NOKIA Bell Labs

# Introduction to Pygame

François Durand

Python Workshop, 17 June 2020

# Sources

`https://fr.wikibooks.org/wiki/Pygame`, especially:

- `https://fr.wikibooks.org/wiki/Pygame/Introduction_%C3%A0_Pygame`
- `https://fr.wikibooks.org/wiki/Pygame/Introduction_au_module_Sprite`
- `https://fr.wikibooks.org/wiki/Pygame/Chimp_-_Ligne_par_ligne`

# What is Pygame?

- **Pygame** is a Python package that wraps the SDL library.
- **SDL (Simple DirectMedia Layer)** is a cross-platform development library written in C, designed to provide access to **multimedia**: 2D graphics, audio, keyboard, mouse and joystick.
- Pygame can be used to design **games** but not only: anything that makes uses of a graphical interface, sound, etc. And in particular, it can be used for **demos**.
- Pygame is **not** designed to make **GUIs** (with buttons, menus, etc). For that, you can use Tkinter.

# Minimal Example

Cf. `minimal_example.py`.

- `pygame.init()`: initialize all **modules** of Pygame, such as `pygame.display`, `pygame.event`, `pygame.image`, `pygame.time`, `pygame.sprite`, `pygame.mixer`, `pygame.key`, `pygame.mouse`, `pygame.font` and `pygame.transform`.
- `screen = pygame.display.set_mode(size)`: create a **Surface** object with the image to display.
- `pygame.event.get()`: catches **events** (quit, key strokes, etc).

   

# Basics: Bouncing Ball (1)

Cf. `bouncing_ball.py`.

- `clock = pygame.time.Clock()`: launch a **clock**.
- `ball = pygame.image.load("ball.gif").convert_alpha()`: create a **Surface** object with the image of the ball.
  - The `convert_alpha()` is not strictly necessary, but it is a good habit to have.
  - Depending on whether you want to deal with transparency, use `convert()` or `convert_alpha()`.
- `ball_rect = ball.get_rect()`: create a **Rect** object representing a rectangular zone.
  - A Rect is just a convenient way to store `x`, `y`, `width`, `height`.
  - But it also gives access to attributes (`left`, `right`, `top`, `bottom`, `center`, `topleft`...) and methods (`move`, `colliderect`...).
  - Default initialization: `x = y = 0`.

   

# Basics: Bouncing Ball (2)

Cf. `bouncing_ball.py`.

- `ball_rect.move_ip(ball_speed)`: modify `ball_rect` “in place” so that it contains the **new coordinates**.
  - Variant: `ball_rect = ball_rect.move(ball_speed)`.
- `screen.fill(black)`: to start preparing the new version of the image, we cover everything with a black background, but the new version of the image is not displayed yet (**double buffer**).
- `screen.blit(ball, ball_rect)`: **copy** the pixels of `ball` on `screen`, in the coordinates given by `ball_rect`.
- `pygame.display.update()`: actually **display** the prepared image.
  - Variant: `pygame.display.flip()` would do the same thing, but has less options.
- `clock.tick(100)`: **wait** 0.01 s.

# More on Surface and Rect: Bouncing Ball in a Box

Cf. `bouncing_ball_in_a_box.py`.

- `box_empty`: a Surface to store the picture of the **empty box**.
- `box`: a Surface where we draw **the box + the ball**.
- `ball_rect`: note that `ball_rect`, by itself, does not "know" that its coordinates are relative to the box. They are just **plain old numbers**.

   

# Sprite: Bouncing Balls

Cf. `bouncing_balls.py`.

- `class Ball(pygame.sprite.Sprite)`: a **Sprite** must have attributes `image` and `rect`, and a method `update`.
- `def update(self)`: the method **need not deal with display**, it just says what the sprite should do when an update is asked, typically once per frame.
- `all_sprites = pygame.sprite.RenderPlain([Ball() for _ in range(10)])`: generally, a Sprite should belong to one or more **Group** objects. There are several subclasses of Group with various capabilities: GroupSingle, RenderPlain, RenderClear, RenderUpdates. For most usages, RenderPlain is fine.
- `all_sprites.update()`: call the method `update` of all sprites.
- `all_sprites.draw(screen)`: call the built-in method `draw` of all sprites.

# Remark: Back to The Single Bouncing Ball Example

`bouncing_ball_sprite.py`.
This new version using Sprite is slightly longer than the original, but it is **cleaner** and therefore easier to **maintain** and **expand**.

   

# Collision and Sound: Bouncing Balls with Collision

Cf. `bouncing_balls_collision.py`

- `self.explode_sound = pygame.mixer.Sound('balloon_pop.wav')`: create a **Sound** object.
- `self.explode_sound.play()`: **play** the sound.
- `pygame.sprite.spritecollide(red_ball, yellow_balls, dokill=1)`: return a **list** of the yellow balls collided by the red ball.
  - `dokill=1`: remove the collided balls from the group.
  - By default, collision is determined only by the `rect` attribute of each sprite, but it is possible to specify a custom function for collision detection (next slide).

   

# Cleaner Collision: Bouncing Balls with Collision, revisited

Cf. `bouncing_balls_collision_circle.py`

- `self.radius = self.rect.width / 2`: our sprite will need a `radius` attribute.
- `pygame.sprite.spritecollide(red_ball, yellow_balls, dokill=1, collided=pygame.sprite.collide_circle)`: use the method `collide_circle` for collision detection.
  - Other methods are available, in particular `collide_rect_ratio`. Used with a ratio $< 1$, it gives often satisfactory results without needing to compute an exact “hit mask”.

   

# Group Collision: Bouncing Balls with Contamination

Cf. `bouncing_balls_contamination.py`.

- `pygame.sprite.groupcollide(healthy\_balls, contaminated\_balls, 0, 0)`: **list** the healthy balls that collide with contaminated balls.
- `healthy_balls.remove(ball)`: **remove** the ball from the group.
- `contaminated_balls.add(ball)`: **add** the ball to the group.

   

# Use the Keyboard: Collide Them All!

Cf. `collide_them_all.py`.

- `pressed_keys = pygame.key.get_pressed()`: a **tuple of Booleans**, indicating for each key if it is pressed or not.
- `pressed_keys[pygame.K_UP]`: the constant `pygame.K_UP` is the **number** representing the “up arrow” key.
- In `BallPlayer.update`, the syntax with multiple “ifs” accounts for the cases where several keys are pressed **simultaneously**.

   

# A Simple Game: Chimp (1)

Cf. `chimp.py`.

- `if not pygame.font`: when a module's import fails, its value is **None**.
- `def load_image`, `def load_sound`: it is convenient to have **import functions** that are a bit more sophisticated than the built-in ones.
- `image.set_colorkey(colorkey, pygame.RLEACCEL)`: use a color in the original image as representing **transparency**.
- `pos = pygame.mouse.get_pos()`: get the position of the **mouse**.
- `hit_box = self.rect.inflate(-5, -5)`: define a **slightly smaller** rect (to be used for collision detection).

   

# A Simple Game: Chimp (2)

Cf. `chimp.py`.

- `self.image = pygame.transform.flip(self.image, 1, 0)`: **flip the image** of the chimp when changing direction.
- `self.image = pygame.transform.rotate(self.original_image, self.rotation_angle)`: **rotate the image** of the chimp when hit.
- `pygame.display.set_caption("Monkey Fever")`: change the **title** of the window.
- `pygame.font`: this module is used to put **text** on a Surface.

   

# Summary: Pygame Modules

- `pygame.display`: the display window.
- `pygame.event`: catch events like `pygame.QUIT`, `pygame.KEYDOWN`, etc.
- `pygame.image`: images.
- `pygame.time`: clock.
- `pygame.sprite`: sprites.
- `pygame.mixer`: sounds.
- `pygame.key`: keyboard.
- `pygame.mouse`: mouse.
- `pygame.font`: text.
- `pygame.transform`: transform images.
- There are several others, such as `pygame.draw` to draw simple shapes like lines, polygons, etc.

   

# Thanks For Your Attention!

  


NOKIA