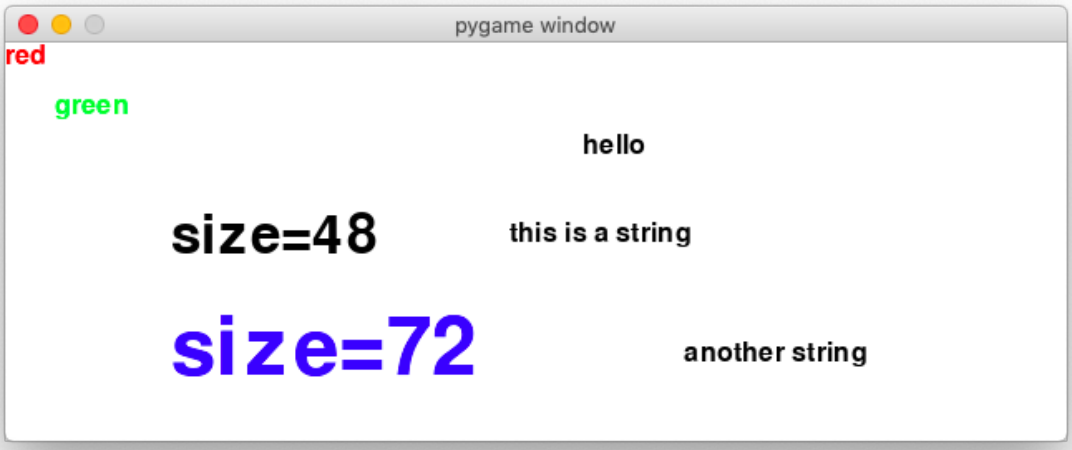
**on_key**(*event*)
Edit the text. Backspace to delete.

**render**()
Render the string and create an Surface object.

**class** draw6.**ListLabel**(*label*, *items*, *index=0*, ***kwargs*)
Draw a label with an item chosen from a list. Display 'Label = item'.

**class** draw6.**TextDemo**
Draw text in different sizes and colors.

**on_event**(*event*)
React to mouseclicks and keydown events.




## 3.6 Drawing shapes

The **pygame.draw** module has methods for drawing shapes to the screen:

- pygame.draw.rect
- pygame.draw.polygon
- pygame.draw.circle

The methods only draw the shape once. Using an object-oriented approach we are going to define classes for each shape. There is a common class which we call `Shape`

The class defintion begins with a couple of **class attributes**:

```
class Shape:
    """Base class for geometric shapes having size, color and thickness."""
    size = [50, 20]  # default size
    color = BLUE     # default color
    d = 0            # default thickness
    v = [0, 0]       # default speed
```

The constructor methods finds the attribute values for the shape either from the class attribute, or from the argument passed:

```
if pos != None:
    App.pos = list(pos)
self.pos = App.pos[:]

if size != None:
    Shape.size = list(size)
self.size = Shape.size[:]
App.pos[1] += Shape.size[1]

if color != None:
```

(continues on next page)

(continued from previous page)

```
    Shape.color = color
self.color = Shape.color

if d != None:
    Shape.d = d
self.d = Shape.d

if v != None:
    Shape.v = list(v)
self.v = Shape.v
```

At the end we define the enclosing rectangle which is used by some of the drawing methods. Finally the object is appended to the objects list:

```
self.rect = Rect(self.pos, self.size)
App.objects.append(self)
```

The `draw()` method needs to be instantiated separately for each object type:

```
def draw():
    pass
```

## 3.7 Rectangles ane Ellipses

This are two derived classes:

**class** pygamelib.**Rectangle**(***kwargs*)
Draw a rectangle on the screen.

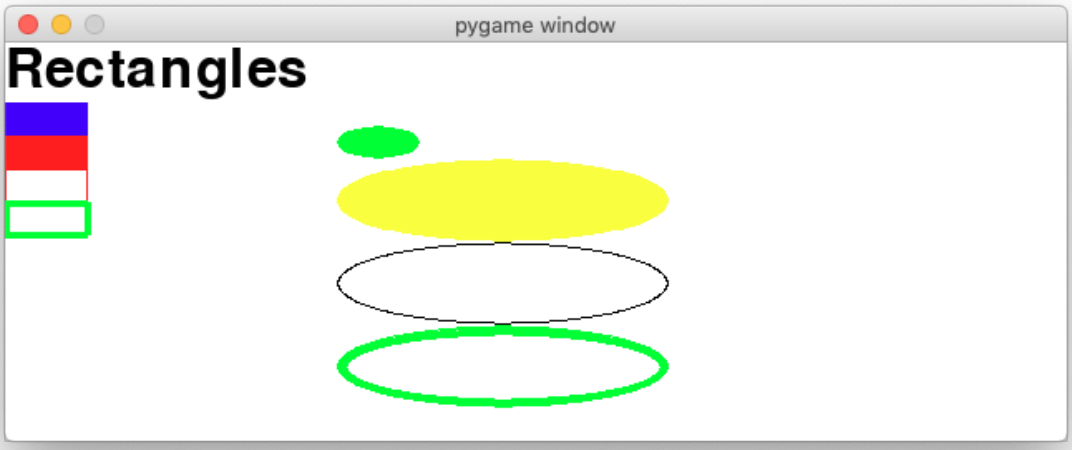
**draw**()
Draw the object to the screen.

**class** pygamelib.**Ellipse**(***kwargs*)
Draw an ellipse on the screen.

**draw**()
Draw the object to the screen.

**class** pygamelib.**Rectangle**(***kwargs*)
Draw a rectangle on the screen.

**draw**()
Draw the object to the screen.




## 3.8 Polygons, Arcs and Lines

This are some more derived classes:

**class** pygamelib.**Polygon**(*points=[]*, ***kwargs*)
Draw a polygon on the screen.

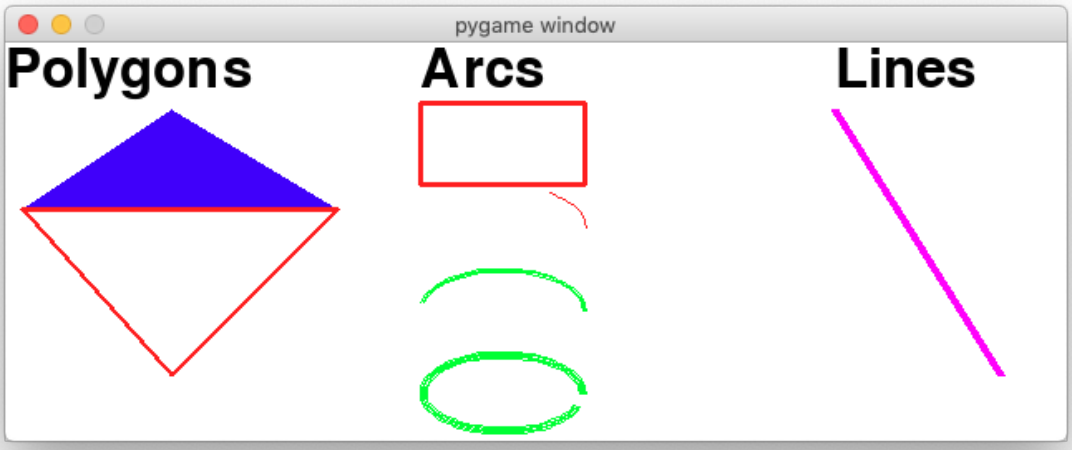
**draw**()
Draw the object to the screen.

**class** pygamelib.**Arc**(*start*, *stop*, ***kwargs*)
Draw an arc on the screen.

**draw**()
Draw the object to the screen.

**class** pygamelib.**Line**(*start*, *stop*, ***kwargs*)
Draw a line on the screen.

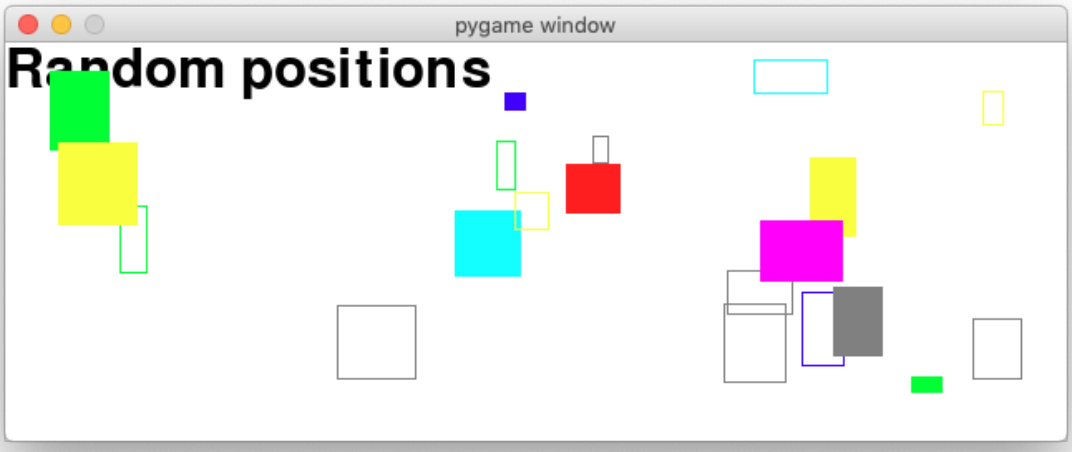
**draw**()
Draw the object to the screen.




## 3.9 Randomly moving shapes

In order to move the shapes, we add an `update()` method to `Shape`:

```
def update(self):
    self.pos[0] += self.v[0]
    self.pos[1] += self.v[1]
    if not 0 < self.pos[0] < App.screen.get_width()-self.size[0]:
        self.v[0] *= -1
    if not 0 < self.pos[1] < App.screen.get_height()-self.size[1]:
        self.v[1] *= -1
    self.rect.topleft = self.pos
```



CHAPTER 4

# Work with text

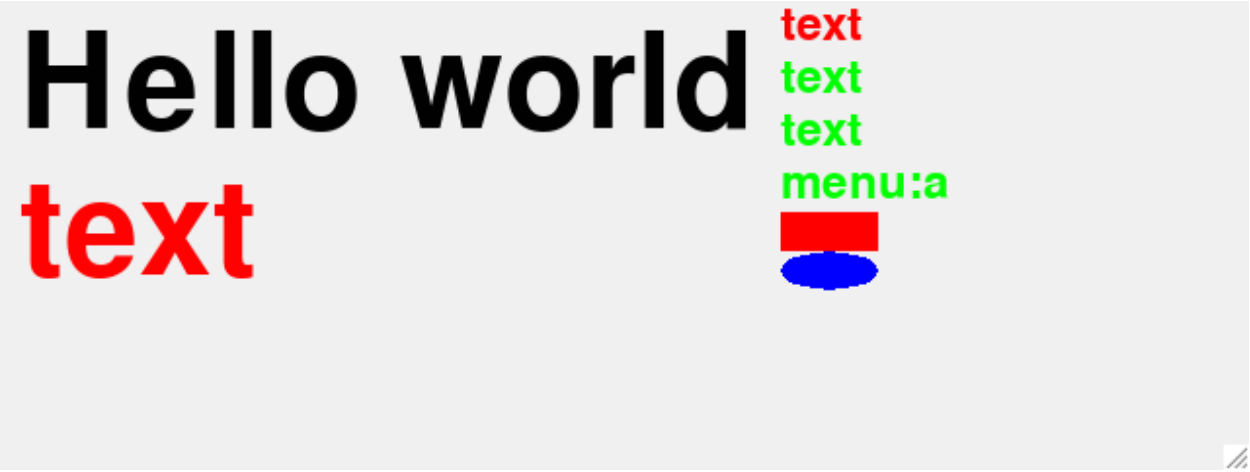
## 4.1 Draw Text with attributes

Text is placed on the screen as a **Text()** object. They are automatically positioned vertically. Text can have the attributes:

- color
- size
- position
- font

The next text object inherits the previous attributes.

**class** text1.**TextDemo**
Draw text in different sizes and colors.



## 4.2 Floating text

In this exemple words from a list float over the screen, with random speed vx and vy in the range (-3, ... +3), and starting at random positions.

**class** text2.**TextDemo2**
Draw text in different sizes and colors.


## 4.3 GUI

### 4.3.1 Shortcut keys

The simplest way to decode shortcut keys is to use a dictionary. This is fast, short and easy to extend. We define the following dictionary which uses as the key a simple **keyboard key code** or the combination of key and modifier, given as a tuple (k, m). The dictionary value is a string which is executable:

```
d = {
    K_a:'print("A")',
    (K_a, KMOD_LSHIFT):'print("shift+A")',
    (K_a, KMOD_LCTRL):'print("ctrl+A")',
    (K_a, KMOD_LALT):'print("alt+A")',
    (K_a, KMOD_LMETA):'print("cmd+A")',
    K_UP:'print("UP")',
    K_LEFT:'print("LEFT")',
}
```

Inside the `ShortcutDemo` class we implement the `on_event()`. If the event is a keydown event, we define local variables *k* and *m* for the key and modifier. If *k* is in the dictionary we execute the associated command string. Otherwise we execute look if the tuple *(k, m)* is in the dictionary:

```
def on_event(self, event):
    if event.type == KEYDOWN:
        k = event.key
        m = event.mod
        if k in d and m == 0 :
            exec(d[k])
        elif (k, m) in d:
            exec(d[k, m])
```

Define shortcut keys with modifiers.

**class** gui1.**ShortcutDemo**

**on_event**(*event*)
Implement a general-purpose event handler.

### 4.3.2 Buttons

Place clickable buttons on the screen.

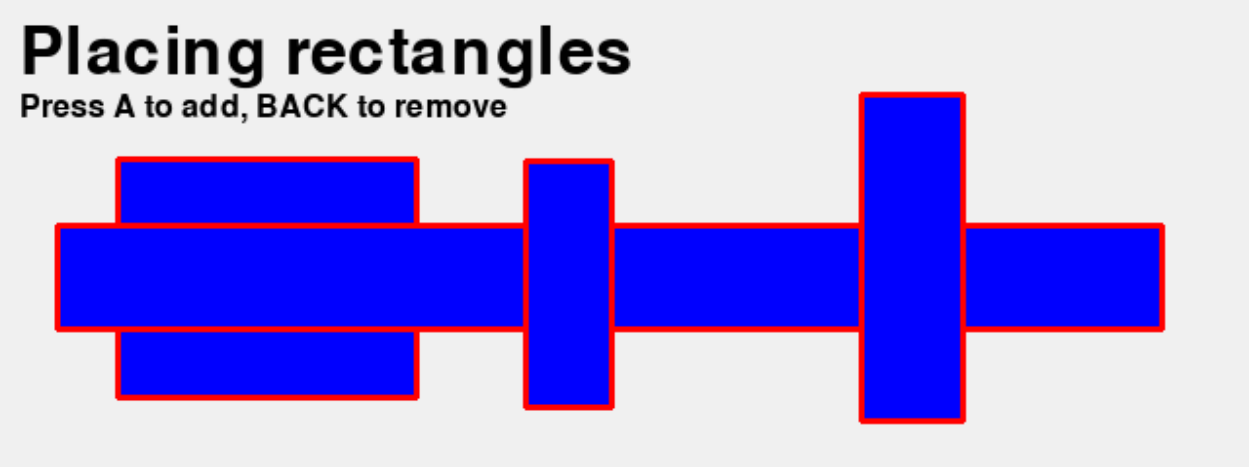
**class** gui2.**ButtonDemo**
Draw text in different sizes and colors.


### 4.3.3 Selecting objects

Editing graphical shapes. placing rectangles

**class** gui4.**GuiDemo**
Draw text in different sizes and colors.

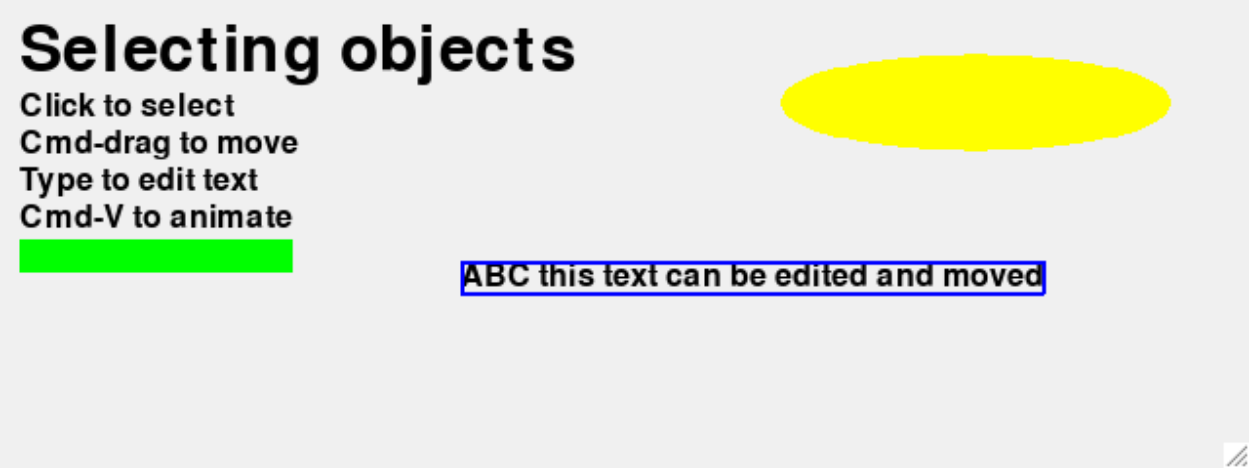
**on_event**(*event*)
Implement a general-purpose event handler.



### 4.3.4 Selecting objects

Selecting and editing objects.

**class** gui5.**GuiDemo**
Make a subclass of the App class.



CHAPTER 5

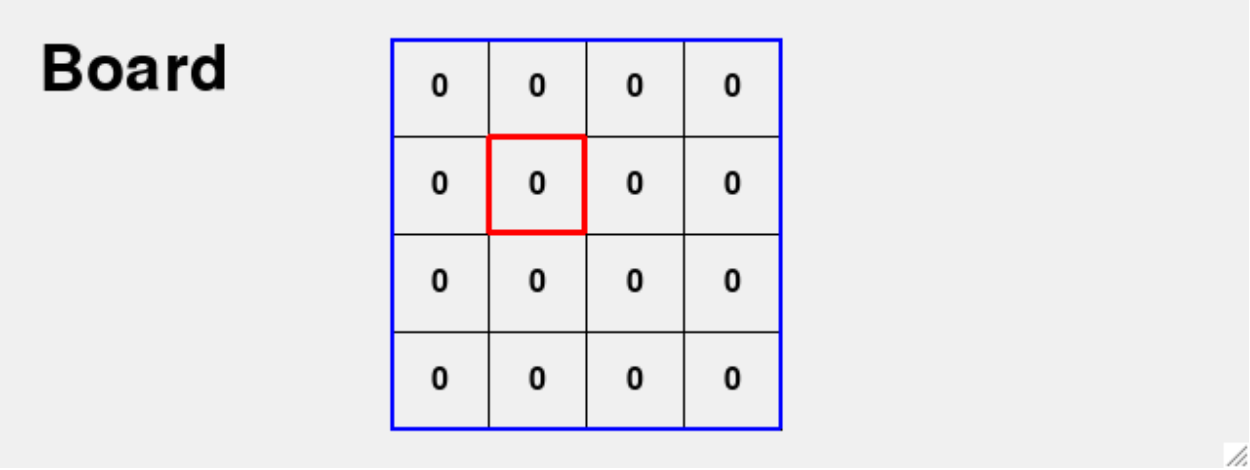
# Board Games

In this section we create the framework for board games. These games are based on a nxm grid. Each cell can have

- text
- color
- image

Draw a 4x4 board.

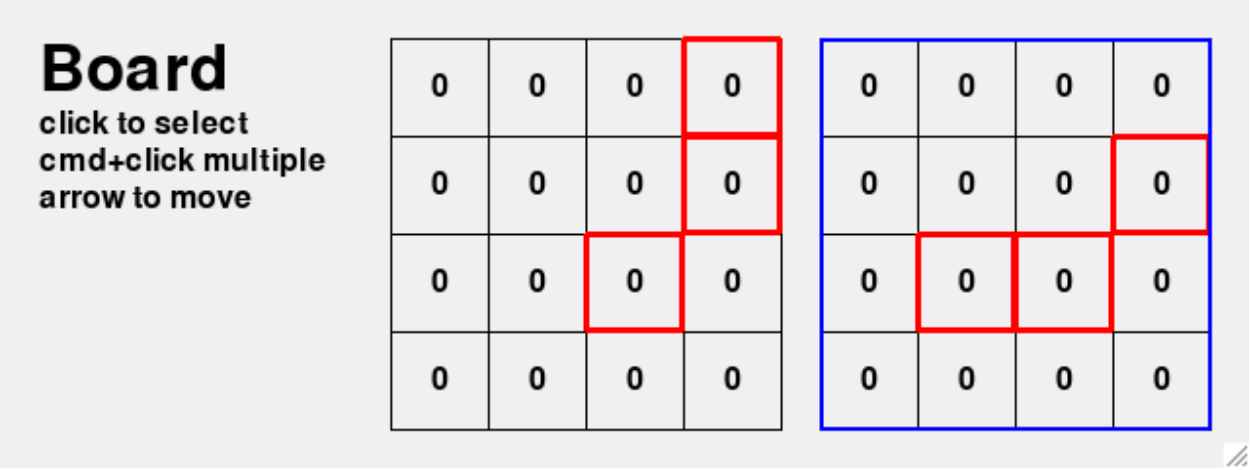
**class** board1.**BoardDemo**
Draw a playing board.




## 5.1 Selecting cells with the mouse

Place two 4x4 boards on the screen.

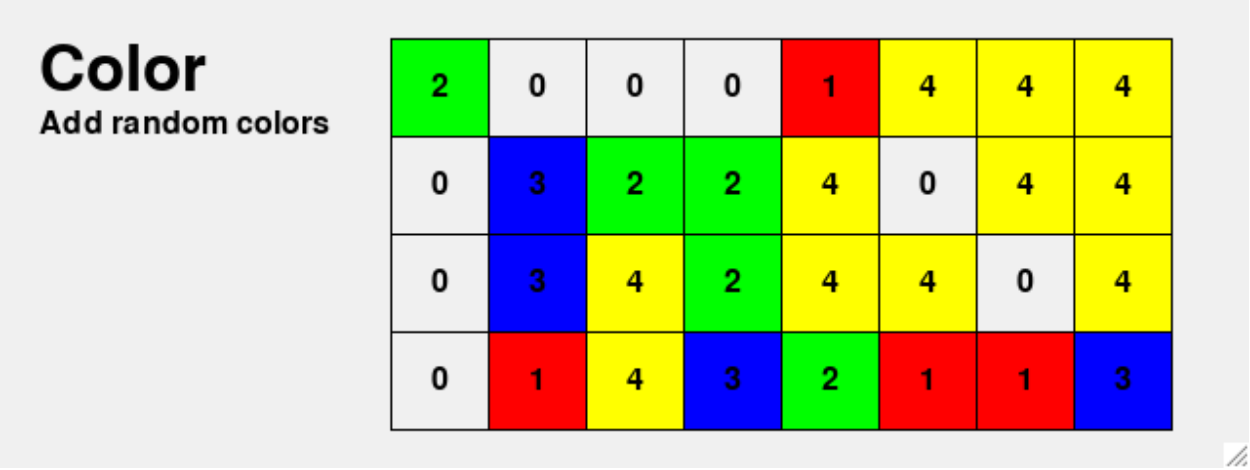
**class** board2.**BoardDemo2**
Draw two 4x4 boards and select cells with mouse click.




## 5.2 Adding background color

Add color to the cells.

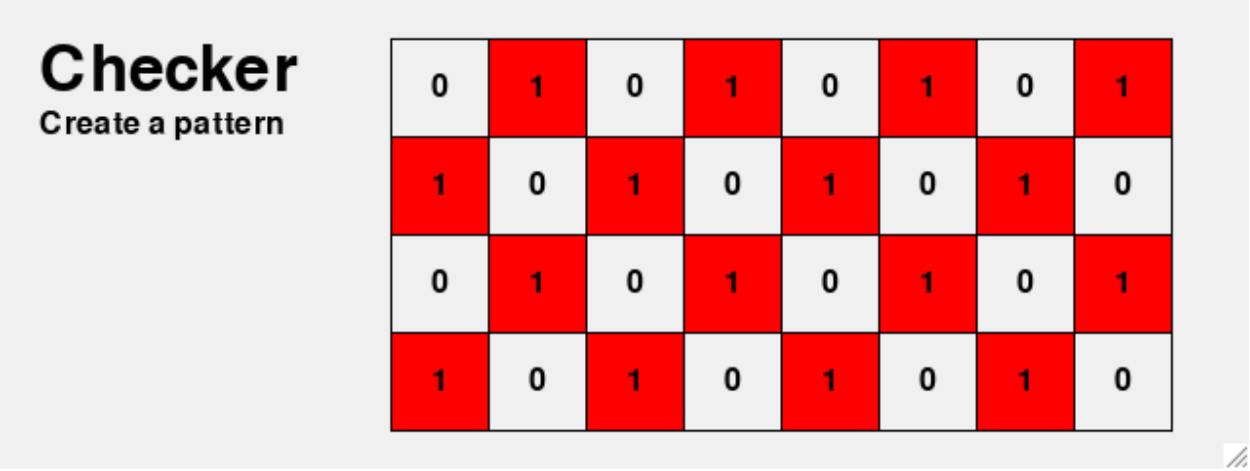
**class** board3.**BoardDemo**
Draw cells in random colors.




## 5.3 Create a checkerboard pattern

Create a checkerboard pattern.

**class** board4.**BoardDemo**
Calculate the color pattern.


Checker
Create a pattern
0 1 0 1 0 1 0 1
1 0 1 0 1 0 1 0
0 1 0 1 0 1 0 1
1 0 1 0 1 0 1 0

CHAPTER 6

# Sandbox

## 6.1 Domains

Domains have been introduced into sphinx to make it available for other languages than just Python. Domains can provide custom indeces (like the Python module).

**spam**(*eggs*)
**ham**(*eggs*)
Spam or ham the foo.

**filterwarnings**(*action*, *message*)

The function *spam()* does a similar thing.

The class *App* is always used to subclass a game application.

**pyfunc**()
Describes a Python function.

Reference to *pyfunc()* inmidst of text.

## 6.2 Cross-referencing syntax

`get()`

## 6.3 Directives

`Timer.`**repeat**(*repeat=3*, *number=1000*)
Describe the function.

`Timer.`**repeat**(*repeat=3*, *number=1000*)
Describe a method.

**number=1000**
Describe data.

**class App**
Describe class without parameters.

**run**()

Describe the method.

**class App**(*parameters*)
Describe class with parameters.

**objects**

Global class attribute.

**send_message**(*sender*, *recipient*, *message_body*[, *priority=1* ])
Send a message to a recipient

**Parameters**

- **sender** (*str*) – The person sending the message
- **recipient** (*str*) – The recipient of the message
- **message_body** (*str*) – The body of the message
- **priority** (*integer or None*) – The priority of the message, can be a number 1-5

**Returns** the message id

**Return type** int

**Raises**

- **ValueError** – if the message_body exceeds 160 characters
- **TypeError** – if the message_body is not a basestring

## 6.4 The math domain

$$e^{i\pi} + 1 = 0 \tag{6.1}$$

Euler's identity, equation (6.1), was elected one of the most beautiful mathematical formulas.

**html_sidebars = {** '**': ['globaltoc.html', 'sourcelink.html', 'searchbox.html'], 'using/windows': ['windowssidebar.html', 'searchbox.html'], }

## 6.5 The pygamelib module

Classes are listed in alphabetical order.

This pygame library provides useful classes for making games quickly.

**class** pygamelib.**Shape**(*pos=None*, *size=None*, *color=None*, *d=None*, *v=None*)
Base class for geometric shapes objects to place in the game. Shapes have the following attributes:

- size
- color

- thickness
- position, speed,
- friction, gravity

**draw**()
Draw the object to the screen.

**on_click**(*event*)
Handle a mouse click.

**on_key**(*event*)
Handle a key press event.

**select**()
Surround the object with a frame.

**update**()
Update the position of the object.

**class** pygamelib.**Rectangle**(***kwargs*)
Draw a rectangle on the screen.

**draw**()
Draw the object to the screen.

**class** pygamelib.**Ellipse**(***kwargs*)
Draw an ellipse on the screen.

**draw**()
Draw the object to the screen.

**class** pygamelib.**Polygon**(*points=[]*, ***kwargs*)
Draw a polygon on the screen.

**draw**()
Draw the object to the screen.

**class** pygamelib.**Arc**(*start*, *stop*, ***kwargs*)
Draw an arc on the screen.

**draw**()
Draw the object to the screen.

**class** pygamelib.**Line**(*start*, *stop*, ***kwargs*)
Draw a line on the screen.

**draw**()
Draw the object to the screen.

**class** pygamelib.**Text**(*str=”*, *size=None*, *color=None*, *bgcolor=None*, *font=None*, ***kwargs*)
Draw a line of text on the screen.

**render**()
Render the string and create an Surface object.

**draw**()
Draw the text on the screen.

**on_key**(*event*)
Edit the text. Backspace to delete.

**class** pygamelib.**ListLabel**(*label*, *items*, *index=0*, ***kwargs*)
Draw a label with an item chosen from a list. Display ‘Label = item’.

**next**()
Increment cyclically to the next item.

**class** pygamelib.**Button**(*msg*, *cmd*, *size=None*, *color=None*, *d=None*, ***kwargs*)
Draw Button on the screen.

**draw**()
Draw the object to the screen.

**on_click**(*event*)
Handle a mouse click.

**class** pygamelib.**Board**(*n=4*, *m=4*, *dx=50*, *dy=50*, *pos=None*, ***kwargs*)
Represents a nxm board with n rows and m columns. n, m number of cells (row, column) i, j index of cell (row, column) dx, dy size of cell x0, y0 origin of first cell

**draw**()
Draw the object to the screen.

**fill**(*index*, *color*)
Fill cell (i, j) with color.

**get_index**(*pos*)
Get index (i, j) from position (x, y).

**get_pos**(*index*)
Get position (x, y) from index (i, j).

**get_rect**(*index*)
Get the cell rectangle from the index (i, j).

**on_click**(*event*)
Add clicked cell to selection.

**on_key**(*event*)
Move the current cell if there is only one.

**class** pygamelib.**App**
Define the main application object and its methods.

**run**()
Run the main event loop. Handle the QUIT event and call `on_event`.

**on_event**(*event*)
Implement a general-purpose event handler.

**update**()
Update the screen objects.

**draw**()
Draw the game objects to the screen.

**capture**()
Save a screen capture to the directory of the calling class, under the class name in PNG format.

**find_objects**(*pos*)
Return the objects at position.

**select_objects**(*event*)
Select objects at position pos.

**do_shortcuts**(*event*)
Check if the key/mod combination is part of the shortcuts dictionary and execute it. More shortcuts can be added to the `self.shortcuts` dictionary by the program.

**where**()
Print the current module and path.

## 6.6 The App class

**class** pygamelib.**App**
Define the main application object and its methods.

**run**()
Run the main event loop. Handle the QUIT event and call `on_event`.

**on_event**(*event*)
Implement a general-purpose event handler.

**update**()
Update the screen objects.

**draw**()
Draw the game objects to the screen.

**capture**()
Save a screen capture to the directory of the calling class, under the class name in PNG format.

**find_objects**(*pos*)
Return the objects at position.

**select_objects**(*event*)
Select objects at position pos.

**do_shortcuts**(*event*)
Check if the key/mod combination is part of the shortcuts dictionary and execute it. More shortcuts can be added to the `self.shortcuts` dictionary by the program.

**where**()
Print the current module and path.

CHAPTER 7

# To do

## 7.1 capture to current repository (8 May)

get the name of the calling class:

```
name = type(self).__name__
```

save a file in the same folder of where it runs

> module = sys.modules['__main__'] path = os.path.dirname(module.__file__) filename = path + '/' + name + '.png' pygame.image.save(Game.screen, filename)

## 7.2 Resize the display

When risizing the display the objects scale and get out of ratio. However the image captured stays the same.

## 7.3 Combine left and right cmd/alt/shift keys

## 7.4 To do

- add serial number to captured images
- add click-sound to image capture
- add options menu (full screen, capture, sound, colors, wrap)
- use tab key to move between active objects
- subclass Board from Shape
- cmd+arrow to add speed to selected object

## 7.5 Done

- add background color for text
- mark active object with blue frame
- use arrow keys to move between active cell (**board**)

CHAPTER 8

# To do

- cmd-click : select an item
- alt-click : move selection
- Board games table: number, visibilty, text, color, image
- Selection : arrows and mouse clic
- Multiple selection : cmd+click, drag, (border margins)
- Games: Pong, Snake, Bricks, Space invader,
- Memory, 2048, Wordament
- Astroid, bullets, gravity
- Platformer games
- Dame, Go, Chess
- Button, ListMenu, TextMenu, CheckBox, Slider
- Select, move and resize an objet
- Edit a polygon
- Collision between objects
- Schedule : one-time and regular
- Music, sounds, images
- Object : x, y, z, scale, rotation, visible, anchor

object selection

- click outside to deselect
- TAB to select next (shift-TAB) to select previous
- draw rect to make multiple selection

CHAPTER 9

# Indices and tables

- genindex
- modindex
- search

# Python Module Index

# Index

## L



## N



## O



## P



## R



## S



## T



## U



## W